POLYTECH LILLE

Université de Lille 1 SCIENCES ET TECHNOLOGIES

POLYTECH LILLE

Rapport de projet final

P31 - Supervision des serveurs de la plateforme informatique

Septembre à Mars 2018

Taky DJERABA
Promo 2019

Resp. École : M. Thomas VANTROYS
M. Xavier REDON

Sommaire

Introduction

Le rapport qui suit fait état du travail réalisé pendant six mois dans le cadre du module de PFE. Ce projet vise à réaliser un tableau de bord affichant l'état des serveurs physiques et des machines virtuelles de la plateforme informatique ainsi qu'à mettre en place un système de sauvegarde automatique. L'idée est de faciliter la gestion de la plateforme informatique ainsi que de permettre aux personnes se chargeant de son administration d'être alerté en cas d'incident.

Ma principale motivation lors du choix de ce sujet fut ma volonté de travailler dans sur un projet ayant pour thématique les systèmes/Réseaux, ce qui, d'une part, rentre en accord avec mon expérience passé, et, d'autre part, correspond à mon projet professionnel.

Chapitre 1

Présentation du contexte

La plateforme informatique sur laquelle j'ai travaillé se compose de trois serveurs de visualisation : *bisban*, *sandbox* et *beta*. Ces derniers sont reliés à un stack de commutateurs donnant accès aux réseaux ADSL, SDSL et Renater. On trouve aussi le serveur de sauvegarde principal *baleine* ainsi qu'un serveur de sauvegarde secondaire en salle E306 : *cachalot*.

L'architecture en salle serveur se présente comme suit :

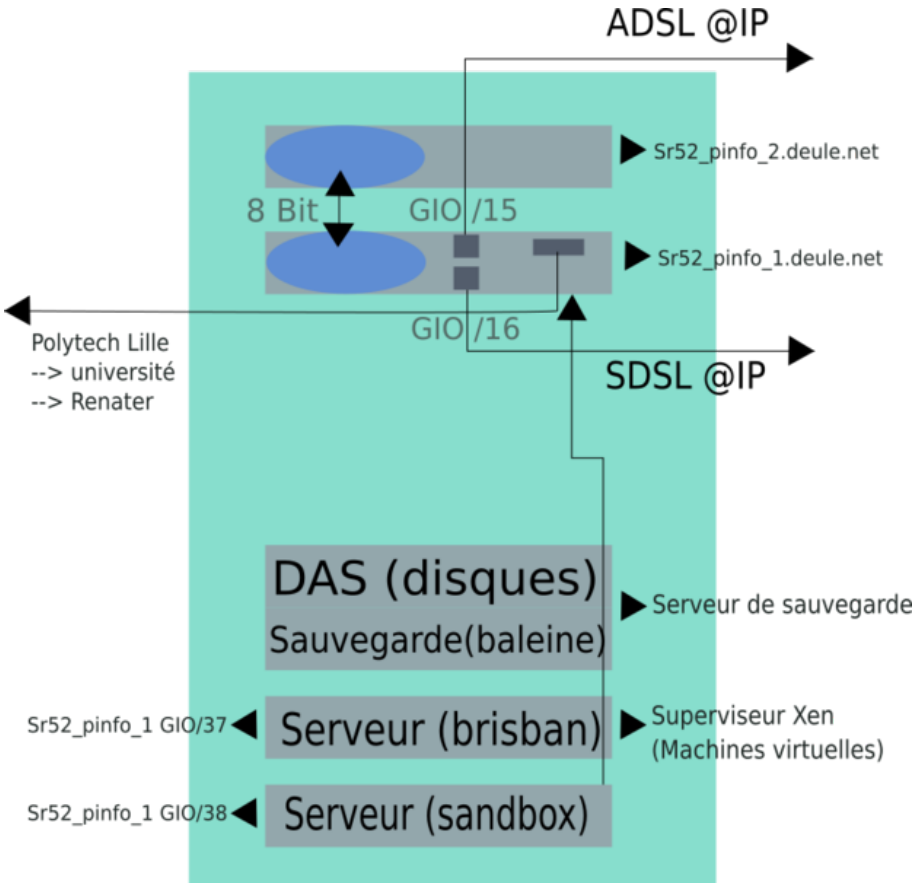

FIGURE 1.1 – Source : https ://wiki-ima.plil.fr/ - Onglet plateforme

Chapitre 2

Présentation du cahier des charges

Objectifs

Les éléments suivants doivent être monitorés :

— État de santé des machines physiques : température, état des disques, ...
— Occupation des machines physiques : utilisation CPU, utilisation espace disque, utilisation mémoire
— État de santé des connexions réseau : réseau Renater, réseau ADSL, réseau SDSL
— État de santé des machines virtuelles : temps d'exécution de chaque machine virtuelle
— Occupation des machines virtuelles : utilisation disque et mémoire
— État de certaines applications critiques :
    Date de validité des clefs DNSSEC
    Dates des dernières sauvegardes des machines virtuelles
— Vérification des certificats https de sites web.
— Éventuellement, l'affichage de la température en salle serveur

En ce concerne le système de sauvegarde automatique, les contraintes sont les suivantes :

— Le système de sauvegarde doit être simple et ne doit nécessiter aucune configuration.
— L'espace de stockage sur le serveur de sauvegarde est limité, ce qui exclue les backups complètes régulières.
— Le système doit idéalement permettre de garder en mémoire une backup datant d'un jour, d'une semaine et d'un mois.

Au cour de l'année, les tâches suivantes sont venu se greffer :

— Ajout d'une seconde baie DAS de disques 1To sur baleine.
— Construction d'un serveur de sauvegarde secondaire avec ajout de quatre baies DAS de disques 512Go (E306).
— Ajout sur le superviseur de scripts ansible pour le déploiement des agents Nagios/Prometheus sur les serveurs à surveiller.
— Reconstruction d'un gitlab en récupérant au maximum les archives du défaillant archives.plil.fr.

Environnement de travail

Afin de réaliser des tests de la solution de monitoring ainsi que du système de sauvegarde, plusieurs machines virtuelles ont été installés.

Sur le serveur de sauvegarde *baleine* :

— *supervise* : le serveur de supervision.
— *vmtest* : une machine virtuelle de test.
— *TestAnsible* : Afin de tester les scipts Ansible.
— *hyperion* : Afin de tester la sauvegarde d'une VM possédant un volume virtuel.

Sur le serveur de virtualisation *beta*, les VMs suivantes ont été installés afin de tester le système de sauvegarde sur des serveur distants :

— *dune*.
— *fondation*.

A cela, on peut ajouter la machine virtuelle *stargate*, installée sur le serveur *brisban*, et routé vers l'extérieur, afin de pouvoir tester les réseaux ADSL, SDSL et Renater ainsi que la date de validité des clés DNSSEC.

Adresses IP

```
2001:660:4401:6011:216:3eff:fe47:ba1f   stargate
172.26.64.13    supervise
172.26.64.15    baleine
172.26.64.16    vmtest
172.26.64.18    TestAnsible
172.26.64.22    cachalot
172.26.64.23    beta
172.26.64.24    dune
172.26.64.25    fondation
172.26.64.30    hyperion
```

Chapitre 3

Travail effectué

Le travail effectué au cours de l'année s'est, dans un premier temps, centré autour du cahier des charges initial, à savoir la mise en place d'une solution de supervision ainsi que la réalisation d'un système de sauvegarde. Par la suite, d'autres tâches sont venus se greffer, comme l'installation du serveur de sauvegarde secondaire ou la duplication de l'actuel serveur gitlab.

3.1 Choix des technologies

En ce qui concerne la surveillance des machines virtuelles et physiques de la plateforme informatique, j'ai eu la possibilité de me reposer sur des solutions de monitoring déjà existante comme Nagios ou Prometheus. Ces solutions ont été accompagnées d'agents tel que NRPE ou encore Node_exporter, afin de prélever des métriques sur serveur à monitorer.

3.1.1 Nagios Core & NRPE

La première solution à avoir été employée est Nagios Core. Ce choix à surtout été motivé par la popularité de cet outil ainsi que par l'importance de sa communauté, mettant à disposition plugins et addons sur la plateforme communautaire Nagios Exchange.

Nagios est un outil de supervision permettant la surveillance d'équipements, de services ou de réseaux. Ce dernier effectue des contrôles intermittents sur les hôtes et services spécifiés en utilisant des plugins externes qui retournent un statuts d'état à Nagios.

Nagios utilise deux types de supervision :

— La supervision active : le serveur de supervision va chercher l'information à surveiller.
— La supervision passive : le serveur de supervision reçoit l'information à surveiller.

En plus de l'utilisation de Nagios Core, l'agent Nagios Remote Plugin Executor (NRPE) a été installé sur les serveurs à surveiller afin de permettre l'exécution de scripts à distance par Nagios via le plugin check_nrpe.

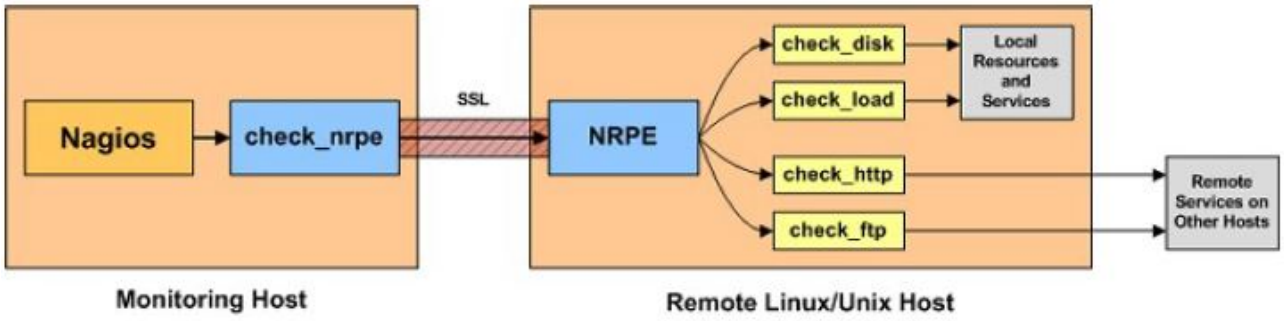

FIGURE 3.1 – Schéma de fonctionnement de Nagios+NRPE

Configuration de Nagios

La configuration de Nagios se fait de manière assez simple par la modification des fichiers de configuration présent dans le répertoire d'installation de Nagios.

