Automatisation dans un S.I et mise en place d'une Solution de Monitoring

Université de Bordeaux
LPRO ADSILLH 2020

Marc Cenon

marc.cenon33@gmail.com

marc-cenon.github.io/my_resume/

18 août 2021

Table des matières

Remerciements

Tout d’abord, je voudrais remercier mon maître de stage, **Mr Thomas Coleno**. Il a su me faire confiance et a partagé ses connaissances de manière très pédagogique. Je le remercie aussi pour sa disponibilité et la qualité de son encadrement en entreprise.

Je tiens à remercier également **Mr Arthur Bertinetti** et **Mr Laurent Potou** pour leur patience et leur grande pédagogie. Ils ont su m’aider sur pleins de problématiques.

J’ai pu ainsi bénéficier de leur grande expérience, ce qui m’a permis d’avoir une bonne montée en compétence.

En effet, chacune des personnes de l’équipe à su me consacre du temps et partager avec moi leur expertise, méthodes et connaissances tout au long de ce stage. Ils m’ont permis de rendre cette expérience de 6 mois enrichissante et pleine d’intérêt.

J’ai énormément appris. Ils m’ont fait confiance pour travailler avec eux sur plein de projets et avec une grande autonomie et je les en remercie vivement.

Je les remercie également pour la bonne humeur qu’ils ont su me communiquer et l’envie qu’ils m’ont donné de travailler au sein de leur équipe.

Je tiens à remercier également le corps enseignant de l’Université, notamment **Mr Samuel Thibault** et **Mr Olivier Delmas** pour leurs soutiens et leurs enseignements. Ils m’ont permis de mener à bien ma reconversion professionnelle grâces à leurs conseils, à leurs excellents cours.

Introduction

Dans le cadre de la Licence professionnelle Administration et Développement de Systèmes Informatiques à base de Logiciels Libres et Hybrides **ADSILLH**, j'ai effectué un stage de 6 mois au sein de l'équipe **ENT** (Espace Numérique de Travail) / **Local GOV** (Gouvernement Local) dans la Bussiness Unit **TPSHR** (Transport, Secteur Public, Ressources Humaines)dans l'entreprise CGI (Conseillers en Gestion Informatique ou Consultants to Government and Industry en anglais) au Haillan.

Je vais vous présenter dans ce rapport l'entreprise qui m'a accueilli et plus précisément l'équipe où j'ai réalisé mon stage. Vous trouverez en annexes un tableau qui reprend les tâches sur lesquelles j'ai travaillé, semaine après semaine.
Lien ici

J'ai eu l'occasion de pouvoir travailler sur pleins de projets différents mais avec comme fil conducteur l'automatisation et c'est pourquoi j'ai choisi comme thème de rapport de stage **l'automatisation dans un S.I** avec un focus sur le déploiement d'une solution de **monitoring** avec **Ansible** car c'est l'un des projet qui m'a beaucoup plus. J'utilisai cette solution de monitoring chez moi pour mon infrastructure. Le fait de devoir travailler dessus m'a beaucoup plus et m'a vraiment fait progresser sur le monitoring et m'a permis d'améliorer mon installation.

Je présenterai dans un premier temps Ansible, son fonctionnement et quelques commandes utiles. Le but est de comprendre Ansible avant de présenter la solution de monitoring.

Je présenterai en suivant les différents élément de la solution de monitoring, et l'infrastructure sur laquelle elle sera déployée.

J'illustrerai avec des morceaux de codes les différents point important du projet

Et avant la conclusion, je parlerai de la configuration supplémentaires possible des des différentes évolution set amélioration qui pourront êtres mis en place.

Aucunes données confidentielles ne seront présentées dans ce rapport.

Le but de ce stage était d'intégrer l'équipe Infrastructure afin de participer au développement du numérique à l'école ainsi que sur la gestion de cette infrastructure. Ce stage m'a permis d'apprendre et de manipuler des technologies comme :

- Automatisation : Ansible, Openstack
- Virtualisation : vSphere, VMware ESXi
- Conteneurisation : Docker, Kubernetes
- Base de Données : Mariadb, Postgresql, Influxdb
- Applications diverses : Moodle, Big Blue Button, Jupyter
- Messagerie : Zimbra
- Outils de ticketing : Jira
- Documentation : Confluence
- Monitoring : Centreon, Grafana, Loki, Telegraf, Promtail
- Scripting : Bash, Python

Au-delà du gain en compétences techniques, l'immersion au sein d'un processus de gestion de projet m'a appris à reconnaître et interagir avec chacune des phases du projet sur le terrain.

Cette immersion au sein d'un environnement complexe m'a également appris à être plus efficace, que ce soit par le biais d'une meilleure gestion de mon temps ou encore une meilleure communication sur l'avancement de mes tâches auprès de l'équipe que j'ai intégré.

Ce rapport est disponible sur mon Github personnel en Markdown :

```
https://github.com/marc-cenon/rapport_de_stage/blob/master/rapport.md
```

Vous y trouverez le Playbook de monitoring que je vais présenter dans mon rapport. Ce PDF a été généré à partir du rapport en Markdown grâce à **Pandoc** et au fichier **text.tex** qui comprend les différentes variables utilisées pour le bon formatage de ce dernier. Vous pouvez compiler le rapport avec la commande suivante, à condition d'avoir installé Pandoc.

```
git clone https://github.com/marc-cenon/rapport_de_stage.git

cd rapport_de_stage

pandoc --listings -H text.tex rapport.md -o files/rapport.pdf --pdf-engine=xelatex
```

Présentation de l’entreprise et du cadre du stage

Présentation de CGI

Fondé en juin 1976 par Serge Godin à Québec, Canada, CGI est un groupe canadien actif dans le domaine des technologies de l’information et en gestion des processus d’affaires. Au cours des dix premières années d’existence, CGI a développé une stratégie, un modèle et un ensemble de principes de gestion qui se sont traduits par une croissance considérable. Devant les demandes des clients d’externaliser leurs systèmes informatiques, CGI s’est adapté et à élaborer une nouvelle stratégie pour se positionner sur le marché émergent de l’externalisation.

Durant la fin des années 80 et début 90, CGI commença à acquérir des sociétés proposant des services d’externalisation. Dès lors, CGI est en mesure d’offrir à ses clients des services informatiques complets tels que des services en TI (Technologies de l’Information) et en gestion, des services d’intégrations de système et d’externalisation.

Dans les 20 dernières années, CGI chercha à atteindre une taille critique sur les marchés géographiques de ses clients, d’acquérir une croissance approfondie de leurs secteurs d’activités ainsi que de développer des pratiques spécialisées et des solutions novatrices. En 2010, CGI fait l’acquisition de Stanley Inc. et de ses filiales Oberon et Techrizon dans le but de doubler la taille de ses activités aux États-Unis. Deux années plus tard, CGI réalisa sa plus grosse acquisition en fusionnant avec l’entreprise Logica faisant passer son nombre de collaborateurs de 31 000 à 68000.

Au cours de son histoire, CGI a réussi une expansion imprésionnante et continue pendant 35 ans grâce à une stratégie de rachat et de conquête des différents marchés comme en témoigne le tableau

CGI est l’un des leaders mondiaux du conseil et des services numériques. Avec plus de 40 ans d’expertise et de savoir-faire et présent dans plus de 40 pays, le groupe CGI est implanté dans 21 villes en France avec environs 11 000 salariés.

L’entreprise est actuellement dirigée par trois personnes :

- Serge Godin : Fondateur et président exécutif du conseil
- André Imbeau : Fondateur et membre du conseil d’administration
- George D. Schindler : Président et chef de la direction

Avec une présence dans 40 pays, une solide expertise dans tous ses marchés cibles et un éventail complet de service en IT, la priorité de CGI reste de satisfaire ses clients. Grace à une approche cohérente, disciplinée et responsable en matière de prestation de services, CGI affiche un bilan inégalé de 95% de projets réalisés dans le respect des échéances prévues et affiche un indice de satisfaction des clients qui est constamment supérieur à 9 sur 10. Ce score de satisfaction couplé à la croissance continue de CGI témoigne de la confiance que ses clients accordent à CGI et du dévouement de ses collaborateurs.

Ceci dans le but de devenir un fournisseur de services complets, d’atteindre des résultats grâce à des ressources mondiales, à une connaissance approfondie de l’industrie, à une stabilité et des professionnels motivés. CGI possède maintenant

6 domaines d'expertises métiers qui sont le Business Consulting, l'intégration de systèmes, l'Outsourcing IT, les Services d'infrastructures, l'Application management et les Business procès services. Ces 6 domaines d'expertises sont répartis dans pas moins de 9 secteurs d'activités.

CGI est la cinquième plus importante entreprise indépendante en services IT et en gestion des processus d'affaires au monde au service avec plus de 10 000 clients dans le monde dont 500 en France.

Le groupe est composé de 70 000 membres répartis sur 400 bureaux répartis dans 40 pays dont 22 en France et réalise 7,6 milliards € de revenus mondiaux dont 1 milliard en France, au travers de projets intégration de système, d'outsourcing IT et également plus de 100 solutions exclusives soutenant les activités critiques de nos clients.

L'implantation de CGI en France résulte de la fusion de CGI avec Logica en 2012. Au niveau national, la filiale française de CGI est dirigée par Jean-Michel Baticle, entré dans le groupe en 1969. Son implantation dans la plupart des grandes villes françaises lui procure une implantation homogène pour couvrir l'ensemble du territoire métropolitain.

La structure de direction de CGI France est centrée autour des clients et chacune de ses activités sont regroupées au sein de Business Units qui sont au cœur même du modèle de CGI.

Le contexte de travail

En France, CGI est organisé en différentes Business Units **B.U**. J'ai réalisé mon stage dans la B.U TPSHR, plus précisément dans le groupe **Local GOV**, au service des collectivités locales.