L'arborescence dans ce répertoire se présente comme suit :

```
|-- cgi.cfg                  Config de l'application web de nagios
|-- htpasswd.users           Contient les identifiant pour la connexion à l'application web
, ici: nagiosadmin:1234
|-- nagios.cfg               Fichier de configuration de Nagios
|-- resource.cfg             Définition de chemin pour l'accès aux scripts lancé par Nagios
|-- objects                  Dossier contenant les fichier de configuration relatifs aux
services et hôtes
|    |-- commands.cfg        Fichier de configuration des commandes pouvant être lancé
par Nagios
|    |-- contacts.cfg        Configuration des contacts à avertir en cas de problème
|    |-- localhost.cfg       Configuration de la supervision du serveur hôte Nagios
|    |-- printer.cfg         Définition des imprimantes hôtes à superviser
|    |-- switch.cfg          Définition des switchs hôtes à superviser
|    |-- templates.cfg       Fichier contenant des templates pour la définition d'hôtes
ou de services
|    |-- timeperiods.cfg     Définition des configs de temps
|    |-- windows.cfg         Exemple d'un fichier de configuration pour une machine
Windows
|-- servers                  Dossier contenant les fichiers de configuration servant
à la définition des hôtes à monitorer et des services à exécuter
|    |-- baleine.cfg         Définitions pour l'hôte baleine
|    |-- vmBaleine.cfg       Définitions pour les VMs se trouvant sur baleine
```

Afin se surveiller une donnée, il est nécessaire de définir deux principaux élément :

— Un hôte à surveiller.
— Un service à mettre en place.

La définition de ces objets se fait au moyen de macros, dans des fichier de configuration situés dans un répertoire dédié : /usr/local/nagios/etc/servers.

Les définitions ci-dessous se situent dans le fichier /usr/local/nagios/etc/servers/baleine.cfg et permettent le lancement par Nagios du plugin check_load présent sur le serveur baleine.

On définie un hôte en lui attribuant une adresse :

```
define host
      use linux-box
      host_name baleine.insecserv.deule.net{
      alias Serveur de sauvegarde Baleine
      address 172.26.64.15
}
```

Et on met en place un service sur cet hôte :

```
define service{
     use generic-service
     host_name baleine.insecserv.deule.net
     service_description Current Load
     check_command check_nrpe!check_load
}
```

On peut voir dans la définition du service ci-dessus que Nagios exécute la commande "check_nrpe", laquelle prend en paramètre "check_load".

La définition de la commande check_nrpe est faite dans le fichier de configuration commands.cfg :

```
define command{
     command_name check_nrpe
     command_line /usr/lib/nagios/plugins/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
}
```

La commande check_nrpe va ensuite faire un appel au daemon NRPE exécuté sur la machine distante à monitorer pour lancer un plugin check_load.

Configuration de NRPE

L'installation de NRPE sur les machines à surveiller se fait de manière tout aussi simple en modifiant le fichier /usr/local/nagios/etc/nrpe.cfg :

On indique quels serveurs sont autorisés à communiquer avec NRPE :

```
allowed_hosts=127.0.0.1,172.26.64.13
```

Et on définit les commandes qui seront lancés :

```
command[check_load]=/usr/local/nagios/libexec/check_load -r -w .15,.10,.05 -c .30,.25,.20
```

L'exemple ci-dessus permet à NRPE de lancer la commande check_load lorsque le serveur de supervision Nagios en fait la demande.

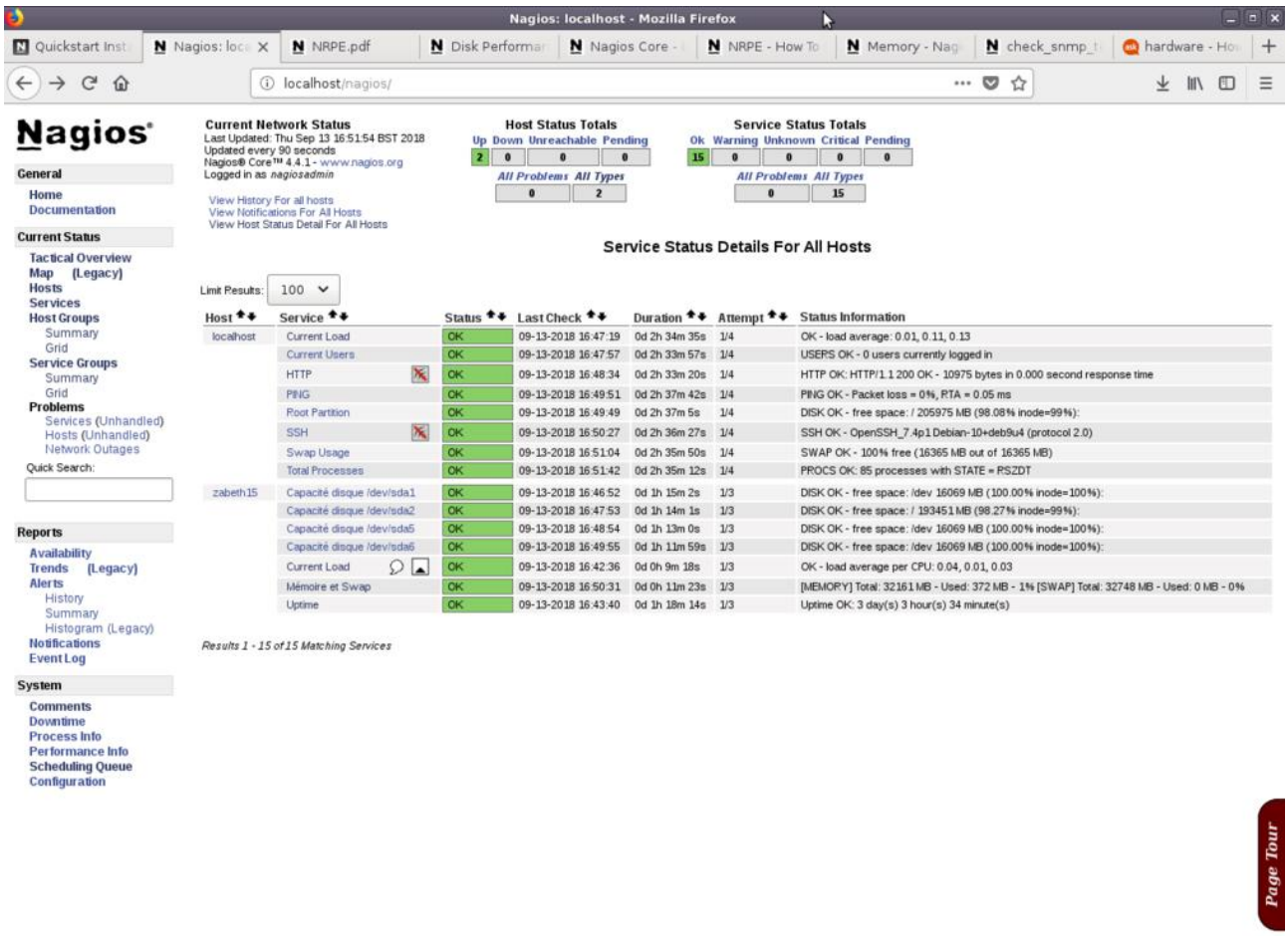

FIGURE 3.2 – Interface Nagios

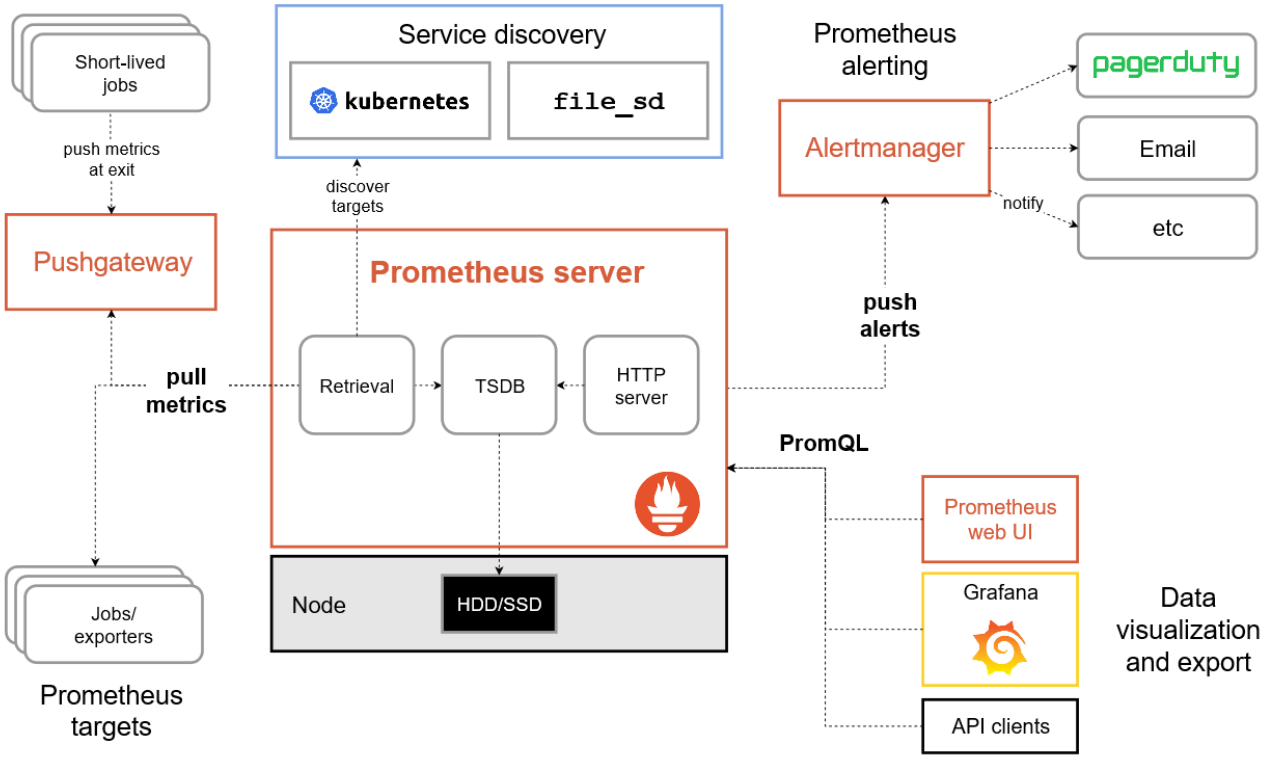

FIGURE 3.3 – Architecture de Prometheus

3.1.2 Prometheus & Node_exporter

La deuxième solution à avoir été testé est Prometheus. Il s'agit, comme pour Nagios, d'un outil de supervision, à la différence que ce dernier est beaucoup plus moderne. Prometheus est inclus avec une base de donné en série temporelle, ce qui lui permet de stocker des métriques de manière optimisé, et donc, de surveiller un serveur pendant un temps beaucoup plus long. A cela, on peut ajouter un fonctionnement extensible, du fait d'une compatibilité avec d'autre modules comme Kubernetes (permettant d'automatiser le déploiement de conteneurs), Grafana (Affichage de métriques), ou PagerDuty (système d'alerte), etc ...