Local Gov a pour but de proposer aux collectivités territoriales des solutions de services visant à faciliter le quotidien du citoyen, rendre les accès plus directs aux services et permettre un plus grand bénéfice de la dématérialisation.

Mon maitre de stage **Mr Thomas Coleno** ainsi que **Mr Laurent Potou** et **Mr Arthur Bertinetti** m'ont accueilli dans leur équipe. Le contexte sanitaire actuel a fait que 99% de mon temps de travail été à distance. Grâce aux outils collaboratifs comme Teams et Slack ainsi que la visioconférence ont permis de pouvoir communiquer dans de bonnes conditions.

Ce contexte m'a forcé à travailler sur mon autonomie. Cela a été pour moi très important car cela m'a poussé à chercher par moi-même et à solliciter mes collègues seulement en cas de difficultés. Dans un sens, cela a été très formateur. A partir du mois de Juillet, nous avons pu nous réunir une fois par semaine dans les locaux de CGI au Haillan.

Le fait de pouvoir télétravailler pour moi a été une réelle découverte comparée à mes postes précédant où, en tant que courtier en vin j'étais en déplacement constant et ne pouvais pas travailler depuis mon domicile.

Le télétravail m'a permis de trouver un certain confort pour équilibrer le contexte professionnel et personnel.

Mes missions

J'ai été recruté pour rejoindre l'équipe qui travaille dans le secteur de l'éducation nationale et particulièrement sur **l'ENT** : Espace Numérique de Travail, qui est utilisé par plusieurs régions de France. Cet ENT, très complet fournis des solutions clés en mains au collégiens et lycéens mais également aux professeurs et parents d'élève. Dans le contexte sanitaire actuel, l'équipe a dû s'adapter très rapidement pour fournir une solution performante et robuste afin de pouvoir supporter le fort développent du télé-enseignement. Je présenterai rapidement les principaux outils de cet ENT afin de comprendre les différentes applications sur lesquelles j'ai pu travailler.

Je suis donc arrivé en Avril 2021 afin de pouvoir accompagner l'équipe en place dans leur travail au quotidien. Je peux définir mon travail durant le stage en 3 axes :

- **Prévention** :
  Tous les jours je rédigé un rapport sur les alertes de la veille. Ce rapport utilise la solution de monitoring CENTREON, avec des sondes et des paramètres spécifiques à la surveillance de l'infrastructure. Je relevé également les anomalies sur les différentes machines remontées par l'antivirus CLAMAV Le but étant de surveiller les différentes infrastructures en place afin d'être proactif dans la résolution d'incidents.
- **Action** :
  Une bonne partie de mon travail a consisté à automatiser des tâches qui aurait été très chronophages. Mon tuteur **Mr Thomas Coleno** a une excellente maîtrise de cet outil et il m'a permis d'apprendre en réalisant plusieurs scripts Ansible, particulièrement le déploiement d'une stack de monitoring que je présenterai dans la partie 2 de ce rapport. J'ai également aidé l'équipe sur toutes les tâches qu'ils ont pu me confier. J'ai eu la chance d'avoir un stage avec des missions très variés. Ce qui a été très formateur sur beaucoup de technologies différentes et avec des problématiques différentes.
- **Montée en Compétences** :
  La diversité des briques logicielles a fait que j'ai grandement appris et je suis monté en compétences sur beaucoup de domaines comme Ansible, la gestion de BBD ou la mise en place de serveurs web. Cela m'a amené à faire beaucoup de troubleshooting sur différentes technologies afin de savoir comment elles fonctionnent..

Quelqu'un des projets sur lesquels j'ai pu participer :

- Création d'un Playbook Ansible pour le déploiement de la configuration de BBB
- Création d'un Playbook Ansible pour le Monitoring
- Création d'un Playbook Ansible Apache
- Création d'un Playbook Ansible pour le déploiement d'un établissement de formation avec différentes briques logicielles
- Utilisation de VSphere et NSXEDGE pour créer des Vlans, Firewalls, Loadbalancing, TLS, …
- Installation de divers serveurs d'applications : Moodle, Peertube, Drupal, BigBlueButton, …
- Installation de différentes Bases de données : Maria, PostgreSQL, InfuxDB
- Mise a jours de messagerie Zimbra pour diverses Régions
- Création d'un Playbook pour l'automatisation de la création et le paramétrage de VM dans vSphere
- Installation d'un WAF (Web Application Firewall)

Présentation de l'ENT

Un espace numérique de travail (ENT) est un ensemble de services numériques choisis et mis à disposition d'un ou plusieurs établissements scolaires dans une ou plusieurs régions de France. En annexe, vous trouverez un exemple d'ENT pour la région Nouvelle Aquitaine avec les différentes applications qui sont proposées.

lien ici

Les services proposés

En fonction des régions et des besoins, plusieurs services sont disponibles dans l'ENT. Voici une liste des plus important :

- **Messagerie Zimbra** :
  La messagerie collaborative Zimbra propose une couverture fonctionnelle étendue. En plus des fonctionnalités classiques de messagerie, Zimbra propose des outils intégrés comme le carnet d'adresses, l'agenda, ou encore le gestionnaire de tâches.
- **Moodle** :
  Moodle est une plateforme d'apprentissage en ligne. Elle permet aux enseignants de mettre en ligne des cours / quizz pour les étudiants. Sa force réside dans la grande variété de plugins qui permettent de répondre à des besoins spécifiques pour la création.
- **Big Blue Button** :
  Big Blue Button est une solution de visioconférence idéale pour la formation à distance. (En temps de covid cette solution a été extrêmement sollicité).
- **Peertube** :
  Peertube est une solution d'hébergement de vidéos décentralisées permettant la diffusion en peer to peer et c'est également un média social sur lequel les utilisateurs peuvent interagir et partager des vidéos en streaming.
- **Jupyter** :
  Jupyter est une application web permettant aux étudiants de coder en différents langages.
- **Wekan** :
  Wekan est un logiciel en ligne pour gérer des projets et partager des tâches grâce à la méthode Kanban.
- **Riot** :
  Riot est une messagerie instantanée chiffrée, multi-plateforme et pouvant être décentralisée, avec une interface très intuitive et une bonne gestion des salons et communautés.
- **PMB** :
  PMB est un service qui organise tout type de documents (livres, documents audiovisuels, des périodiques et d'une manière générale tout type de documents numériques à vocation documentaire) en une seule base de données. Cela permet pour les professeurs d'avoir une interface de catalogage unique et pour les étudiant une seule interface de recherche.
- **Libre Office Online** :
  LOOL est une suite bureautique très complète en ligne.

L'ensemble des solutions utilisées par les ENT sont Open Source (avec des versions payantes disponibles pour certaines des applications). Les scripts Ansible nous

permettent de déployer rapidement ces services à la demande en fonction du besoin des régions car chaque région utilise une base commune et des services spécifiques.

L'automatisation

Pourquoi automatiser ?

L'automatisation consiste à utiliser des logiciels pour créer des instructions reproductibles dans le but de remplacer ou de réduire l'intervention humaine. C'est un gain de temps et surtout cela permet de garantir le même résultat pour une opération réalisé **n fois** avec les mêmes paramètres : c'est le principe **d'idempotence**.

On passe du temps à écrire des règles d'automatisations mais une fois ces dernières testées et approuvées sur des environnement de développement ou de test, on peut s'assurer du résultat et enlever les erreurs humaines (ex : faute de frappe, …)

L'automatisation est un élément clé de l'optimisation de l'environnement informatique dans un monde qui évolue rapidement, c'est donc un rôle essentiel. Je peux prendre par exemple un administrateur système qui doit déployer 30 machines avec de la configuration pointue dans plusieurs fichiers avec plusieurs briques logicielles impliquées. Installer 30 machines les unes après les autres est chronophages et on peut perdre son attention et faire des erreurs.

En automatisant le processus, on aura le même résultat sur toutes les machines.

Cela également peut permettre à une personne d'utiliser un script de configuration simplement sans pour autant avoir les compétences techniques pour configurer un service complexe.

Ansible est un outil libre qui sert à automatiser la gestion de la configuration, du déploiement et de l'orchestration. Ses points forts :

- Pas d'agents à déployer sur les machines
- Permet de déployer des configurations normalisées : la même configuration sur un grand nombre de machine
- Permet de déployer des configurations plus spécifiques : on peut cibler une machine ou un groupe de machines
- Utilisation de SSH pour communiquer les tâches d'exécutions sur les machines cibles (pas besoins d'ouvrir de ports spécifiques)
- Utilisation de YAML comme langage – Grande communauté Lancé en 2013 et acquis par Red Hat en 2015. Avec plus d'un quart de millions de téléchargements, il est actuellement l'outil d'automatisation de logiciel libre le plus populaire sur GitHub.
- Ansible Galaxy : collection de rôles pour un grand nombre de tâches. Plus besoin de faire de Script Bash Pour des tâches comme installer un serveur NGINX, des rôles sont disponibles où seul un paramétrage des variables du Playbook permet d'obtenir un résultat reproductible, prévisible et fiable.

Ansible permet d'automatiser la configuration à plusieurs différents niveaux (systèmes d'exploitation, composantes d'application), et peut être appliqué à différents équipements (serveur, stockage, réseau) ou infrastructures (Bare-metal, VM, Cloud).

Ansible s'inscrit dans la mouvance **IaC : Infrastructure as Code**, c'est à dire gérer la configuration d'une Infrastructure à l'aide de fichiers de configuration stockable, et versionable.

Avec le développement des Infrastructure Cloud, Ansible, couplé à des outils comme Terraform et Packer, permet de gérer un infrastructure Cloud en mode IaC.

Personnellement, je ne vois que des avantages dans ce mode de gestion IaC. C'est ce que j'utilise pour gérer mon homelab (Cluster sous Kubernetes de 8 raspberry pi).

Le fait de pouvoir redéployer son infrastructure et sa configuration grâce à des fichiers de configurations versionables, est un atout majeur en cas de problème technique. Une réinstallation d'un service peut être réalisé rapidement.

Différence entre Ansible et Script Bash

Les scripts Bash sont fréquemment utilisés pour configurer voir automatiser certaines actions. Écrire des Script en Bash nécessite une bonne connaissance de ce langage de Scripting. De mon point de vue :

- Bash décrit des **actions**. (ex : copie tel fichier, réalise telle action, n'autorise pas telle action ….)
- Ansible est accés sur **l'état de la machine** avant l'action. Il réalisera une tache si cela implique un changement d'état, sinon il ne l'executera pas.

Pour illustrer ces propos, je vais prendre l'exemple d'un script bash qui va installer **nginx** avec le module **passenger** qui permet de déployer des applications **Ruby on Rails** et transposer cela avec Ansible

Le script shell :

```
# Installation clé PGP
gpg --keyserver keyserver.ubuntu.com --recv-keys 561F9B9CAC40B2F7
gpg --armor --export 561F9B9CAC40B2F7 | apt-key add -

# Installation https support pour apt
apt-get install apt-transport-https -y

# Ajout du repo  passenger
echo "deb https://oss-binaries.phusionpassenger.com/apt/passenger raring main" >
    /etc/apt/sources.list.d/passenger.list
chown root: /etc/apt/sources.list.d/passenger.list
chmod 600 /etc/apt/sources.list.d/passenger.list

# Mise a jour du cache
apt-get update

# Installation nginx
apt-get install nginx-full passenger -y

# Configuration de passenger dans nginx
sed -i "s/# passenger_root/passenger_root/" /etc/nginx/nginx.conf
sed -i "s/# passenger_ruby/passenger_ruby/" /etc/nginx/nginx.conf
```

```
22
23 # Démarrage Nginx
24 service nginx restart
```

Le script Ansible :

```
1  ---
2  - hosts: all
3    tasks:
4
5    - name: Garantir que la cle PGP est installé
6      apt_key: >
7        state=present
8        id=AC40B2F7
9        url="http://keyserver.ubuntu.com/pks/lookup?op=get&fingerprint=on&search=0x561F9B9CAC40B2F7"
10
11   - name: Garantir que le support https pour apt est installé
12     apt: >
13       state=present
14       pkg=apt-transport-https
15
16   - name: Garantir que le répo pour passenger est ajouté
17     apt_repository: >
18       state=present
19       repo='deb https://oss-binaries.phusionpassenger.com/apt/passenger raring main'
20
21   - name: Garantir que Nginx est installé
22     apt: >
23       state=present
24       pkg=nginx-full
25
26   - name: Garantir que passenger est installé
27     apt: >
28       state=present
29       pkg=passenger
30       update_cache=yes
31
32   - name: Garentir que Nginx a la bonne configuration
33     copy: >
34       src=/app/config/nginx.conf
35       dest=/etc/nginx/nginx.conf
36
37   - name: Garentir que Nginx est démarré
38     service: >
39       name=nginx
40       state=started
```

Le script Bash va réaliser une **suite d'actions** sans contrôler l'état des actions du script. Si on relance le script, il va a nouveau effectuer toutes ces actions.

Ansible s'assure de **l'état désiré de la machine** grace à l'instruction **state** (état) qui est utilisée a chaque tâches. Si on relance, le script, Ansible ne changera rien

car il vérifie l'état avant d'agir et comme il n'y aura pas eu de changement d'état, les tâches ne seront pas à nouveau exécutées.

Ansible, à l'inverse de Bash, se soucis donc plus de l'état que de l'action. Cela permet d'avoir une gestion de la configuration en mode déclarative et idempotente et permet une gestion fiable de l'exécution à distance, avec des nouvelles tentatives, logiques évolutive, …

De plus le fait de pouvoir relancer le même Playbook plusieurs fois permet de surveiller les écarts de configurations.

Si un utilisateur venait à modifier la configuration d'un service, le fait de repasser le script Ansible va permettre de remettre la machine à l'état décris dans le script Ansible.

Installation d'Ansible

Ansible est disponible sur de nombreuses Distributions Linux. Il peut être installé par un gestionnaire de paquet ou par PIP car Ansible s'appuie majoritairement sur le langage Python. Pour l'installer sur CentOS, il faut configurer le contrôleur en ajoutant le bon répo puis en installant le bon paquet, qui va se charger d'installer les dépendances nécessaires

```
sudo yum install epel-release && sudo yum update
sudo yum install Ansible
```

On vérifie la bonne installation d'Ansible et des dépendances :

```
Ansible –version
```

et le resultat de cette commande :

```
Ansible 2.9.6
  config file = /etc/Ansible/Ansible.cfg
  configured module search path = ['/home/marc/.Ansible/plugins/modules',
      '/usr/share/Ansible/plugins/modules']
  Ansible python module location = /usr/lib/python3/dist-packages/Ansible
  executable location = /usr/bin/Ansible
  python version = 3.8.10 (default, Jun  2 2021, 10:49:15) [GCC 9.4.0]
```

Ansible a besoin que le port SSH soit ouvertert que l'authentification par clé plutôt que par mot de passe soit activée.

```
sudo firewall-cmd --list-services <- pour verifier que le service est ouvert
```

```
ssh-keygen <- on génère une clé ssh
ssh-copy-id "MACHINE_CLIENTE" <- on la copie sur les machines
```

Il est également recommandé d'accorder les droits nécessaires à l'utilisateur qui exécutera les commandes Ansible. Cet utilisateur doit être présent sur les machines clientes.

```
echo "UTILISATEUR_ANSIBLE ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers.d/UTILISATEUR
```

L'environnement de base est configuré. Plusieurs fichiers peuvent être modifié afin de changer le comportement d'Ansible.

Concepts de base

Avant de présenter le Playbook que j'ai réalisé, il est important de comprendre quelques éléments d'Ansible.

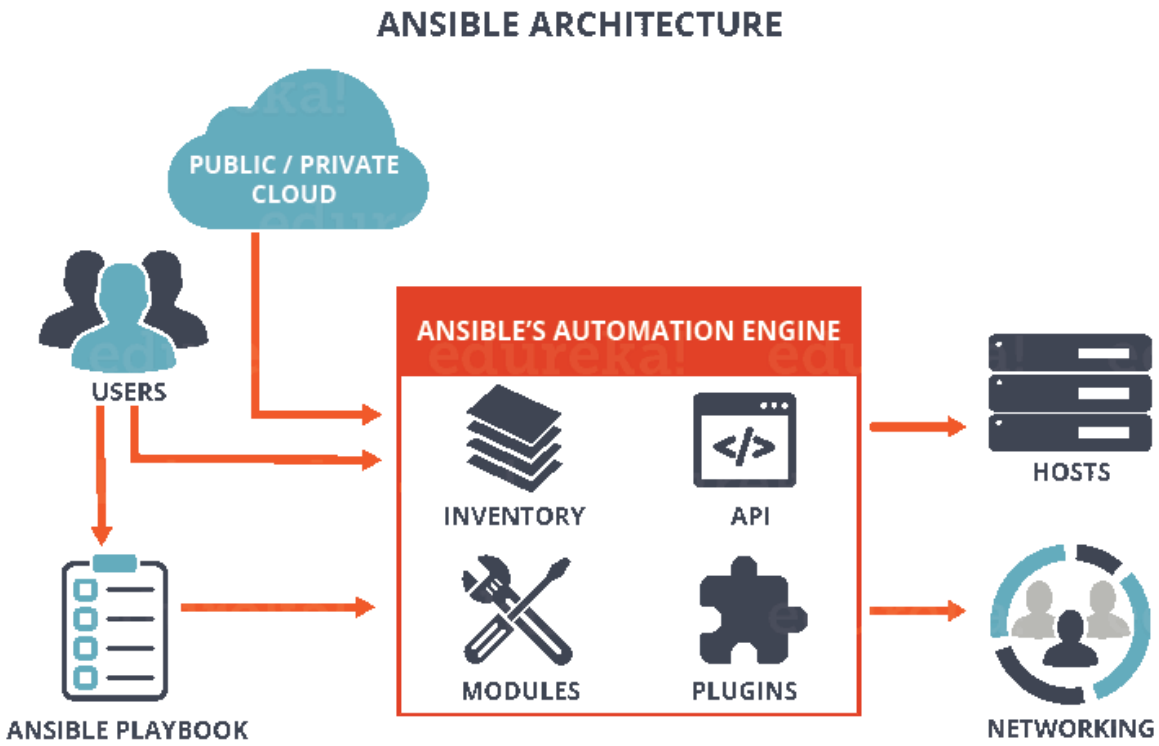

Figure 1 – Ansible

On définit des **rôles**, qui contiennent des **tâches** à exécuter à l'aide de différents **modules**, le tout regroupé dans un **Playbook**, qui va réunir les différents rôles et ou tâches. Le Playbook va donc executer un ensemble de **tache** , appelé également un **Play** sur un groupe de machines, définies dans un **inventaire**.

Il existe de nombreux modules et plugins qui permettent de réaliser presque toutes les action envisagable.

On trouve également des plugins et modules complémentaire sur **Ansible Galaxy** qui sont en libre téléchargement.

Pour ceux qui on une licence **Red Hat**, ils ont également accès à un large panel de rôles, modules et plugins spécialement consus pour les plateformes Red Hat.

Ansible utilise également des Templates, au format **Jinja2** afin de facilité la création de fichiers de configurations et la gestion des variables.

Il est de bonne pratique de créé un dossier par projet. Ce dossier va contenir plusieurs éléments.