De la même manière que pour Nagios, Prometheus a besoin d'agent afin de collecter différentes métriques. Je me suis tourné pour cela vers Node_exporter, un job exporter proposé par Prometheus, permettant d'exporter un grande nombre de métriques en utilisant des modules inclus dans Linux (CPU, Température, Mémoire, Espace Disque, Interfaces réseaux, etc ...)

Configuration de Prometheus

La seule configuration à effectuer pour mettre en place Prometheus est de lui indiquer les adresses IP des machines à superviser dans le fichier prometheus.yml :

```
scrape_configs:
 # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
 - job_name: 'node'
   static_configs:
           - targets: ['172.26.64.15:9100','localhost:9100','172.26.64.16:9100']
```

Configuration de Node_exporter

Node_exporter possède l'avantage d'être prêt à l'emploi, aucune configuration n'est requise si ce n'est pour mettre en place le module Textfile Collector.

Le module Textfile Collector permet de récupérer des métriques à partir de fichier en .prom. De cette manière, on peut exporter des données issues de scripts lancé périodiquement (grâce à un outil comme cron, par exemple).

Voici la configuration d'un job cron proposé sur le git du Node_Exporter :

```
echo job_bash > /path/to/directory/my_batch_job.prom.$$
mv /path/to/directory/my_batch_job.prom.$$ /path/to/directory/my_batch_job.prom
```

L'activation du module Textfile Collector de fait de la manière suivante :

```
./node_exporter --collector.textfile --collector.textfile.directory=/path/to/promfile
```

3.1.3 Grafana

Afin de pouvoir afficher des métriques issues à la fois de Nagios et de Prometheus, un serveur Grafana a été installé sur le serveur de supervision.

FIGURE 3.4 – Métrique issue du serveur de supervision Nagios

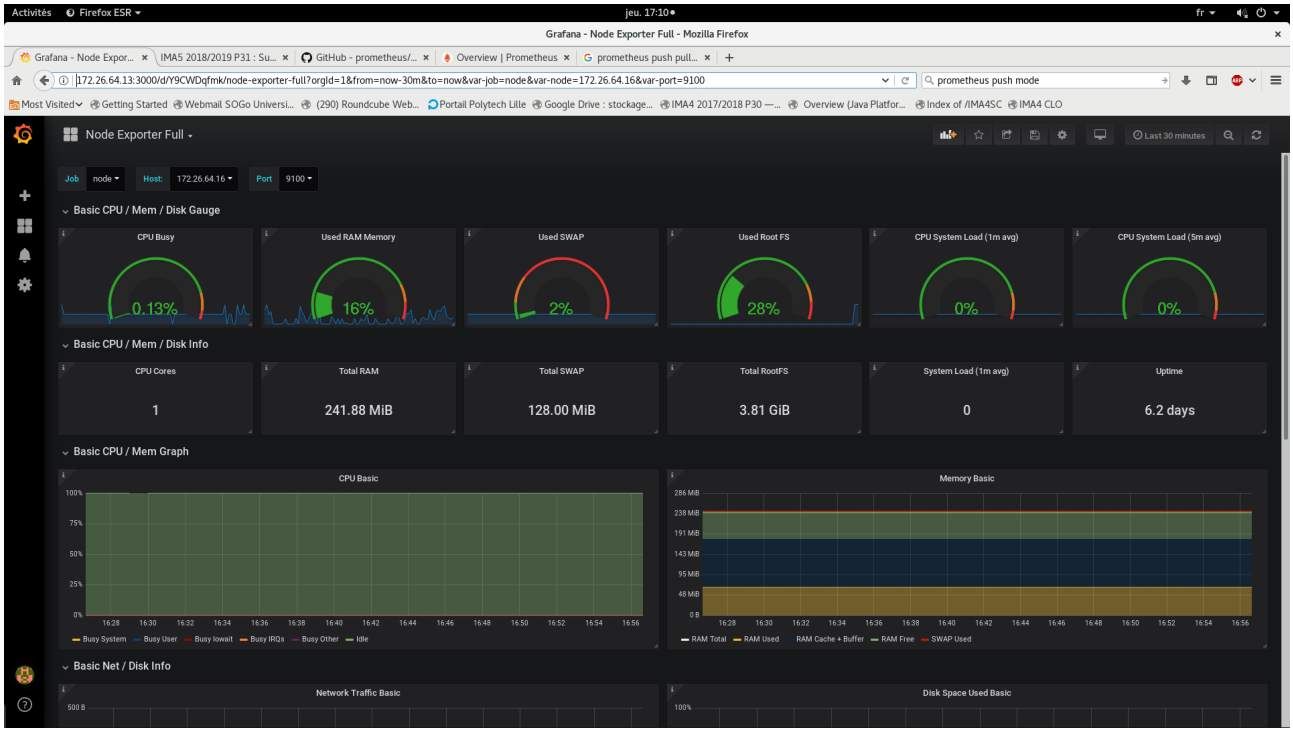

FIGURE 3.5 – Métriques issues de la VM "vmtest" affiché sur grafana

3.1.4 Résumé et choix de la technologie

Le travail qui a été effectué sur Nagios et Prometheus, m'a permis de réaliser une comparaison entres ces deux solutions, laquelle a révélé la supériorité de Prometheus sur de nombreux points.

Nagios

Les avantages :
- Une utilisation simple adapté pour une petite infrastructure.
- La possibilité d'avoir un retour sous la forme de texte.
- La possibilité d'intégrer des sondes écrites manuellement.
- Une grande communauté sur internet.

Les inconvénients :
- Une configuration demandant un peu trop de travail.
- Utilisation surtout adapté pour la récupération de données liées aux ressources d'un serveur (RAM, CPU, utilisation du disque, etc ...).

Prometheus

Les avantages :
— Une utilisation adapté pour les petites comme pour grandes infrastructures.
— Une base de donnée en série temporel, permettant une surveillance pendant une durée beaucoup plus longue (observation de tendances, utilisation adapté pour des serveurs d'application, etc ...)
— La possibilité d'intégrer des sondes écrites manuellement.
— Outil récent et régulièrement mis à jour.
— Fonctionnement extensible (Utilisation simple avec grafana par exemple).
— Configuration et installation très simple
— Plus récent, régulièrement mis à jour, beaucoup de contributeurs.
Les inconvénients :
— Données récupéré uniquement sous forme numérique.

En conclusion, moi choix quant à la technologie s'est tourné vers Prometheus en raison de sa simplicité et de sa plus grande efficacité.

3.2 Création des scripts

Pour la récupération de certaines données spécifiques, comme la récupération de la date de validité des clés DNSSEC, la vérification de l'état de santé des disques ou encore l'état des réseaux ADSL, SDSL et Renater, la création de scripts a été nécessaire.

3.2.1 Output de retour

En fonction de la solution à utiliser (Nagios ou Prometheus), la valeur de retour des scripts est amené à être différente :

Nagios

```
Format:
Message de retour | PerfData_label=value[UOM];[warn];[crit];[min];[max]


Exemple:
Key OK - ZSK Expiration Date = 2018/12/21 11:12:57 | zsk_expiration_date_date=20181221111257
```

En plus du message de retour, le script doit retourner un code de retour correspondant à un statut :

```
0  OK
1  WARNING
2  CRITICAL
3  UNKNOWN
```

Prometeus

```
Format (minimaliste):
# HELP Label Documentation sur la métrique.
# TYPE Label <Type> (Type: counter, gauge, histogram, summary, ou untyped)
Label <Donnée>


Exemple:
# HELP Disque_0 État de santé du disque présent sur le slot 0 sur serveur baleine.
# TYPE Disque_0 untyped
Disque_0 0
```

3.2.2 Date de validité des clés DNSSEC

Préalablement à la rédaction du script, du temps a été consacré à étudier le fonctionnement du protocole DNSSEC, afin de déterminer quelle donnée critique devra être surveillé. Il a finalement été convenu avec le tuteur de projet que la donnée à récupérer est la date de validité des signatures de l'enregistrement DNSKEY. Un script utilisant l'outil *dig* a donc finalement été rédigé.

Le résultat de la commande dig plil.fr RRSIG +multi donne le résultat suivant :

```
plil.fr. 21599 IN RRSIG DNSKEY 5 2 259200 (
                        20181221111257 20181121110621 40768 plil.fr.
                        TWruW7vtrja1i37zJw/egTbLCPvlw5DTVDvEQf7qIPu0
                        P8eKTw0dD3N+JXGU3k1mmvidw+Ljy7YeqZn9A1ZbR1Db
                        qfQnXk6sg5MBRS0Xw3QFVB61cvnETz36bsZQvd7fO9ny
                        KMWpviQOEfjWbZaZk5vbDZXbgFrksj7Mt/55zMvyMFvJ
                        I5fAv+mlo5RGk8M/AUJ20TaVjUOfe/p9qTlDJoEFSjeW
                        A6qN7tFjn7qIvgHLhcghguVBsn7nogxzX9gIZ0uvZSFz
                        xC+xQrn8wEOISSZJWm9ZONwJuHcT8oIgjcSqnDx4rmcu
                        pF3YyB2OYTcxa9JT+Hx8wX0swoc0P+B/Nw== )
plil.fr. 21599 IN RRSIG DNSKEY 5 2 259200 (
                        20181221111257 20181121110621 51828 plil.fr.
                        Oc+8B84qbhy4lh2Uka1bkJnYJhQFFSZO3AfxW6YPbQha
                        vsVd9ozGc3Zbs3jumVm6TLK5sC8B+l24NnBeGD9QCiG4
                        29RbsRMyOi59YaemTA42BRuZCKJMWvJn6e/0FCyvTc2T
                        w/we6MWDXcRoIAqbCD+79Bniq+sbt4pBRU2zrWU= )
```

Les Hashs ci-dessus représentent les enregistrements RRSIG de type DNSKEY des clés ZSK et KSK. Le premier champ à l'intérieur des parenthèse représente la date de validité de la signature (voir figure 3.6 pour plus d'infos).

L'idée pour le script consiste donc à récupérer la date d'expiration et à la parser. La première étape consiste à récupérer l'ID des clés ZSK ou KSK. En raison du roulement des clés, il se peut que deux clés d'un même type soit enregistré sur un domaine.

```
dnssec-tools.org.       21 IN DNSKEY 256 3 5 (
                                AwEAAde0IFUPiuDwEQM8vGlR436nb+TzGLlMWxzMmWDP
                                mHfWMPv8OXgZafScErgijGRnwPfv+t9irTUsX4dkvJie
                                vkV549mtDwhXLb9Vyg9C5JsoqOBq//QJvZXEZHtcHPoQ
                                9tIGVt8W9uctLUKMsAdAVaozfMxl8CuLWjqkL2Gq57HP
                                ) ; ZSK; alg = RSASHA1; key id = 19221
dnssec-tools.org.       21 IN DNSKEY 256 3 5 (
                                AwEAAfDcFXK9ef/BppquoTm7LSk2rIE5x9LPu+WplVEA
                                To9i7OhqxxPy6K6uIomxLNtZAlcDbUybQEeopWhglovk
                                odkvRY72q6ZUxW1vwzBAvmDLZ7dfTjPa0LYKVQ7qsf7Q
                                jFBiyPkWARSAHH2xqBATry25ft8j9O9ULX4/DYdYPq9D
                                ) ; ZSK; alg = RSASHA1; key id = 3147
dnssec-tools.org.       21 IN DNSKEY 257 3 5 (
                                AwEAAazJbvxfEciFb3LmGsddun82dGsrZj4YpCvn8qHI
                                KRQ2c0Hra0LnfhOK09YKG7B9eey3aWzOKWUa50fv/0sl
                                8npI7/5tgWCQFBpIdqcDxoJxtvtFlrUlOpRQz45aFE3x
                                FJoquCfPGYgaNSZu5VGED4qAsZqxGXLvDU8SDX76Eo3G
                                oLBqYJGXG0inRVAQViFNk9ovtA2kZSP4oHTsWBCcnKDx
                                8CRZhF9jrbplzUdW7ahbTRaYv6tPYaKd7nzvH8GksO9b
                                9t5EJ1o1BhL6cDOHXOrkL+Twhc1Mwg5jBhYWH6r1LaTX
                                w2NQN1hAQzGrAZPt13kP0KQp/ybGhoKtunLxVsFiiDb6
                                4HQs+cRiw2wSf/vYaNVAiYesrMHTpq5BLeWTVrMLk2Kt
                                NtQMV972pOKipT9UN7At7Sugdpbm1g7jQ8G3eKP6iRg5
                                YAOPjbuXFNYjmKG9fMTjQGgs5IHbDVe/W3mPHYS+1970
                                AmNX/momejvYG8NlQykZdrcqPUw5dSAOmUnbehl3wzJJ
                                HIR59FaxeG9bO8vrFBlaRapz1eMtrZbqkYV7P8PnUP/3
                                i7WVGZO5AJxPpxpHaSNbd63d575pQKMQxEU7dPPfQsq8
                                wnNxcCHDixb6EetVcfv9Id9up4v9t0O33O4562kN2S9/
                                cwPOOlnobJ/g597cU2sF5Pmxn+r7
                                ) ; KSK; alg = RSASHA1; key id = 34816
```

On voit dans l'exemple ci-dessus qu'il y a deux clés ZSK. Le script enregistre donc les IDs des deux clés afin d'identifier laquelle est utilisé pour signer l'enregistrement RRSIG de type DNSKEY (une seule clé est utilisé pour signer les enregistrements d'un même type). Une fois l'ID de la clé ZSK ou KSK récupéré, le script cherche parmi les enregistrements RRSIG la date de validité correspondante et la retourne.

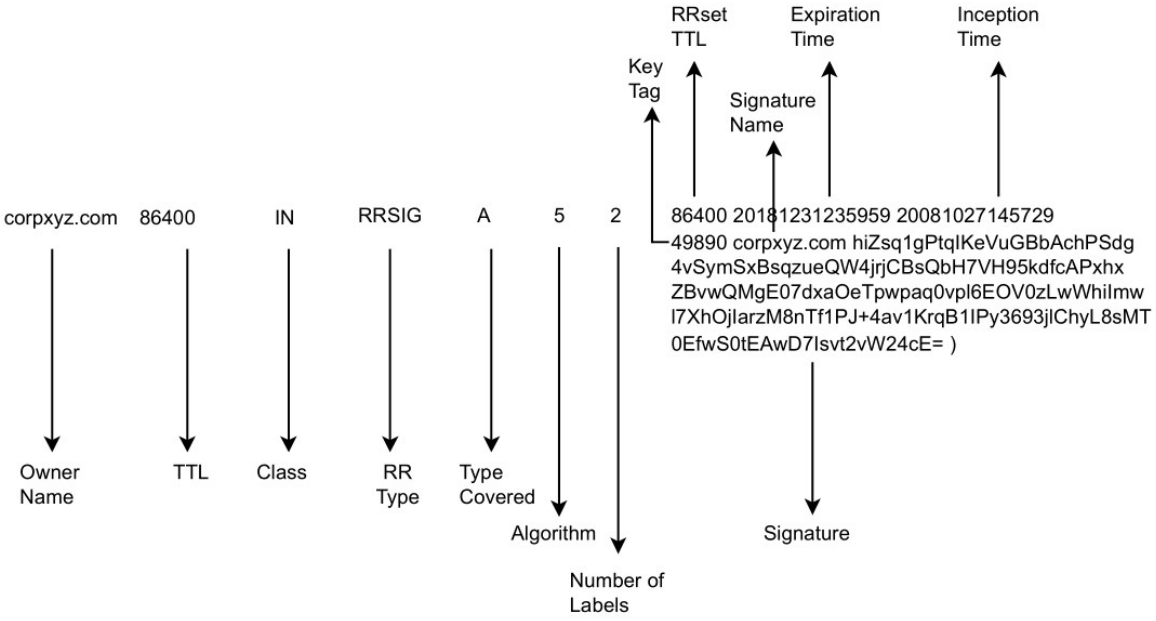

FIGURE 3.6 – Exemple d'un enregistrement RRSIG de type A

L'utilisation du script se fait de la manière suivante :

```
./check\_dnssec.sh <domain> <key\_type>
key_type: KSK ou ZSK
```

Le résultat suivant est obtenu (ici formaté pour une utilisation avec Nagios) :

```
root@stargate:~/PFE_IMA_5 ./check_dnssec.sh plil.fr ZSK
Key OK - ZSK Expiration Date = 2018/12/21 11:12:57 | 20181221111257
```

3.2.3 État des disques du serveur de sauvegarde baleine

Afin de vérifier l'état de santé des disques durs, un script fonctionnant grâce à l'outil smartctl à été mis en place.

Exemple de résultat pour la commande : smartctl -ad cciss,[0-7] /dev/cciss/c0d1 :

```
=== START OF INFORMATION SECTION ===
Vendor:               HP
Product:              DG146A4960
Revision:             HPDB
User Capacity:        146 815 737 856 bytes [146 GB]
...
=== START OF READ SMART DATA SECTION ===
SMART Health Status: OK
Current Drive Temperature:     35 C
Drive Trip Temperature:        70 C
```

```
Manufactured in week 34 of year 2007
Specified cycle count over device lifetime:  50000
Accumulated start-stop cycles:  118
Elements in grown defect list: 0
Vendor (Seagate) cache information
  Blocks sent to initiator = 6238686454743040
Error counter log:
           Errors Corrected by           Total   Correction     Gigabytes    Total
               ECC          rereads/    errors   algorithm      processed    uncorrected
           fast | delayed   rewrites  corrected  invocations   [10^9 bytes]  errors
read:          0        0         0         0          0          0,000           0
write:         0        0         0         0          0          0,000           0
Non-medium error count:      403
SMART Self-test log
Num  Test              Status                 segment  LifeTime  LBA_first_err [SK ASC ASQ]
     Description                              number   (hours)
# 1  Background long   Completed                  -    1819                 - [-   -    -]
# 2  Background short  Completed                  -    1819                 - [-   -    -]
```

On observe, en plus d'une indication sur l'état de santé du disque (SMART Health Status : OK), diverses informations telles que la température ou encore les dysfonctionnements (Elements in grown defect list).

Pour les disques de la baie DAS, on récupère des informations plus détaillées sur l'état de santé des disques, comme la valeur *Current_Pending_Sector*, qui indique le nombre de secteurs du disque qui ne peuvent plus être lus et qui sont en attente d'être remappé.

Commande : smartctl -ad cciss,[8-19] /dev/cciss/c0d1 :

```
Vendor Specific SMART Attributes with Thresholds:
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  1 Raw_Read_Error_Rate     0x002f   200   200   051    Pre-fail  Always       -       0
  3 Spin_Up_Time            0x0027   253   253   021    Pre-fail  Always       -       1150
  4 Start_Stop_Count        0x0032   100   100   000    Old_age   Always       -       6
  5 Reallocated_Sector_Ct   0x0033   200   200   140    Pre-fail  Always       -       0
  7 Seek_Error_Rate         0x002f   200   200   051    Pre-fail  Always       -       0
  9 Power_On_Hours          0x0032   034   034   000    Old_age   Always       -       48633
 10 Spin_Retry_Count        0x0033   100   253   051    Pre-fail  Always       -       0
 11 Calibration_Retry_Count 0x0033   100   253   051    Pre-fail  Always       -       0
 12 Power_Cycle_Count       0x0032   100   100   000    Old_age   Always       -       6
184 End-to-End_Error        0x0033   100   100   097    Pre-fail  Always       -       0
187 Reported_Uncorrect      0x0032   100   100   000    Old_age   Always       -       0
188 Command_Timeout         0x0032   100   100   000    Old_age   Always       -       0
190 Airflow_Temperature_Cel 0x0022   067   050   045    Old_age   Always       -       33
192 Power-Off_Retract_Count 0x0032   200   200   000    Old_age   Always       -       5
193 Load_Cycle_Count        0x0032   200   200   000    Old_age   Always       -       0
194 Temperature_Celsius     0x0022   117   100   000    Old_age   Always       -       33
196 Reallocated_Event_Count 0x0032   200   200   000    Old_age   Always       -       0
197 Current_Pending_Sector  0x0032   200   200   000    Old_age   Always       -       0
198 Offline_Uncorrectable   0x0030   200   200   000    Old_age   Offline      -       0
199 UDMA_CRC_Error_Count    0x0032   200   200   000    Old_age   Always       -       0
200 Multi_Zone_Error_Rate   0x0008   200   200   000    Old_age   Offline      -       0
```

L'idée pour la réalisation de ce script fut donc de reprendre cette commande afin d'avoir un retour sur l'état de santé du disque.