Voici un exemple simple d'arborescence d'un projet, que j'ai adapté depuis la documentation officielle d'Ansible :

```
Playbook.yml

inventory/
    group_vars/
       group1.yml
       group2.yml
    host_vars/
       hostname1.yml
       hostname2.yml
    staging.yml
    production.yml

roles/
    common/
        tasks/
            main.yml
        handlers/
            main.yml
        Templates/
            ntp.conf.j2
        files/
            foo.txt
            bar.sh
        vars/
            main.yml
        defaults/
            main.yml
        meta/
            main.yml
    webtier/
    monitoring/
```

Il est important de respecter une structure et de s'y tenir car un projet peut contenir rapidement beaucoup de fichiers. Un projet Ansible comporte généralement les éléments suivants :

- Un fichier **playbook.yml** :
  Il va contenir l'ensemble des rôles et des tâches à exécuter.
- Un dossier **inventory** : Il va contenir généralement les inventaires et les dossiers où sont stockés les variables. On peut avoir 2 inventaires par exemple, un **staging.yml** pour les tests et un **production.yml** pour la production.

Les inventaires sont des fichiers en **.yml** ou **.ini** qui regroupe la liste des machines. Une machine peut appartenir à un groupe de machine, ou plusieurs groupes, ou aucuns. Les dossiers **group_vars** et **host_vars** sont des dossiers qui vont regrouper des variables qui seront appliquées à un group (group_vars) ou à une machine (host_vars).

- Un dossier **rôle** avec des sous dossiers pour les différents rôles. Chaque sous dossiers peut contenir les sous-dossiers suivants :
  - **/tasks/main.yml** : C'est ici que sont écrites l'ensemble des tâches que le rôle exécute
  - **/Template/NOM_DU_TEMPLATE.j2** : le dossier Template regroupe le/les Templates nécessaires pour le rôle
  - **/handlers/main.yml** : un handler est une tache inactive qui sera active seulement si elle est invoquée dans le fichier /tasks/main.yml grâce au mot clé **notify**
  - **/files** : ce dossier contient les fichiers nécessaires au fonctionnement du rôles comme des script bash, des liste csv, ….
  - **/default/main.yml** : contient les valeurs des variables par défaut du rôle
  - **/vars/main.yml** : contient d'autres variables, qui peuvent surcharger celle du /defaults/main.yml
  - **/meta/main.yml** : contient les métadata sur le rôle (auteur, licence, dépendances, …)

Le dossier **role** a une architecture qu'il convient de respecter. Il est possible de générer cette architecture avec la commande suivante :

```
ansible-galaxy init NOM_DU_ROLE
```

Les dossiers **/defaults** et **/vars** ne sont pas obligatoire car les dossier **group_vars** et **host_vars** servent à stocker les valeurs de variables.

Il y a également un fichier **ansible.cfg**, que l'on peut créer à la racine du projet qui permet de configurer plusieurs aspects d'Ansible comme par exemple le fait de ne pas vérifier les clés SSH pour chaque hôte, on peut lui ajouter la ligne suivante :

```
host_key_checking = False
```

On peut également définir les paramètre pour l'élevage des privilège de la facon suivante :

```
remote_user: Ansible

[privilege_escalation]
become.=.true
become_method.=.sudo
become_ask_pass.=.False
```

Cela nous permet de configurer le comportement général pour l'utilisateur distant qui est utilisé pour se connecter et exécuter les commandes ainsi que l'élevage des privilèges.

Ce fichier est très riche et on peut modifier le comportement **général** d'Ansible.

Quelques commandes ad-hoc utiles

Ansible dispose de nombreuses commandes qui permettent de debugger, trouver des informations sur un module, exécuter une action rapidement et qui est employé rarement, sans le besoin d'écrire un rôle ou un Playbook.

```
ansible-inventory --graph --vars
```

Cette commande va nous fournir la liste des hosts ainsi que les variables qui leurs sont attribuées.

Le site D'Ansible dispose de nombreuses informations sur les modules et leur utilisation. Cependant une commande existe qui permet d'avoir de la documentation rapidement dans le terminal

```
ansible-doc
ansible-doc | wl -l --> "plus de 3000 modules"
ansible-doc "NOM_DU_MODULE"
```

Avec ces commandes, on arrive à trouver beaucoup d'information sur les spécificités de chaque module et avec des exemples. C'est l'équivalent des pages MAN sous Linux mais pour Ansible.

Avec l'aide de la documentation, on peut copier des fichiers / dossiers / archives sur toutes les machines avec une simple ligne de commande ou encore installer un paquet sur toutes les machines :

```
ansible "NO_DU_GROUPE" -m copy -a "src=/etc/hosts dest=/tmp/hosts"

ansible all -m ansible.builtin.yum -a "name=nginx state=latest"
```

Les commandes ad-hoc d'Ansible on généralement la même syntaxe : - **ansible** - **Nom du groupe de machines** ou de **la machine** ou **ALL** pour tous ou **UNGROUPED** pour les machines sans groupes - **-m** pour préciser le module que nous voulons utiliser, ici le module **copy** pour copier un fichier du contrôleur sur la/les machines ou **yum** pour installer Nginx sur toutes les machines - entre parenthèse les **arguments** du module à exécuter

Par défaut si nous ne précisons pas de module dans la commande, le module par défaut utilisé sera le module **command** , qui permet de lancer des commande BASH (sans pouvoir utiliser la puissance du Bash comme le pipe, …)

```
ansible all -a "free -m"
```

Avec cette commande, on peut très rapidement obtenir des informations sur la mémoire libre de toutes les machines. On comprend très vite le gain de temps pour faire du debuggage sur un parc de machines. En une commande on a récupéré les informations sur le parc de machines, sans avoir à faire de ssh et de taper la commande, sur chaque machine…

Une autre commande très utile pour un administrateur réseau :

```
ansible all -m listen_ports_facts -i prod-ansible-hosts
```

Ici, on utilise la puissance du module **listen_ports_facts** afin de trouver les informations sur les ports ouverts sur chaque machine. C'est l'équivalent d'une commande Netstat, SS ou NMAP.

Un dernier exemple très utile :

```
ansible all -m setup
```

Cette commande va nous retourner énormément d’information sur les machines ou seront exécuté la commande. La sortie de cette commande est en **JSON**. Ce qui permets de pouvoir filtrer cette commande afin de recherche précisément une information.

JSON est le format principal de sortie pour toutes les commandes d’Ansible.

Monitoring

Le besoin

Afin de s'assurer de la bonne santé d'une infrastructure, il est important de la surveiller, de mettre des alertes sur des seuils que l'administrateur va définir afin d'être prévenu de tout changement d'état. Cela permet de voir venir un problème et d'anticiper plutôt que de corriger dans l'urgence et le stress.

Il m'a été demandé de travailler sur le déploiement d'une solution de monitoring utilisant des logiciels open sources, moderne, robuste et facilement scalable au cas ou infrastructure devait évoluer.

L'infrastructure à surveiller

Cette solution de monitoring va surveiller plusieurs éléments d'une infrastructure d'une trentaine de VM qui comprend :

- Serveurs d'applications (Jupyter, Moodle, Drupal, Peertube, …)
- Serveurs web Nginx et Apache
- Plusieurs BDD (MariaDB, MongoDb et PostgreSQL)

Etant donnée la composition de l'infrastructure très variée, nous voulons pouvoir récupérer les information suivantes :

- **statistiques par machine** :
  - Mémoire
  - CPU
  - Uptime
  - Stockage
  - Disk I/O
- **serveur web Nginx et Apache** :
  - Load
  - Network I/O
  - Traffic
  - Différentes requêtes
  - Nombres de connexions

On pourra reconfigurer très facilement notre stack pour monitorer les différentes BDD en surveillant :

- Erreurs
- SQL commands/sec
- Heatmap (queries/sec) cache

On veut également pouvoir analyser des logs important suivants :

- Logs système (cron - access.log - audit.log …)
- Logs serveurs web (seulement Nginx dans le playbook que je présente)
- Logs applicatifs (Peertube, Moodle pour le moment)

Dans un second temps, on configurera la récupérations des logs ded autres applications.

La solution de monitoring

La solution de monitoring retenue a été la suivante :

- **Grafana** pour la centralisation des graphiques
- **Influxdb** comme base de données pour les différentes métriques
- **Telegraf** pour la collecte des métriques
- **Loki** pour la gestion des logs
- **Promtail** pour la récupération des logs

Grafana et Influxdb et Loki seront installé seulement sur le serveur de monitoring. Promtail et Telegraf seront installés sur toutes les machines pour faire remonter les données au serveur.

Cette solution est facilement transposable pour une autre infrastructure. Je vais présenter les briques de bases qui permettent de monter cette solution de monitoring mais avec un peu de temps, on peut très rapidement reconfigurer le Playbook pour convenir aux besoins d'une autres infrastructure.

Grafana Labs

Grafana Labs est une entreprise spécialisé dans la création d'outils de visualisation de métrique Open source. Elle propose également des services pour la gestion et la collectes de logs. Voici une liste de service proposé par Grafana Labs :

- Grafana
- Graphite
- Loki
- Metrictank
- Prometheus
- Tanka
- Tempo
- k6

Grafana

```
1 github.com/grafana/grafana
```

Grafana est un outil supervision moderne et open source. Il permet d'exposer sous formes de **dashboards** (tableau de bord) les métriques brutes ou agrégées provenant d'un **datasource** comme Influxdb pour les métriques Loki pour les logs.

L'une de ses grandes forces est qu'il permet de créer très facilement des seuils d'alertes et les actions associées comme l'envoie de mail pour alerter l'administrateur du S.I

Grafana dispose d'une **WEBUI**. Ce qui est très utile quand on veut monitorer une infrastructure à distance. On l'installe seulement sur le serveur de monitorings et on y accède en https de n'importe où.

Une version payante est également disponible ainsi qu'une version Cloud.

C'est pourquoi on trouve souvent le comba Grafana - Prometheus - Loki (du même prestataire) + Influxdb et Telegraf

Loki

```
https://github.com/grafana/loki
```

Loki est un agrégateur de logs, facilement scalable et inspiré de Prometheus (un autre outil de monitoring qui peut remplacer Influxdb dans la stack).