Le script vérifie dans un premier temps l'état de santé général du disque en vérifiant le résultat du test SMART (PASSED ou OK) et, dans un second temps, vérifie la présence ou non dysfonctionnement (Pre-Fail, Elements in grown defect list).

Selon l'état de santé du disque, le script retourne les outputs suivants :

NB : Le chiffre passé en argument représente le numéro de slot du disque (de 0 à 7, les disques sur le serveur baleine, et de 8 à 19, les disques durs de la baie DAS).

```
root@baleine:/usr/local/nagios/libexec# ./smart_check_disk.sh 0
Utilisation ./smart_check_disk.sh [0-19]
Résultats possible:
DISQUE OK - Aucune erreur
DISQUE OK - Attention, disque endommagé
ERREUR DISQUE
```

3.2.4 État des réseau ADSL, SDSL et Renater

Afin de pouvoir tester l'état de santé des réseaux ADSL, SDSL et Renater, une machine virtuelle nommé *stargate* à été mise en place sur le serveur *brisban*.

La machine virtuelle est routé aux trois serveurs via les interfaces suivantes :

— eth0 - 192.168.1.111 - Réseau ADSL
— eth1 - 2001 :660 :4401 :6011 :216 :3eff :fe47 :ba1f - Réseau Renater
— eth2 - 2001 :7a8 :116e :47 :216 :3eff :fe47 :ba20 - Réseau SDSL
— eth3 - Réseau Renater

Table de routage ipv4 :

```
default via 192.168.1.253 dev eth0
192.168.1.0/24 dev eth0  proto kernel  scope link  src 192.168.1.111
```

Table de routage ipv6 :

```
2001:660:4401:6011::/64 dev eth1  proto kernel  metric 256  expires 923sec
2001:660:4401:6048::/64 dev eth3  proto kernel  metric 256  expires 963sec
2001:7a8:116e:47::/64 dev eth2  proto kernel  metric 256  expires 982sec
fe80::/64 dev eth1  proto kernel  metric 256
fe80::/64 dev eth2  proto kernel  metric 256
fe80::/64 dev eth3  proto kernel  metric 256
fe80::/64 dev eth0  proto kernel  metric 256
default via fe80::211:5dff:fef2:5400 dev eth1  proto ra  metric 1024
expires 1723sec hoplimit 64
default via fe80::4e4e:35ff:fe5e:b943 dev eth2  proto ra  metric 1024
expires 1782sec hoplimit 64
default via fe80::211:5dff:fef2:5400 dev eth3  proto ra  metric 1024
expires 1763sec hoplimit 64
```

Pour la réalisation du script permettant de récupérer l'état de santé des réseaux ADSL, SDSL et Renater, la "difficulté" a surtout résidé dans le fait de forcer l'utilisation d'une interface pour pouvoir tester un réseau en particulier.

Étant donné que l'interface eth0 est la seule à posséder une adresse en ipv4, un simple ping ipv4 vers des machines distantes permet de tester l'état du réseau ADSL.

Pour tester les réseaux SDSL et Renater, il faut faire en sorte d'utiliser uniquement l'une de ces interfaces. On peut observer qu'en pingant une adresse en ipv6 depuis l'interface eth2, les paquets sont envoyés depuis l'adresses ipv6 de l'interface eth1 (Renater) :

```
root@stargate:~# ping6 -I eth2 2001:4860:4860::8888
ping6: Warning: source address might be selected on device other than eth2.
PING 2001:4860:4860::8888(2001:4860:4860::8888) from 2001:660:4401:6011:216:3eff:fe47:ba1f
eth2: 56 data  bytes
64 bytes from 2001:4860:4860::8888: icmp_seq=1 ttl=55 time=5.13 ms
64 bytes from 2001:4860:4860::8888: icmp_seq=2 ttl=55 time=5.55 ms
64 bytes from 2001:4860:4860::8888: icmp_seq=3 ttl=55 time=5.19 ms
```

Afin de résoudre ce problème, on spécifie explicitement l'adresse IP source dans les paramètres du ping6.

Dans l'exemple ci-dessous, on tente de pinger un serveur distant depuis l'interface eth2 (réseau SDSL). On constate que les echo requests sont toujours envoyés via l'interface eth1 :

```
root@stargate:~# tcpdump -i eth1 icmp6
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
16:36:49.282006 IP6 2001:7a8:116e:47:216:3eff:fe47:ba20 > google-public-dns-a.google.com:
ICMP6, echo request, seq 15, length 64
16:36:50.283237 IP6 2001:7a8:116e:47:216:3eff:fe47:ba20 > google-public-dns-a.google.com:
ICMP6, echo request, seq 16, length 64
16:36:51.284455 IP6 2001:7a8:116e:47:216:3eff:fe47:ba20 > google-public-dns-a.google.com:
ICMP6, echo request, seq 17, length 64
```

Les echo reply sont cependant reçus sur l'interface eth2, via le réseau SDSL :

```
root@stargate:~# tcpdump -i eth2 icmp6
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth2, link-type EN10MB (Ethernet), capture size 262144 bytes
16:32:17.341902 IP6 google-public-dns-a.google.com > 2001:7a8:116e:47:216:3eff:fe47:ba20:
ICMP6, echo  reply, seq 4, length 64
16:32:18.343265 IP6 google-public-dns-a.google.com > 2001:7a8:116e:47:216:3eff:fe47:ba20:
ICMP6, echo reply, seq 5, length 64
16:32:19.344553 IP6 google-public-dns-a.google.com > 2001:7a8:116e:47:216:3eff:fe47:ba20:
ICMP6, echo reply, seq 6, length 64
```

Des tests complémentaires ont ensuite été fait directement en salle serveur en débranchant les liaisons ADSL, SDSL et Renater pour simuler un dysfonctionnement.

Un script renvoyant l'état des réseaux ADSL, SDSL et Renater a finalement été rédigé :

```
usage: ./check_ping.sh <Network>
Network : choose: ADSL, SDSL or Renater
root@stargate:~/PFE_IMA_5# ./check_ping.sh ADSL
Connexion ADSL OK - 10/10 paquets reçus
```

3.2.5 Autres scripts

En plus de ce qui a déjà été évoqué, du temps a été passé sur des solutions qui n'ont finalement pas été retenues.

Script de vérification de température

Un script permettant de relever la température d'un CPU a été rédigé en reprenant un plugin déjà existant. Des modifications ont été faite afin de pouvoir avoir un retour dans le cas d'un serveur possédant plusieurs processeurs, comme c'est le cas sur baleine.

La solution a finalement été abandonné après l'installation de Prometheus puisqu'une sonde parvenant au même résultat est déjà présente dans le Node_eporter.

Script pour sauvegarde automatique

Un script permettant de sauvegarder automatiquement l'image d'une machine virtuelle a aussi été rédigé. Le script a par la suite été abandonné après la mise à disposition, par mon tuteur de projet, d'un meilleur script permettant de parvenir au même résultat.

Sauvegarde incrémentielle

Dans la même lignée, la solution de sauvegarde incrémentielle *rsnapshot* a aussi été testé sur la VM de test *vmtest*, puis abandonnée en raison de sa non conformité au cahier des charge (car nécessitant trop de configuration).

3.3 Scripts ansible

En vue d'une utilisation en production, un playbook ansible permettant le déploiement automatique de l'agent Node_exporter ainsi que de Prometheus à été mis en place.

3.3.1 Installation des scripts de service pour node_exporter et Prometheus

Préalablement à la mise en place d'un playbook ansible, des tests visant à installer node_expoter et prometheus en tant que services ont été effectués.

Ceci permet :

— D'avoir un programme fonctionnant dès le démarrage du système (utilise en cas de redémarrage du système).
— D'avoir un retour à tout instant via des logs.

Pour Prometheus, l'installation s'est effectué de la manière suivante :

```
/url/local/bin/prometheus                binaire prometheus
/url/local/bin/promtool                  utilitaire prometheus
/etc/prometheus/prometheus.yml           fichier de configuration prometheus
/etc/prometheus/consoles                 fichiers pour appli web prometheus
/etc/prometheus/console_libraries        fichiers pour appli web prometheus
/var/log/prometheus/prometheus.log       fichier de log prometheus
/etc/init.d/prometheus                   script de service
```

Pour Node_exporter :

```
/url/local/bin/node_exporter             binaire node_exporter
/var/log/prometheus/node_exporter.log    fichier de log node_exporter
/etc/init.d/node_exporter                script de service
```

Les scripts de service, récupérés depuis internet, ont ensuite été installé grâce à la commande suivante :

```
update-rc.d prometheus defaults
update-rc.d node\_exporter defaults
```

La commande update-rc.d permet d'ajouter un script se trouvant dans le dossier /etc/init.d en tant que script d'initialisation System V.

3.3.2 playbook ansible

Parmi les outils permettant de faire de la configuration et de la gestion de noeuds à distance, Ansible fut l'outil dont l'utilisation m'a paru le plus adaptée. Ansible possède en effet l'avantage de ne pas recourir à l'utilisation d'agents et donc d'être simple à utiliser.

Ansible se configure via des fichiers YALM décrivant des tâches à accomplir, lesquels sont utilisés par un playbook.

Fichier "inventory" :

La configuration des hôtes cible se fait dans le fichier "inventory", comme suit :

```
[Serveurs]
beta ansible_host="172.26.64.23"
[VMs]
TestAnsible ansible_host="172.26.64.18"
cachalot ansible_host="172.26.64.22"
dune ansible_host="172.26.64.24"
fondation ansible_host="172.26.64.25"
hyperion ansible_host="172.26.64.30"
```

La connexion d'Ansible aux différents hôtes se fait en ssh, selon deux modalités :

— En précisant un mot de passe pour un groupe d'utilisateurs.
— En mettant la clé publique du serveur sur lequel se trouve Ansible dans le fichier */.ssh/authorized_keys* de la node cible.