Loki utilise un mécanisme de découverte de service et ajoute des labels aux logs au lieu de les indexer, ce qui rend facile leur manipulation et ordonne leur stockage (ex : création de groupes en fonctions des labels, utilisation de règles spécifiques en fonction des labels, …)

Les logs reçus de Promtail se composent du même ensemble de labels que celui des métriques d'applications que Telegraf récupère. Ce qui permet une meilleure intégration des logs et des métriques.

De plus, Loki a besoin de peu de ressources pour fonctionner.

Promtail

```
https://github.com/grafana/loki/releases
```

Promtail est un agent qui expédie les logs vers une instance Loki. Il est déployé sur chaque machine sur laquelle des applications doivent être surveillées. Il fonctionne en 3 temps :

- Découverte des cibles (ce que Promtail doit récupérer en terme de logs)
- Attache des tags aux logs pour pouvoir les identifier et les rapprocher facilement
- Pousse les logs vers une instance Loki.

Promtail est très customisable. Nous verrons plus loin quelques exemples de configuration.

Influxdata

Influxdata est la société qui développe des solution de monitoring Influxdb et Telegraf. Elle est spécialisé dans la collecte et la gestion de metriques.

Influxdb

```
https://github.com/influxdata/influxdb
```

Influxdb est une Time Series Database (TSDB) écrite en Go. Ce type de bases de données est employée notamment pour stocker et analyser des données de capteurs ou des logs sur une période donnée. Ces données doivent être traitées rapidement une fois entrées dans la base de données.

Influxdb intègre un service qui repose sur le protocole NTP Network Time Protocol, pour assurer que l'heure est bien synchrone sur l'ensemble des systèmes et que les logs sont bien traités.

Ces principaux avantages sont :

- Les performances : Elle peut gérer un volume massif d'information

- La durée de rétention importante : capable de gérer la gestion des donnée sur une longue période grace à une excellente gestion des données (taille, compression, …)
- La scalabilité : facilement adaptable pour gérer la charge

Influxdb est une base de données temporelle, à la différence des bases de données relationnelles comme MySQL ou Mariadb. Ce type de base de données idéal quand on doit manipuler des données temporelles comme la mesure de la température du CPU toutes les 10 secondes.

Ce type de BDD permet de traiter une très grande quantité d'informations, et dans un temps très courts, la gestion des données est différente à celle d'une base de données relationnelle.

Les bases de données temporelles disposent de règles de retentions que l'administrateur décide afin de choisir la quantité d'information à stocker/recycler.

Une version Cloud et payante sont également disponibles.

Telegraf

```
https://github.com/influxdata/telegraf
```

Telegraf est un agent de récupération de métriques. Un seul agent est nécessaire par machine. Cet agent sait récupérer des métriques exposées et propose 2 modes de récupération :

- Push : la métrique est poussée dans Telegraf par le composant qui l'expose
- Pull : Telegraf récupère la métrique en interrogeant le composant qui l'expose (le mode le plus utilisé)

Les métriques sont par la suite insérées dans la Base de données Influxdb

Sa force réside dans la grande bibliothèque de plugins disponible afin de pouvoir récupérer les informations. Il peut récupérer des données depuis des Bases de données, des IoT, des sondes (températures, pression de l'air, …) et des applications. C'est là que les plugins vont être très avantageux afin de paramétrer facilement la récupération des informations.

Telegraf est écrit en Go et il est disponible dans un seul binaire sans besoins de dépendances ou besoin d'outils des gestionnaires de paquets (npm, pip, gem, …)

Il est souvent associé à Influxdb (même prestataire) ou Nagios, Prometheus, Graphite ou directement en **JSON** pour pouvoir être interpréter par un logiciel sur-mesure par exemple.

Les autres Solutions de monitoring

D'autres solutions existes comme **Zabbix**, **Elastic Search**, **Centreon**, …

Mise en place du projet

Afin de réaliser ce projet, nous allons créer un dossier avec tout les éléments nécessaire au bon déroulement du script Ansible. Afin de pouvoir contrôler et versionner ce projet, il sera mis en place avec Gitlab.

Cela permettra également un e meilleure collaboration avec les différentes personnes de l'équipe.

Création du Projet

Notre projet peut se résumer en :

- 1 dossier projet
- 1 Playbook
- 1 inventaire
- 1 dossier pour la gestion des variables avec des sous dossiers pour surcharger les variables pour les sous-groupes/
- 1 dossier rôles avec les **5** rôles d'installation des **5** briques logicielles

Arborescence de notre projet

voici l'arborescence de notre projet

```
.
|--- grafana_dashboard.json
|--- inventory
|    |--- group_vars
|    |   |--- all.yml
|    |   |--- apache
|    |   |      --- main.yml
|    |   |--- app
|    |   |      --- main.yml
|    |   |--- job
|    |   |      --- main.yml
|    |   |--- nginx
|    |   |      --- main.yml
|    |   |--- peertube
|    |          --- main.yml
|    |--- host.yml
|--- playbook.yml
|--- roles
    |----install_grafana
    |    |--- handlers
    |    |      --- main.yml
    |    |--- tasks
    |    |      ---main.yml
    |    |--- templates
    |           --- grafana_conf.j2
    |           --- grafana_dashboard.json.j2
    |           --- grafana_provisioning.j2
    |           --- grafana.service.j2
```

```
    |--- install_influxdb
    |    |--- handlers
    |    |      --- main.yml
    |    |--- tasks
    |    |      --- main.yml
    |    |--- templates
    |           --- influxdb_conf.j2
    |           --- influxdb.service.j2
    |--- install_loki
    |    |--- handlers
    |    |      --- main.yml
    |    |--- tasks
    |    |      --- main.yml
    |    |--- templates
    |           --- loki_conf.j2
    |           --- loki.service.j2
    |--- install_promtail
    |   |---handlers
    |   |     ---main.yml
    |   |--- tasks
    |   |      --- main.yml
    |   |--- templates
    |          ---promtail_conf.j2
    |          --- promtail_conf.j2.bak
    |          --- promtail.service.j2
    |--- install_telegraf
        |--- handlers
        |      --- main.yml
        |--- tasks
        |      --- main.yml
        |--- templates
               --- telegraf_conf.j2
               --- telegraf.service.j2

28 directories, 32 files
```

Mise en place des différents éléments

Cette stack peut être très facilement installé grâce à Docker et par ailleurs c'est l'une des solutions les plus utilisés pour monitorer des infrastructure Conteneurisé et dans le Cloud.

Personnellement, j'utilise cette solution conteneurisée, le tout orchestré avec K8S pour monitorer mon homelab.

Le choix fait par CGI et d'éviter la conteneurisation pour les environnements de production dans un soucis de sécurité et de stabilité. Nous sommes donc partis sur une installation en dur des différentes briques de cette solution qui sera déployée par Ansible.

Afin de ne pas divulguer d'informations sensible, j'illustrerai par des graphiques de mon homelab et présenterez dans ce rapport des morceaux du Playbook que je juge

important pour la compréhension du déploiement de cette solution de monitoring.

Ansible utilise le format **YAML** qui permet une lecture facile des différents éléments du Playbook.

Une version du Playbook est disponible sur mon compte Github.

```
https://github.com/marc-cenon/rapport_de_stage/tree/master/files/monitoring_stack/ansible_grafana_v2
```

Il est fonctionnel, idempotent et peut être utilisé avec peu de modification pour monitorer sa propre infrastructure.

Creation des différents éléments du projet

Les différents rôles

Ce Playbook utilise **5 rôles** afin d'installer et de configurer les différentes briques nécessaires pour déployer la stack correctement. Les 5 rôles sont les suivants :

- install_grafana
- install_influxdb
- install_loki
- install_telegraf
- install_promtail

Dans ces rôles, je fait appels à des **handlers** pour le redémarrage des services a un moment défini, des **templates** pour créer les services pour gérer les logiciels avec systemd

Grafana

Les étapes du rôle d'installation de Grafana sont simples. Avec l'aide des modules adéquats d'Ansible, les étapes pour l'installation et la configuration (les différentes tâches du rôles) de Grafana sont les suivantes :

- création du groupe et du compte grafana :monitoring
- création des dossiers nécessaires
- téléchargement du programme et extraction dans le dossier d'installation définie au préalable
- création d'un fichier de configuration grâce à un Template
- import d'un dashboard existant que j'ai créé
- ouverture des ports dans le firewall
- création du fichier .service à l'aide d'un Template
- activation du service et redémarrage

Pour ce rôle, l'utilisation de **Templates** pour générer le fichier de configuration de Grafana et le service associé permettent de simplifier le processus d'installation. Cela permet également de pouvoir modifier rapidement et facilement le rôle en ajustant les variable adéquate dans le fichier /inventory/group_vars/all.yml.

Voici la tâche du rôle Grafana qui utilise le Template crée pour générer le fichier service :

```
- name: "copy Grafana systemd service from Template"
  template:
    src: Grafana.service.j2
    dest: /etc/systemd/system/Grafana.service
```

On utilise le module **template**, qui va chercher le fichier Grafana.service.j2 dans le dossier **grafana.service.j2** et qui va utiliser les valeurs définis dans le fichiers de variable dans /inventory/group_vars/all.yml.