Ici, le mot de passe usuel du compte root a été désigné pour tout les utilisateurs :

```
[all:vars]
ansible_connection=ssh
ansible_user=root
ansible_ssh_pass= <mot_de_passe_usuel>
```

Playbook

Les tâches à effectuer se décomposent en "rôles", lesquelles sont définis dans le playbook (fichier playbook.yml) :

```
---
# List the hosts you want your roles to be installed on
# Roles pour les marchines/VMs a superviser
- hosts: all
  roles:
    - node
# Roles pour le serveur de supervision
- hosts: supervise
  roles:
    - prometheus
```

Rôle "node" :

Le rôle "node" effectue des tâches suivantes :
— Installation de curl, afin d'avoir un aperçu direct des métriques exporté par l'agent node_expoter (commande curl localhost :9100/metrics)
— Installation des binaires et des scripts de services (voir 3.3.1)
— Lancement de l'agent node_exporter

Rôle "prometheus" :

Le rôle "Prometheus" comprend les tâches suivantes :
— Installation des binaires et des scripts de services (voir 3.3.1)
— Lancement du service Prometheus

Exemple tâche Ansible

Les tâches exécutés par Ansible sont rédigés en YALM, ce qui rend leur lecture très simple. Dans l'exemple ci-dessous, le service node_exporter est lancé :

```
- name: "start node\_exporter"
  service:
    name: node\_exporter
    state: started
    enabled: true
```

Tests effectués

Plusieurs tests ont été effectués sur différentes machines virtuelles, dont une créée pour l'occasion (TestAnsible) .

Le lancement du script ansible se fait la manière suivante :

```
ansible-playbook playbook.yml -i inventory
```

3.4 Système de sauvegarde

3.4.1 Cahier des charges

Le système de sauvegarde a été mis en place en tenant compte des contraintes suivantes :

— Un espace de stockage limité.
— Des backups regulières (dernier jour, semaine dernière, mois dernier).

L'idée pour le système de sauvegarde est d'effectuer des full-backups des disques d'une machine virtuelle. L'avantage de ce procédé est de rendre simple le processus restauration, puisqu'il suffit de remplacer l'image ou le volume virtuel par la backup.

3.4.2 Fonctionnement

Le travail effectué sur le système de sauvegarde a été effectué en reprenant les scripts fournis par Mr Redon :

— *VMSave* : effectue une copie des disques d'une machine virtuelle désigné. Le nom et l'adresse IP de l'hôte sont désignés en paramètre.
— *allSave* : lance le script VMSave en lisant une liste de VM dont le nom est donné en paramètre.
— *ListeVM* : Liste les machines virtuelles à sauvegarder et leur adresse IP.

Afin de répondre aux contraintes d'espace disque et de régularité des backups, le système de sauvegarde fonctionne grâce à un roulement et s'articule autour de trois tâches de sauvegarde simultanées :

— Une tâche quotidienne, faisant une full-backup de VMs chaque nuit à 2h du matin.
— Une tâche hebdomadaire, effectuant une full-backup de VMs chaque dimanche, à 2h du matin.
— Une tâche mensuelle, effectuant une full-backup de VMs chaque premier dimanche du mois, à 2h du matin.

A chaque fois qu'une tâche de sauvegarde est effectué, un roulement se produit et la backup la plus ancienne est supprimé. Au total, pour une machine virtelle, on compte 5 full-backups, correspondant aux 3 derniers jours, aux 2 dernières semaines et au dernier mois.

Afin de fonctionner sur plusieurs machines virtuelles distantes, le script de sauvegarde *allSave* prend en paramètre un fichier qui contient la liste des machines virtuelles à sauvegarder ainsi que leur adresse IP :

```
dune/192.168.0.1
fondation/192.168.0.1
hyperion/127.0.0.1
```

Enfin, à chaque exécution, le script *allSave* produit un fichier contenant le timestamp pris au moment de sauvegarde. Le fichier produit sera destiné à être exporté par Prometheus.

3.5 Installations

En plus du travail effectué en lien avec la supervision de la plateforme informatique, du temps a été consacré durant le projet à la réalisaion de diverses installations en salle serveur.

Les tâches accomplies sont les suivantes :
— Rackage et installation de quatre baie DAS sur le serveur de sauvegarde secondaire
— Rackage et installation d'une deuxième baie DAS sur le serveur de sauvegarde primaire.
— Rackage et installation d'un nouveau serveur de virtualisation.
— Installation de cartes réseaux fibre 10G sur baleine et sur brisban.

3.5.1 Installation du serveur de sauvegarde secondaire

La première étape a été l'installation du serveur de sauvegarde secondaire. Ce choix fut motivé par la nécessité de conserver les backups présentes sur baleine avant l'installation de la deuxième baie DAS.

3.5.2 Installation des baie DAS

Le choix pour chaque baie fut le suivant :

— Former un volume logique en RAID5 avec les 11 premiers disques.
— Le 12e disque en spare.

Puis, former un volume virtuel à partir d'un groupe de volumes contenant les quatre volumes logiques issus de chacune des baies.

Une autre possibilité aurait été de mettre en place un seul "Array" contenant tout les disques appartenant à toutes les baies. Cette possibilité aurait cependant entrainé un risque en cas de dysfonctionnement d'une des baies. En effet, si une baie ne fonctionne plus, cette dernière doit être impérativement remplacé afin de reformer le volume logique, tandis qu'avec la solution utilisé, il suffit de retirer un volume logique d'un groupe de volume.

Exemples de commandes :

```
Commande:
ctrl all show config
Résultat:
Smart Array P800 in Slot 4                (sn: P98690E9SUU1CD)
   array A (SAS, Unused Space: 0  MB)
      logicaldrive 1 (820.2 GB, RAID 5, OK)
      physicaldrive 3I:1:1 (port 3I:box 1:bay 1, SAS, 146 GB, OK)
      physicaldrive 3I:1:2 (port 3I:box 1:bay 2, SAS, 146 GB, OK)
      ...
```

```
      physicaldrive 4I:1:8 (port 4I:box 1:bay 8, SAS, 146 GB, OK, spare)
   array B (SATA, Unused Space: 0  MB)
      logicaldrive 2 (4.5 TB, RAID 5, OK)
      physicaldrive 2E:2:1 (port 2E:box 2:bay 1, SATA, 500 GB, OK)
      physicaldrive 2E:2:2 (port 2E:box 2:bay 2, SATA, 500 GB, OK)
      ...
      physicaldrive 2E:2:12 (port 2E:box 2:bay 12, SATA, 500 GB, OK, spare)
   array C (SATA, Unused Space: 0  MB)
      logicaldrive 3 (4.5 TB, RAID 5, OK)
      physicaldrive 2E:3:1 (port 2E:box 3:bay 1, SATA, 500 GB, OK)
      physicaldrive 2E:3:2 (port 2E:box 3:bay 2, SATA, 500 GB, OK)
      ...
      physicaldrive 2E:3:12 (port 2E:box 3:bay 12, SATA, 500 GB, OK, spare)
   array D (SATA, Unused Space: 0  MB)
      logicaldrive 4 (4.5 TB, RAID 5, OK)
      physicaldrive 1E:2:1 (port 1E:box 2:bay 1, SATA, 500 GB, OK)
      physicaldrive 1E:2:2 (port 1E:box 2:bay 2, SATA, 500 GB, OK)
      ...
      physicaldrive 1E:2:12 (port 1E:box 2:bay 12, SATA, 500 GB, OK, spare)
   array E (SATA, Unused Space: 0  MB)
      logicaldrive 5 (4.5 TB, RAID 5, OK)
      physicaldrive 2E:1:2 (port 2E:box 1:bay 2, SATA, 500 GB, OK)
      physicaldrive 2E:1:3 (port 2E:box 1:bay 3, SATA, 500 GB, OK)
      ...
```

NB : pas de disque de spare pour le volume logique 5, puisqu'un des disques de la 4e baie DAS ne fonctionne pas.

Création d'un volume logique en RAID5 :

```
ctrl slot=4 create type=ld drives=2E:2:1,2E:2:2,2E:2:3,...,2E:2:11 raid=5
```

Création d'un disque de spare sur l'array B :

```
ctrl slot=4 array B add spares=2E:2:12
```

Supprimer le volume logique n°2 :

```
ctrl slot=4 ld 2 delete
```

3.5.3 Installation des volumes virutels

Afin de faciliter la gestion de l'espace de stockage sur les serveurs de sauvegarde, un unique volume a été mis en place à partir des volumes logiques précédemment créé. Afin de parvenir à ce résultat, l'outil *lvm* a été utilisé.