Voici le Template utilisé pour créer le service :

```
[Unit]
Description=Grafana
Wants=network-online.target
After=network-online.target
After=postgresql.service mariadb.service mysql.service

[Service]
Type=simple
User={{ grafana_account_name }}
Group={{ grafana_account_group }}
RuntimeDirectory=grafana
RuntimeDirectoryMode=0750
WorkingDirectory={{ grafana_main_folder }}/grafana
ExecStart={{ grafana_main_folder }}/grafana/bin/grafana-server
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Les parties intéressantes de ce Template sont les parties entre les accolades **{{ }}**. La variable {{ grafana_account_name }} va être remplie par la valeur dans le fichier de variable dans **/inventory/group_vars/all.yml**

Bien que Grafana (comme Loki et Influxdb) sont installés sur une seule machine, afin de simplifier le Playbook et pour éviter d'avoir trop de fichiers d'inventaire, un groupe comportant une seule machine, celle du monitoring est créée dans l'inventaire.

De ce fait, nous pouvons définir les variables dans le seul fichier **/inventory/group_vars/all.yml** où se trouve la majorité des variables. Cela évite d'utiliser le dossiers **host_vars** et d'avoir un autre fichier avec des variables pour le serveur de monitoring.

Le risque est qu'avec trop de fichiers de variables, il peut être difficile de s'y retrouver et de savoir où se trouve les bonnes variables, et de ne pas surcharger les variables par erreur.

En fonction d'où se trouve le fichier qui contient les variables dans l'arborescence du projet, il y a une hiérarchie qui, si ignorée peut poser des problèmes.

Nous pouvons utiliser des "boucle" **loop**, comme dans un langage de programmation, pour répéter une même action dans une tâche avec des variables différentes. Voici un exemple pour l'ouverture des ports dans le firewall :

```
- name: "open firewall port 3000 on the machine and port 25 for SMTP email"
  firewalld:
    state: "{{ item.state  }}"
    port: "{{ item.port  }}"
    zone:
    immediate:
    permanent: yes
  with_items:
    - { state: 'enabled', port:'3000/tcp'  }
    - { state: 'enabled', port:'25/tcp'  }
```

Ici, on utilise le module **firewalld** et la fonction **with_items** (qui est identique a **loop**) qui va itérer sur les **{{ item }}** en appliquant les valeurs définis pour **state** et **port**, c'est-à-dire l'état et le port.

Avec ces quelques lignes, on ouvre les ports, dans la zone par défaut (car nous n'avons pas renseigné de zone spécifique dans zone), de manière permanente et immédiate.

Influxdb

Les étapes pour l'installation d'Influxdb (la liste des tâches) sont sensiblement identique à celle de Grafana :

- création du groupe et du compte influxdb :monitoring
- créations des dossiers nécessaires
- téléchargement du programme et extraction dans le bon dossier
- création d'un fichier de configuration et du service à partir d'un Template
- ouverture des ports dans le firewall
- activation du service
- pause de quelques secondes, le temps que la BDD soit opérationnelle
- configuration d'Influxdb en passant une commande shell avec les paramètres définis dans le fichier de variable

La difficulté ici et la dernière étape pour automatiser la configuration d'Influxdb, on passe une commande shell, avec des arguments issus de variables définis dans group_vars/all.yml pour la création des éléments nécessaires à Influxdb.

```
- name: "check if folder exist"
  stat:
    path: "{{ Influxdb_main_folder  }}/.Influxdbv2"
  register: folder_exist

- name: "configure Influxdb as Influxdb user and not root"
  become_user: "{{Influxdb_account_name}}"
  shell: >
    {{ Influxdb_main_folder  }}/Influxdb/influx setup --org {{ Influxdb_organization  }} --bucket
        {{ Influxdb_bucket  }} --username {{ Influxdb_username  }} --password {{ Influxdb_password
        }} --token {{ Influxdb_token  }} --force
  when: not folder_exist.stat
```

La condition **when** est intéressante. Elle permet de s'assurer que le rôle se déroule bien car si on essaie de configurer la base de données alors que le dossier de

configuration est déjà présent, la tâche va échouer et le Playbook ne sera pas déroulé dans son intégralité.

La valeur de la condition **when** est **not folder_exist.stat**. Dans la tâche du dessus, on utilise le module **stat** afin de récupérer des informations sur le dossier de configuration d'Influxdb et on stocke le résultat dans la variable **folder_exist** grâce à la commande **register**

Et comme le format de sortie de toutes les instructions d'Ansible est le JSON, il suffit de filtrer la variable **folder_exist** pour récupérer la valeur de la clé **stat** qui contient la location du dossier de configuration.

La tâche est donc lancée quand le dossier de configuration n'est pas présent.

L'un des principes d'Ansible, comme expliqué plus haut est l'idempotence. On peut lancer le Playbook autant de fois qu'on le souhaite est rien de sera modifié si rien n'a changé dans le Playbook car Ansible est axé sur l'état désiré de la machine et avant l'action.

Le point que je souhaitais mettre en avant ici est la facilité avec laquelle on peut définir des conditions pour lancer, ou non des tâches sans grande connaissance en programmation et la facilité de lire le code grâce au format YAML

Depuis la version 2.0 d'Influxdb, le langage de requête InfluxQL a été remplacé par le langage **Flux**, qui est plus performant.

Flux est une alternative à InfluxQL et à d'autres langages de requête de type SQL pour interroger et analyser des données. Il utilise des modèles de langage fonctionnels, ce qui le rend capable de surmonter bon nombre des limitations d'InfluxQL. Sa syntaxe est en partie inspiré de Javascript.

Quelques notions importantes pour pouvoir écrire des requêtes avec Flux :

- Utilisation de **pipe forward" |>** pour enchainer des actions
- Toutes les données sont structuré sous forme de tableau.
- Un regroupement de tableaux avec une politique de rétention est un Bucket.

Voici quelques exemples de requêtes en langage Flux :

- Nombre de processus par machine :

```
from(bucket: "bucket-vm")
  |> range(start: 2021-07-05T02:28:35Z, stop: 2021-07-05T08:28:35Z)
  |> filter(fn: (r) => r["_measurement"] == "processes")
  |> filter(fn: (r) => r["_field"] == "total")
  |> group(columns: ["host"])
  |> aggregateWindow(every: 20s, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

- Utilisation du CPU par machine :

```
from(bucket: "bucket-vm")
  |> range(start: 2021-07-05T02:29:36Z, stop: 2021-07-05T08:29:36Z)
  |> filter(fn: (r) => r["_measurement"] == "cpu")
  |> filter(fn: (r) => r["_field"] == "usage_system")
```

```
  |> filter(fn: (r) => r["cpu"] == "cpu-total")
  |> group(columns: ["host"])
  |> aggregateWindow(every: 20s, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

Influxdb dispose également d'une **WEBUI** qui permet de faciliter grandement la création de requêtes complexes. Il suffit de choisir les critères dans le menu et d'importer la requête dans Grafana, qui nous permettra de visualiser le résultat avec un graphique très customisable.

L'ensemble des requêtes du Playbook est également disponible dans le fichier **dashboard.json**. Egalement en annexe un exexemple de requete avec Influxdv Lien (ici)[#requête-influxdb]

Flux est un langage très puissant mais l'interface d'Influxdb permet d'arriver au même résultat rapidement et de gérer les buckets (équivalent à une database) et la politique de rétention des données très facilement sans avoir à maîtriser Flux.

En effet, il est important de gérer la rotation du stockage des données car en fonction du nombre de machines, du nombre de critères de monitoring et de l'intervalle de récupération des métriques, le volume de donnée stocké peut rapidement être important.

Telegraf

Pour compléter notre stack TIG, il faut également déployer nos agents grâce en charge de récupérer les différents métriques grâce au rôle Telegraf. Il installera sur toutes les machines à surveiller l'agent. Les étapes du rôle sont les suivantes :

- Création du groupe et du compte telegraf :monitoring
- Création des dossiers nécessaires
- Téléchargement et extraction dans le bon dossier
- Création d'un fichier de configuration et d'un service avec un Template
- Activation du service

Les étapes sont sensiblement les mêmes que pour Grafana et Influxdb. Le point important ici est le fichier de configuration. Une partie de la configuration sera la même pour toutes les machines. On va récupérer par exemple :

- %CPU
- %RAM
- Uptime
- %SDD

En fonction des spécificités des machines, la configuration sera à affiner pour récupérer des métriques spécifiques comme des métriques sur Nginx, Apache, Mariadb, PostgreSQL, Moodle, Peertube, …

Pour cela, deux stratégies sont possibles :

- Déployer la même configuration sur toute les machines et ajouter la configuration spécifique manuellement … ce qui ne paraît pas logique quand on est dans une démarche d'automatisation avec une démarche IaC.
- Créer des sous dossiers dans group_vars ou host_vars (si déploiement d'une config

spécifique à une machine) avec dedans un fichier avec les variables nécessaires à la configuration spécifique des machines.

C'est le deuxième choix qui semble le plus avantageux et le plus logique d'un point de vue automatisation.

Quand il y a de la configuration spécifique à un groupe de machine, il suffit de définir les variables adéquates dans un fichier de variables qui se trouve dans un dossier qui porte le nom du groupe de machine dans le dossier **group_vars**.

Par exemple, pour le groupe de machine **Peertube**, nous avons le dossier **inventory/group_vars/peertube**

Dans ce dossier le fichier **main.yml** comprend les valeurs des variables qui seront appliquées seulement aux machines du groupe Peertube, qui sont définies dans le fichier **/inventory/host.yml**.

Ainsi, on peut déployer en une seule fois une application, avec une configuration de base à toute les machines avec en plus une configuration spécifique à un groupe de machine.

C'est également ce fonctionnement qui sera utilisé pour le déploiement de la configuration de Promtail.

Promtail

L'installation de Promtail suit le même schéma que Telegraf. Comme cet agent sera déployer sur toute les machines, il y aura un bout de configuration commune et un autre spécifique à un groupe de machine.

La configuration spécifique se trouve dans le même fichier que pour les configurations spécifiques de Télégraf.

La configuration de Promtail utilise des **Scrape Job**. On va configurer dans un fichier yaml exactement ce que Promtail doit récupérer comme logs, l'endroit, le nom que l'on donne au scrape job ainsi qu'un label qui sera utilisé plus tard pour classer les logs et appliquer des règles de trie par exemple pour pouvoir les regrouper avec d'autres logs ou avec des métriques de Telegraf.

On peut créer un dashboard pour Loki qui va regrouper par exemple dans une fenêtre tous les logs d'erreurs de toute les machines, une autre fenêtre avec tous les logs cron de toute les machines.

Voici un exemple de configuration de Promtail pour récupérer les logs de cron :

```
#scrape job for cron log
- job_name: cron
  static_configs:
  - targets:
      - localhost
    labels:
      job: cron
      __path__: /var/log/cron
```

On a bien défini dans cet exemple le nom du scrape job (job_name), la machine ou Promtail doit récupérer les logs (localhost) et le chemin d'accès des logs.

En voici un autre pour les logs Nginx :

```
- job_name: nginx
    entry_parser: raw
    static_configs:
      - targets:
          - localhost
        labels:
          job: nginx
          __path__: /var/log/nginx/*log
```

Dans notre infrastructure à surveiller, plusieurs serveurs Nginx sont déployés. Grâce au label que nous allons utliser pour les logs Nginx (nginx) on va pouvoir regrouper tous les logs qui on ce label dans une même fenêtre. La valeur **targets :localhost** prendra le nom de la machine **/etc/hostname** afin de savoir a quelle machine appartient quel log.

Il est également possible de filtrer et de formater les logs avec des règles spécifiques a définir dans les scrape jobs. Le github de Promtail regorge d'information sur la configuration et le paramétrage de la récupération des logs.

Un Template est utilisé pour deployer les **Scrape Jobs** en fonction des différents groupes de machine. Le Template est dans le dossier **template** du rôle Promtail et les variables sont définies dans les sous-dossiers qui portent le nom de chaque groupe, dans le dossier **group_vars**.

Il faut s'assurer que Promtail a les **droits nécessaires** pour lire les logs que nous voulons remonter dans Loki puis Grafana. Bien souvent les logs système sont définis avec un **mod 640**.

Il faut donc penser à configurer les autorisations nécessaires pour Promtail.

Loki

L'installation de Loki est identique à celle de Grafana et de Promtail. Il n'y a pas de difficultés majeures. Le point intéressant dans le rôle est le fichier de configuration de Loki qui est déployé grâce à un Template.

Voici la tâche qui utilise le module **template** pour déployer le fichier de configuration :

```
- name: "create custom loki configuration file from template"
  template:
    src: loki_conf.j2
    dest: "{{ loki_main_folder }}/loki/custom.yml"
    mode: 0755
    owner: "{{ loki_account_name }}"
    group: "{{ loki_account_group }}"
  notify:
    - restart loki
```

On source le fichier template qui se trouve dans le sous dossier template dans le dossier du rôle **install_loki**, en lui indiquant en destination l'emplacement où le fichier doit se trouver et en appliquant les bon droit sur ce fichier. Une fois le fichier créé, nous utiliser un **notify** qui va alerter le **handler** de redémarrer le service de Loki.

Le template pour la configuration de Loki tronqué :

```
ingester:
  wal:
    enabled: true
    dir: {{ loki_main_folder }}/loki/wal

storage_config:
  boltdb_shipper:
    active_index_directory: {{ loki_main_folder }}/loki/boltdb-shipper-active
    cache_location: {{ loki_main_folder }}/loki/boltdb-shipper-cache
    cache_ttl: 24h         # Can be increased for faster performance over longer query periods,
        uses more disk space
    shared_store: filesystem
  filesystem:
    directory: {{ loki_main_folder }}/loki/chunks

compactor:
  working_directory: {{ loki_main_folder }}/loki/boltdb-shipper-compactor
  shared_store: filesystem

ruler:
  storage:
    type: local
    local:
      directory: {{ loki_main_folder }}/loki/rules
  rule_path: {{ loki_main_folder }}/loki/rules-temp
  alertmanager_url: http://localhost:9093
  ring:
    kvstore:
      store: inmemory
  enable_api: true
```

Quelques élements pour la compréhension pour la configuration de Loki :

- **ingester** :
  Ce service est responsable d'arranger les logs pour les placer dans une base de données, généralement de type NoSQL comme Cassandra ou DynamoDB
- **boltd-shipper** :
  C'est la base de données, écrite en Go, embarqué par le service Loki pour gérer les logs.
- **compactor** :
  Il s'agit d'un module, specifique à boltd-shipper afin d'améliorer ses performances. Grafana Loki conseille d'activer ce service.
- **ruler** :
  Ce service est est chargée d'évaluer en permanence un ensemble de requêtes configurables et d'effectuer une action en fonction du résultat.

conclusion sur les rôles

Ansible s'appuie sur des modules. Il se peut que dans certains cas la configuration d'un service ne puisse se faire avec un module car il n'existe pas. Ansible dispose alors de 3 modules qui vont permettre de contourner ce problème. Il s'agit des modules :

- **raw** :
  Exécute une commande de bas niveau. Très utile pour déployer de la configuration sur des machines dépourvu d'interpréteur, ou sur des machines spécifiques comment des switchs, routeurs, …
- **shell** :
  Exécute une commande sur une machine distante dans un SHELL en s'appuyant sur la force du SHELL (ex : le **pipe** n'est pas possible avec command)
- **command** :
  Exécute une commande sur une machine distante

Dans le cas du rôle Influxdb, la configuration ne peut se faire qu'avec une commande **SHELL** et la condition **WHEN** permet de s'assurer que le Playbook n'échoue s'il est relancé car la BDD est déjà configurée.

Cependant, j'ai découvert récemment qu'il existe un package « communautaire » disponible sur Ansible Galaxy qui rajoute des modules à la liste par défaut. Il est possible de l'installer avec la commande suivante :

```
ansible-galaxy collection install community.general
```

Dans le pack de module supplémentaire, on peut trouver le module **community.general.influxdb_database** qui peut également nous permettre de configurer Influxdb.

L'inventaire

C'est l'un des fichiers les plus important. C'est dans ce dernier que l'on va définir la liste des machines que nous voulons intégrer à notre Playbook. Il peut être au format **.ini** ou **.yml**

Voici un exemple de fichier **hosts.yml** qui est utiliser pour réaliser des actions sur les machines spécifiques :

```
all:
  children:
    monit:
      hosts:
        monitoring-vm1:
          Ansible_host: 192.168.0.1
    clients:
      children:
        bdd:
          hosts:
            Moodle-bdd-vm1:
              Ansible_host: 192.168.0.2
            springboard-bdd-vm2:
              Ansible_host: 192.168.0.3
```

```
        nginx:
          hosts:
            springboard-nginx01:
              Ansible_host: 192.168.0.4
            springboard-nginx02:
              Ansible_host: 192.168.0.5
        apache:
          hosts:
            Moodle-apache01:
              Ansible_host: 192.168.0.6
```

On a beaucoup de flexibilité et de modularité dans le fichier **host.yml** pour créer des groupes et des sous-groupes. Cela nous permet de pouvoir déployer de la configuration avec une très grande précision et de cibler une machine ou un groupe de machines.

Le dossier de variables

Le dossier **group_vars** contient l'ensemble des variables du projet. Il y a un fichier **all.yml** qui comprend les variables utilisées pour l'ensemble des machines ainsi que des sous dossiers spécifiques aux groupes de machines pour déployer de la configuration spécifiquement à un groupe. Voici un extrait du fichier all.yml

```
# remote user account
user: marc


##################################
#        grafana                 #
##################################

# keep in mind to check that the template is still valid when upgrading

grafana_account_name: grafana
grafana_account_shell: /bin/bash
grafana_account_group: monitoring
grafana_main_folder: /appli/monitoring/grafana
grafana_download_url: https://dl.grafana.com/oss/release/grafana-7.5.4.linux-amd64.tar.gz

grafana_protocol: http
grafana_domain_name: monitoring-vm.fr
grafana_default_login: admin
grafana_default_password: admin_pass

### SMTP configuration for grafana ###
grafana_email: grafana-monitoring@admin.fr

##################################
#         influxdb               #
##################################

influxdb_account_name: influxdb
influxdb_account_shell: /bin/bash
```

```
influxdb_account_group: monitoring
influxdb_main_folder: /appli/monitoring/influxdb
influxdb_download_url:
    https://dl.influxdata.com/influxdb/releases/influxdb2-2.0.6-linux-amd64.tar.gz

# influxdb setup
influxdb_organization: organization-bucket1
influxdb_bucket: bucket-bdd
influxdb_username: influxdb
influxdb_password: influxdb
influxdb_access_url: http://localhost:8086
influxdb_token: tokentokentokentokentokentokentoken
```

On définie les variables sous forme clé :valeur. Il est tout à fait possible de définir une variable avec une liste ou dictionnaire comme valeur. Cela est très utile par exemple pour créer plusieurs utilisateurs.

Voici un exemple :

En variable :

```
users :
 - username : marc
   group : admin
 - username : ludivine
   group : student
```

Et la tâche pour créer les utilisateurs, avec leur dossiers, le bon groupe :

```
- name : create users
  user :
    name : "{{ item.username }}"
    group : "{{ item.group }}"
  loop : "{{ users }}"
```

la tâche va boucler **loop** sur la variable **users** et parcourir la liste afin de récupérer les valeur pour **username** et **group**.

Le fichier playbook

Le Playbook va regrouper les différents rôles afin de les exécuter à la suite. Voici comment le rôle Grafana est appelé dans le Playbook :