Affichage des volumes logiques créés avec la commande fdisk -l :

```
Disk /dev/cciss/c0d1: 4.6 TiB, 5000742854656 bytes, 9767075888 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk /dev/cciss/c0d2: 4.6 TiB, 5000742854656 bytes, 9767075888 sectors
Units: sectors of 1 * 512 = 512 bytes
```

```
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk /dev/cciss/c0d3: 4.6 TiB, 5000742854656 bytes, 9767075888 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes</code>
Disk /dev/cciss/c0d4: 4.6 TiB, 5000742854656 bytes, 9767075888 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

Création d'un groupe de volume :

```
vgcreate mvg /dev/cciss/c0d1 /dev/cciss/c0d2 /dev/cciss/c0d3 /dev/cciss/c0d4
```

Résutat obtenu avec la commande vgdisplay :

```
 --- Volume group ---
 VG Name               mvg
 System ID
 Format                lvm2
 Metadata Areas        4
 Metadata Sequence No  4
 VG Access             read/write
 VG Status             resizable
 MAX LV                0
 Cur LV                1
 Open LV               1
 Max PV                0
 Cur PV                4
 Act PV                4
 VG Size               18.19 TiB
 PE Size               4.00 MiB
 Total PE              4769076
 Alloc PE / Size       788992 / 3.01 TiB
 Free  PE / Size       3980084 / 15.18 TiB
 VG UUID               MseO7m-SVYJ-P6I2-mske-g78n-24Ds-yewwfK
```

Création d'un volume virtuel de 3 To :

```
lvcreate -n Backup -L 3T mvg
```

Résutat obtenu avec la commande lvdisplay :

```
--- Logical volume ---
 LV Path                /dev/mvg/Backup
 LV Name                Backup
 VG Name                mvg
 LV UUID                4gNfSp-kChc-gQ8x-jv3m-8irm-Tc7S-MHPZ3u
 LV Write Access        read/write
 LV Creation host, time devuan1, 2019-01-28 17:01:09 +0100
```

```
LV Status              available
# open                 1
LV Size                3.01 TiB
Current LE             788992
Segments               1
Allocation             inherit
Read ahead sectors     auto
- currently set to     256
Block device           254:0
```

3.5.4 Installation des cartes réseau fibre 10G

Du temps fut aussi consacré à l'installation de cartes réseaux fibre 10G sur les serveurs baleine et brisban. Une liaison fibre entre le nouveau serveur de virtualisation et baleine a aussi été mise en place. L'idée derrière cette installation fut de na pas surcharger le réseau lors de la prise des backups par le serveur de sauvegarde baleine.

Au cours de cette installation, je me suis heurté à un problème matériel empêchant la détection de la liaison fibre entre les deux serveurs. Après avoir effectué une série de tests sur un grand nombre de cartes réseaux, je suis parvenu à détecter la source du problème. En cause : les cartes réseau fibre 10G de la marque Cisco. Afin de mettre en évidence ce problème, les cartes réseaux ont été testées une par une sur le commutateur.

3.5.5 Installation d'un nouveau serveur de virtualisation

Un nouveau serveur de virtualisation a aussi été mis en place en salle serveur. L'objectif de cette installation fut de tester le système de sauvegarde sur des machines virtuelles distantes. Deux nouvelle machine virtuelles furent créées à cette occasion :

— Dune
— Fondation

La configuration suivante a été mise en place :

— Réseau local entre baleine et beta (le nouveau serveur de virtualisation) en 192.168.0.*.
— Bridge relié au réseau insecure.

Configuration réseau :

```
# Liaison fibre
auto eth0
iface eth0 inet static
        address 192.168.0.1
        netmask 255.255.255.0
# Liaisons ethernet
auto eth2
iface eth2 inet manual
        up ip link set $IFACE up
auto eth3
iface eth3 inet manual
        up ip link set $IFACE up
auto bridge
iface bridge inet static
```

```
bridge_ports eth2 eth3
address 172.26.64.23
netmask 255.255.240.0
gateway 172.26.79.254
```

3.6 Serveur gitlab

Les problèmes rencontrés récemment sur la base de donnée de l'actuel serveur gitlab archives.plil.fr ont conduit à la volonté de mettre en place un nouveau serveur gitlab sur le serveur archives.ilot.org.

Les objectifs qui m'ont été donné sont les suivants :

— Dupliquer l'actuel serveur gitlab (conserver la même configuration).
— Récupérer (si possible) les projets présent sur l'actuel gitlab.

3.6.1 Installation et configuration

L'installation du nouveau serveur gitlab s'est faite de manière très simple en installant le package omnibus. Afin de rester fidèle a l'ancien gitlab, la version communautaire a été installé.

La configuration LDAP présente sur archives.plil.fr a ensuite été calqué sur le nouveau gitlab afin de permettre aux étudiants et aux enseignants de se connecter.

3.6.2 Restauration des projets du serveur archives.plil.fr

Le principal problème posé par la mise en place du nouveau gitlab est la récupération des projets présent sous la forme de dossier git. Ces dernier sont accessibles "en dur" dans le dossier suivant :

```
/var/opt/gitlab/git-data/repositories/
```

Dans un premier temps, une restauration en "dur" des projets fut tenté en utilisant la commande suivante :

```
gitlab-rake gitlab:import:repos['/var/opt/gitlab/git-data/repositories/']
```

Cependant, recourir à cette méthode n'a pas permis pas d'associer les projets aux utilisateurs, ces derniers n'existant pas dans la base de donnée du nouveau serveur gitlab.

Des observations faites sur la base de donnée ont révélées que lors de l'importation d'un projet "en dur", le serveur gitlab créé un "namespace" correspondant à l'identité du détenteur du projet. Il peut s'agir d'un utilisateur, d'un groupe ou d'un sous groupe.

Lors d'une première connexion sur archives.ilot.org, il se passe les choses suivante :

— Un utilisateur correspondant à l'identifiant polytech est créé (par exemple, tdjeraba).
— Le serveur gitlab détecte la présence du namespace "tdjeraba", qui à été ajouté lors de l'importation des projets git présent dans le dossier du même nom.
— Le namespace "tdjeraba1" est ensuite créé.

Si on inverse les étapes précédentes, c'est à dire que l'on créé l'utilisateur "tdjeraba" lors d'une première connexion et que l'on importe le projet par la suite, on parvient par contre à associer les projets à l'utilisateur "tdjeraba".

Avant d'importer les projets, il faut donc faire en sorte de peupler la nouvelle base de donnée avec les utilisateurs et avec leur namespace.

3.6.3 Restauration de la base de donnée psql du serveur archives.plil.fr

L'étape suivante a donc consisté à tenter de restaurer l'ancienne base de donnée sur le nouveau serveur gitlab afin d'associer les projets à leurs utilisateur.

Sauvegarde et restauration

Gitlab est fourni avec gitlab-rake, un outil permettant d'effectuer des tâches de maintenance via des rake-tasks.

Pour faire une backup, on utilise la commande suivante :

```
gitlab-rake gitlab:backup:create
```

Cette commande effecte une backup que l'on peut retrouver dans le dossier suivant : /var/opt/gitlab/git-data/backups/, sous la forme d'une archive tar, et dont le nom se présente sous la forme suivante : timestamp_gitlab_backup.tar

La commande suivante permet de restaurer une backup :

```
gitlab-rake gitlab:backup:restore BACKUP=<Timestamp>
```

Problèmes et résolution

Archives git corrompues Dans un premier temps, l'ensemble des projet corrompus ont du être purgé afin de pouvoir effectuer une backup sans bug. Pour cela, un petit script permettant de vérifier l'intégrité de chaque projet git a été rédigé. Ce dernier rentre dans chaque dossier git et exécute la commande suivante : *git fsck*. 57 projets git ont finalement été mis de côté dans le dossier /root/repo-bug du serveur limonade.

Échecs de Restauration Plusieurs tentatives de restauration ont été effectués sans grand succès, et ce pour plusieurs raisons :

— Possibles bugs dans l'ancienne base de donnée.
— Différence de version entre le nouveau et l'ancien serveur gitlab.
— Différence de version entre la nouvelle et l'ancienne base de donnée psql.

Afin de résoudre ces problèmes, j'ai tenté de downgrader le nouveau serveur gitlab afin de restaurer l'ancienne base de donnée, mais ce fut sans succès. En dépit d'une restauration effectué sans erreur, le nouveau serveur gitlab n'a pas fonctionné.

3.6.4 Peuplement de la nouvelle base de donnée sur le nouveau serveur gitlab

A cause des problèmes évoqués précédemment, la nouvelle base de donnée a dû être peuplé manuellement avec les données récupérées sur l'ancienne.

Les données issues de l'ancienne base de données furent récupérées à partir de la backups généré précédemment, sur un fichier sql contenant l'ensemble des requêtes permettant le peuplement de la nouvelle base de donnée.

Exemple de requête permettant de peupler la table "namespaces" :

```
--
-- Data for Name: namespaces; Type: TABLE DATA; Schema: public; Owner: gitlab
--
COPY namespaces (id, name, path, owner_id, created_at, updated_at,
type, description, avatar) FROM stdin;
1   root    root    1       2015-03-08 08:55:01.965703      2015-03-08 08:55:01.965703
\N              \N
2   rex     rex     2       2015-03-10 15:16:39.707054      2015-03-10 15:16:39.707054
\N              \N
3   tmaurice        tmaurice        3       2015-03-10 18:20:27.326589
2015-03-10 18:20:27.326589
\N              \N
4   tvantroy        tvantroy        4       2015-03-10 19:49:38.046898
2015-03-10 19:49:38.046898
\N              \N
...
```

Afin d'accéder à la base de donnée, on utilise la commande suivante :

```
gitlab-rails dbconsole
```

De là, on peut accéder aux informations de la bdd et y ajouter de nouveaux éléments.

Préalablement à l'ajout des utilisateurs dans la nouvelle bdd, des modifications ont dû être faite au niveau des données à insérer. En effet, puisque les versions des bdds sont différentes, certaines entrées présente dans l'ancienne bdd ne sont plus présente dans la nouvelle.

Les données à insérer ont donc été récupérées, mises dans des fichier csv, purgées des données inutiles (car non existante dans la nouvelle bdd), puis reformaté et enfin inséré dans la nouvelle base de donnée.

Les tables suivantes ont été peuplés :

— *identities*, contenant des informations relatives à l'identité d'un utilisateur, récupérées depuis le serveur LDAP.
— *users*, contenant des informations relatives aux utilisateurs (nom, adresse mail, avatar, bio, etc ...)
— *namespaces*, reliant les utilisateurs à leurs namespace.

Problèmes et résolutions

L'ajout des utilisateurs dans la bdd s'est effectué avec succès. Un autre problème est cependant survenu, avec le renvoi de l'erreur "Access denied for your LDAP account" lors d'une tentative de connexion.

En consultant le fichier log suivant : /var/log/gitlab/gitlab-rails/application.log, on constate que l'erreur suivante : "Namespace route can't be blank".

Ici, le test a été effectué sur moi et sur eloi zalczer :

```
February 14, 2019 17:58: (LDAP) Error saving user cn=ezalczer,ou=people,dc=polytech-lille.fr
(Eloi.Zalczer@polytech-lille.net): ["Namespace route can't be blank"]
February 14, 2019 17:59: (LDAP) Error saving user cn=tdjeraba,ou=people,dc=polytech-lille.fr
(Taky.Djeraba@polytech-lille.net): ["Namespace route can't be blank"]
```

Après avoir fouillé dans la nouvelle base de donnée de gitlab, j'ai découvert une table "route", permettant d'associer l'id (indiqué ci-dessous dans source_id) d'un namespace à un répertoire dans le dossier /var/opt/gitlab/git-data/repositories/ :

```
gitlabhq_production=> SELECT * FROM routes WHERE source_type='Namespace';
  id  | source_id | source_type |        path        | ...
------+-----------+-------------+--------------------+ ...
 1001 |         1 | Namespace   | root               | ...
 1002 |         2 | Namespace   | rex                | ...
 1003 |         3 | Namespace   | tmaurice           | ...
 1004 |         4 | Namespace   | tvantroy           | ...
 1005 |         6 | Namespace   | bmaliar            | ...
 1006 |         7 | Namespace   | bcoenen            | ...
 1007 |         8 | Namespace   | cvillain           | ...
 1008 |         9 | Namespace   | jturut             | ...
 1009 |        10 | Namespace   | dfourny            | ...
 ...
```

Étant que cette table n'existe pas dans l'ancienne base de donnée, j'ai "simulé" la création de mon compte en commençant par supprimer toutes les références à mon compte dans la bdd puis en me connectant à nouveau au gitlab. J'ai ensuite étudié la base de donnée afin de voir comment les tables sont remplis lors de la création d'un compte.

On voit dans l'exemple ci-dessous que le namespace de nom "tdjeraba" et d'id "40" a été créé :

```
gitlabhq_production=> SELECT * FROM namespaces WHERE name='tdjeraba';
id |   name   |   path   | owner_id | ...
---+----------+----------+----------+ ...
40 | tdjeraba | tdjeraba |      226 | ...
(1 row)
```

Dans la table ci-dessous, on voit que le path ayant pour source_id l'id du namespace "tdjeraba" a été ajouté :

```
gitlabhq_production=> SELECT * FROM routes;
id | source_id | source_type |                path                | ...
---+-----------+-------------+------------------------------------+ ...
83 |        40 | Namespace   | tdjeraba                           | ...
84 |        49 | Project     | tdjeraba/ima3_tutorat_pa_2017      | ...
85 |        50 | Project     | tdjeraba/Djeraba_Cartier_Mouvement | ...
86 |        51 | Project     | tdjeraba/PFE_IMA_5                 | ...
87 |        52 | Project     | tdjeraba/Un_test                   | ...
(5 rows)
```

Grâces aux données précédentes, j'ai finalement pû peupler la table "route" à partir de la table "namespace", et ce, pour tout les utilisateurs. Cela m'a finalement permis d'associer efficacement les projets à leurs utilisateurs.

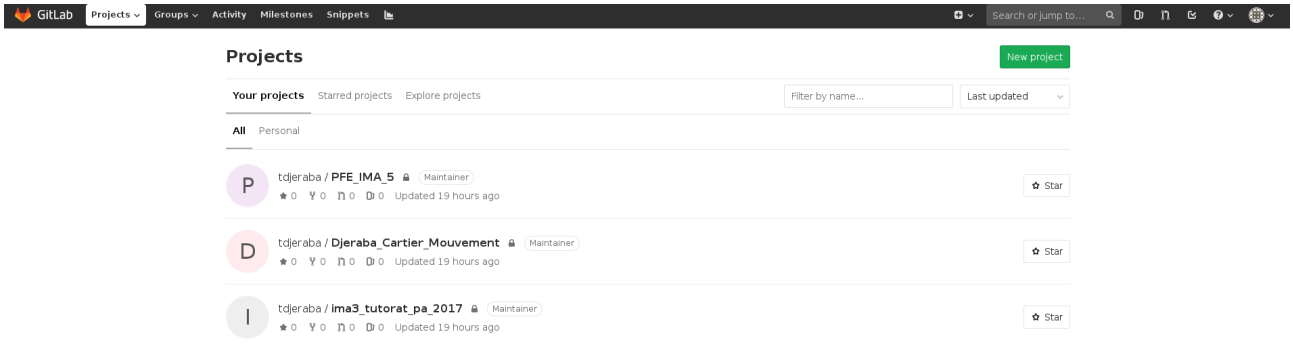

FIGURE 3.7 – Apperçu de la nouvelle interface gitlab pour l'utilisateur "tdjeraba"

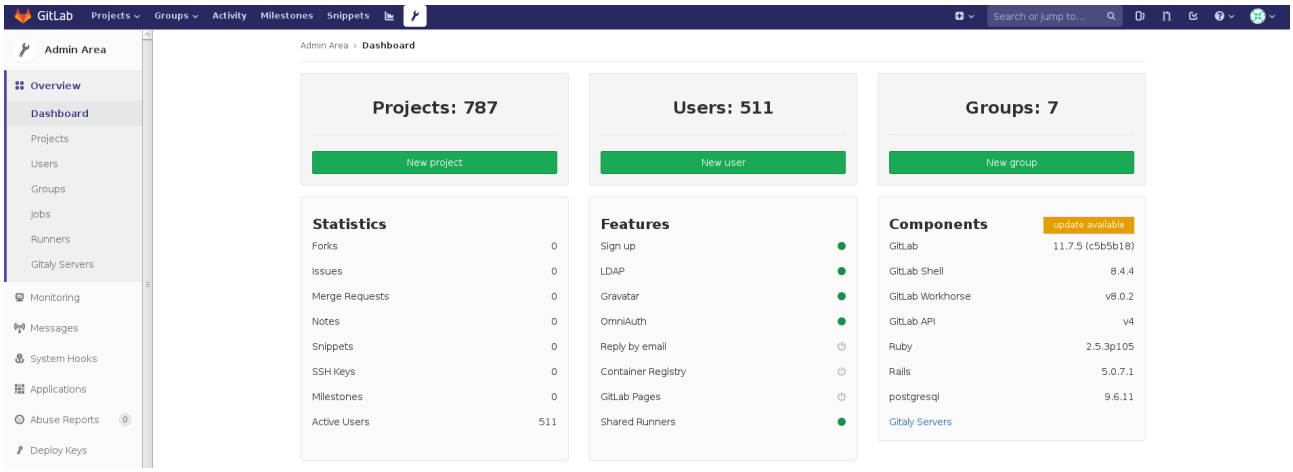

FIGURE 3.8 – Tableau de bord du compte root

3.6.5 Pertes

A l'issue de la duplication du serveur gitlab, certaines données ont réussis à être conservées, d'autres cependant, furent perdues.

Sont conservés :

— Les utilisateurs et les groupes.
— Les fichiers des projets liés à leur créateur.
— Les commits.
— Les statistiques.

Pertes :

— Les participants des projets.
— Les issues.
— Les merge requests.

Chapitre 4

Mode d'emploi

Accès a l'interface Grafana

L'accès à l'interface Grafana se fait en accédant à l'addresse suivante :

```
Adresse 172.26.64.13:3000,
login: admin
mot de passe root usuel
```

On accède alors à la page principale, affichant les informations suivantes :
— Uptime machines.
— États des disques du serveur de sauvegarde baleine.
— États réseau ADSL, SDSL et Renater.
— Dates de validités des clés DNSSEC.

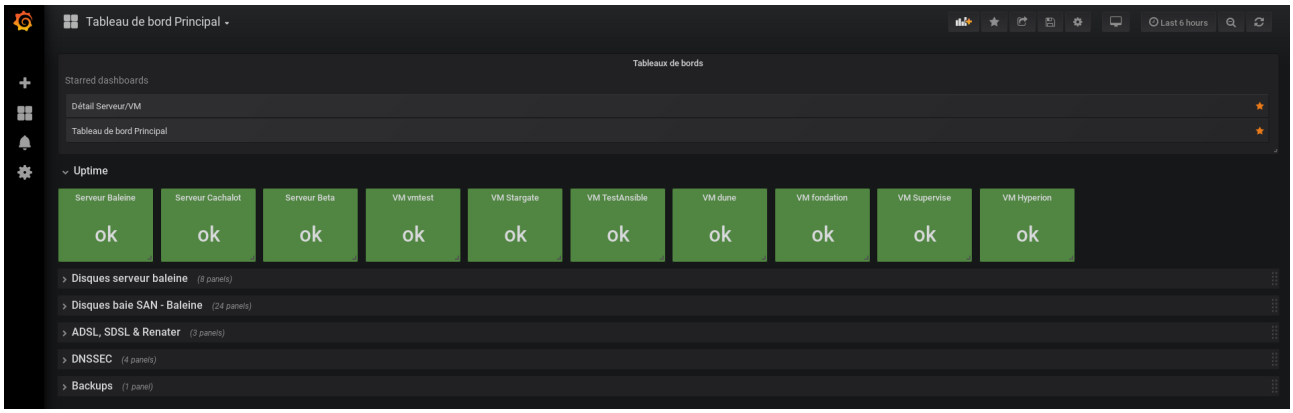

FIGURE 4.1 – Page d'accueil

Pour avoir le detail des VMs/Serveurs, cliquer sur "Détail Serveur/VM" :

Lancement playbook Ansible

Pour lancer le playbook ansible, se rendre dans le répertoire Ansible sur baleine (/etc/ansible/) et lancer la commande suivante :

```
ansible-playbook playbook.yml -i inventory
```

NB : Python2.7 doit être installé sur les machines cibles

FIGURE 4.2 – Détail des serveurs/VMs

Importation des projets git

Afin d'importer les projets git à partir de l'ancien archives.plil.fr sur le nouveau serveur gitlab, on commence par transférer les repositories d'un serveur à l'autre.

```
scp -r * /var/opt/gitlab/git-data/repositories/* \
root@193.48.57.234:/var/opt/gitlab/git-data/repositories/
```

NB : attention à ne pas tenter de compresser les archives git dans un tar au risque de faire planter momentanément le serveur archives.plil.fr

Sur le nouveau serveur gitlab, on change l'utilisateur des archives puis on importe ensuite les repos grâce à la commande suivante :

```
chown -R git.git /var/opt/gitlab/git-data/repositories/*
gitlab-rake gitlab:import:repos['/var/opt/gitlab/git-data/repositories/']
```

Projet Git

le projet git est disponible via le lien suivant :

```
https://archives.plil.fr/tdjeraba/PFE_IMA_5
```

Conclusion

En conclusion, le projet m'aura permis d'explorer les différentes facettes du métier d'administrateur système, que ce soit à travers la rédaction de scripts pour le monitoring de serveurs ou le rackage et l'installation de serveurs.

Ce projet m'a aussi permis de me familiariser avec certaines technologies très utilisées dans le monde du DevOps, comme Prometheus, Nagios, Grafana ou encore Gitlab.

Enfin, ce projet m'aura permis de prendre toute la mesure du niveau de responsabilité que l'on possède lorsque l'on travaille sur une infrastructure utilisée au quotidien par un grand nombre de personnes.