```
- name: "install Grafana"
  remote_user: "{{ user  }}"
  become: true
  hosts: monit
  tags: [Grafana]
  roles:
    - role: install_Grafana
```

Plusieurs éléments sont importants quand on appelle un rôle dans un Playbook :

- **remote_user** :
  C'est l'utilisateur qui est utilisé pour se connecter à distance et effectuer les actions qui ne demande pas de privilège.
- **become : true** :
  Cela nous permet de passer root, ce dont nous avons besoins pour copier le fichier service dans le bon répertoire et pour l'activer.
- **host** :
  C'est le nom du groupe dans le fichier inventaire qui contient la machine.
- **tags** :
  C'est ce qui va nous permettre si on en a besoin de lancer seulement ce rôle en spécifiant le tag dans la ligne de commande d'Ansible.

On répète le même schéma pour les autres rôles.
Si par exemple, certaines de ces valeurs ne changent jamais, il est possible de les définir dans le fichier **ansible.cfg** qui se trouve à la racine du projet et qui permet de contrôler plusieurs aspects du fonctionnement d'Ansible.

Execution du Playbook

La commande suivante permettra de déployer notre stack de monitoring :

```
1 Ansible-Playbook Playbook.yml -i inventory/host.yaml
```

Il est également possible de redéployer seulement un rôle en précisant le tag du rôle dans la commande ci-dessus.

Ce qui donne par exemple :

```
1 Ansible-Playbook Playbook.yml -i inventory/host.yaml --tags="NOM_DU_ROLE"
```

On peut complexifier la commande et utiliser plusieurs paramètres ensemble. Par exemple, pour lancer le playbook, sur un groupe de machines, un rôle précis :

```
1 Ansible-Playbook Playbook.yml -i inventory/host.yaml --tags="NOM_DU_ROLE" –limit "NOM_DU_GROUPE"
```

Lorsqu'une tache est exécutée, il y a plusieurs états possibles :

- **OK** :
  La tâche a été exécutée correctement mais aucuns changements n'a été réalisés C'est le cas lorsqu'on relance un Playbook ou des taches n'ont pas été modifiée.
- **CHANGED** :
  La tâche a été exécutée correctement et un changement a été appliqué.
- **FAILED** :
  la tâche n'a pas été exécutée correctement. Généralement cela signifie que le Playbook ne sera pas déroulé dans son intégralité sauf si nous gérons la gestion des erreurs en utilisant comme paramètre à une tâches **ignore_errors : yes** avec également **force_handlers : yes**. Si par exemple on demande un redémarrage d'un service avec le paramètre **notify** et **force_handlers : yes** , le Playbook continuera même si le démarrage du service échoue.
- **IGNORED** :
  C'est le résultat d'une tâche qui ne s'est pas déroulée correctement mais qui permet au Playbook de poursuivre son exécution.

- **UNREACHABLE** :
  C'est quand la machine cliente n'est pas joignable (machine éteinte, port ssh bloqué, …) La machine est alors marquée comme **injoignable** est Ansible la retire de la liste des machines actives pour le reste du Playbook.

Configuration de Grafana

Ajout des data sources dans Grafana

Une fois les agents Promtail et Telegraf configurés pour envoyer les données à Influxdb et Loki, il faut par la suite ajouter dans Grafana les **data sources**, c'est à dire Influxdb et Loki.

Cette action est réalisée dans les options de Grafana en lui indiquant le chemin d'accès pour Influxdb et Loki ainsi que les éléments d'identification nécessaires. En Annexe, vous trouverez la capture d'écran qui illustre le paramétrage des data sources.
Lien ici

Importation du Dashboard

Le Playbook contient également un **dashboard** (tableau de bord) que j'ai créé précédemment et qui peut être réutilisé pour chaque nouveau déploiement. Il suffit de le charger dans le menu à gauche et nous avons les graphiques correspondant à chaque requêtes d'Influxdb. Il contient des commandes génériques qui vont pouvoir récupérer les informations sur le CPU, le % de RAM de libre, le %de disque de libre, …

En Annexe, vous trouverez la capture d'écran qui illustre l'utilisation du dashboard.
Lien ici

Pour les logs, pour le moment, il n'y a pas de dashboard de créé. Il suffit d'aller dans **explorer** puis sélectionner Loki comme data source et nous pouvons trouver les logs que Promtail à récupérer.

Utilisation de Grafana

Grafana permet de créer des alertes en fonction de critères choisis par l'administrateur. On peut par exemple définir l'envoi d'un mail lorsqu'un seuil est franchi.

C'est très utile pour surveiller l'espace disque. L'administrateur va définir un seuil d'alerte (ex : 80% Plein) et quand il est atteint, un mail est envoyé.

Plutôt qu'un mail, il est possible de créer des alertes dans Teams, ou Slack en configurant des **webhooks**.

Exemple de configuration pour une alerte

Grafana inclut un server SMTP qu'il faut paramétrer dans son fichier principal de configuration. Afin de simplifier les changements de configuration et pour éviter

de devoir réécrire des rôles pour modifier le fichier de configuration, il est plus simple et pratique d'utiliser un Template pour modifier ce dernier.

Dans le rôle d'installation de Grafana, j'utilise une tâche qui crée le fichier de configuration selon ce que j'aurai défini dans le fichier de Template.

Voici la tâche que j'ai utilisé et qui va créer le fichier de configuration avec la bonne configuration :

```
- name: "create custom Grafana configuration file from Template"
  template:
    src: Grafana_conf.j2
    dest: "{{ Grafana_main_folder  }}/Grafana/conf/custom.ini"
    mode: 0755
    owner: "{{ Grafana_account_name  }}"
    group: "{{ Grafana_account_group  }}"
  notify:
    - restart Grafana
```

Voici une partie de la configuration du serveur SMTP dans le Template :

```
#################################### SMTP / Emailing ##########################
[smtp]
enabled = true  <- on active le serveur SMTP par défaut
host = localhost:25
from_address = "{{ Grafana_email  }}" <- une variable ici pour pouvoir changer le mail qui sera
    utiliser pour envoyer les notifications
from_name = Grafana-monitoring
```

Le serveur étant configuré, il ne reste plus qu'a mettre en place les alertes dans Grafana. Par exemple, j'ai défini les alertes suivantes pour la surveillance de mon cluster :

- Température des CPU
- Utilisation des CPU
- Utilisation de la Mémoire
- Surveillance des nodes du cluster

En Annexe, vous trouverez la capture d'écran qui illustre la configuration des alertes dans le dashboard.
Lien ici

Par exemple, pour surveiller l'utilisation de la mémoire, il suffit d'écrire une requête qui va déclencher l'envoie d'un mail si l'utilisation de la mémoire dépasse 85%.

Le WEBUI de Grafana facilite grandement la création d'alertes.

Voici comment cela se traduit :

```
Rule Name Memory Usage alert
Evaluate every 60s For 0m

Conditions
```

```
WHEN max () OF query (B, 5m, now) IS ABOVE 0,85
No Data & Error Handling

Alerting
Notifications Send to marc.cenon33@gmail.com
Message : alerte dépassement mémoire
```

En Annexe, vous trouverez la capture d'écran qui illustre cette configuration.

Un outil de monitoring n'est utile que s'il est bien configuré. Un AdminSys ne va pas passer son temps à regarder des graphs de monitoring.

En créant des alertes sur des points importants, on va recevoir une notification afin d'agir sur le problème et d'être plus efficace sur d'autres tâches de travail.

Cela peut également nous permettre d'anticiper certaines actions comme par exemple l'ajout d'un disque en LVM. En surveillant l'espace libre d'un SDD et en mettant une alerte, il est possible de planifier une action d'ajout d'espace disque plutôt que de devoir le faire en urgence au dernier moment.

Cet outil de monitoring nous permet d'avoir une grande visibilité sur l'infrastructure et sur les actions à entreprendre pour anticiper les problèmes.

Paramétrage supplémentaire

Rendre le service accessible depuis l'extérieur

Le dernier point important de ce projet à été de rendre Grafana accessible depuis l'extérieur afin d'avoir accès au monitoring même en dehors du réseau interne. Plusieurs éléments étaient à prendre en compte pour y arriver :

Configuration d'OVH

CGI utilise OVH pour la majeure partie de son infrastructure. J'ai dû configurer :
- Dans la zone DNS, la création d'une entrée A qui lie une IP publique au nom de domaine choisi pour accéder à Grafana.

Configuration dans vSphere

L'infrastructure tourne sous ESXI avec NSX et VSphere pour fournir une interface graphique et les API REST pour la création, la configuration et la surveillance des composants tels que les contrôleurs, commutateurs logiques, ….

Grâce à VSphere et NSX, on peut configurer depuis un navigateur Web les VM, Firewall, Règles NAT, LoadBalancing, Vlan, ….

C'est avec cet outils que j'ai créé la VM de Monitoring où est installé Grafana - Influxdb - Loki (ainsi que Telegraf et Promtail pour exposer les informations de la machine de monitoring dans Grafana)

Sans rentrer dans les détails car ce n'est pas le sujet de ce rapport de stage, voici les étapes principales pour la configuration du serveur de monitoring :

- Création de la VM sous CentOS 7 :
  - création de la VM avec suffisamment de CPU + RAM pour faire tourner les applications confortablement
  - configuration du LVM (Logical Volume Manager) avec un disque dur de 50 Go monté en /appli, formaté EXT4 où sont installé les applications
- Configuration du VLAN :
  - Création d'un commutateur logique
  - Création d'un profil de protocole réseau sur le bon datacenter
  - Définition de la plage d'IP pour le VLAN
  - Configuration du contrôleur pour accéder au nouveau VLAN
- Configuration des règles dans le firewall
  - Ajout d'un dispositif NSX Edge Services Gateway
  - Configuration de l'interface principale avec son adresse IP Principale
  - Affectation au bon VM Network
  - Configuration de la Passerelle
  - Rajout des Certificats WildCard dans la configuration du firewall NSX Edge
  - Création des règles d'entrée/sorties (IPTABLE)
  - Configuration des règles NAT Il faut également configurer les règles NAT (DNAT et SNAT) pour traduit l'IP privée + port/service -> IP public + port/service.

Une fois ces étapes terminées, nous pouvons accéder à Grafana sur la bonne url en HTTPS.

Evolution et améliorations

Dans ce schéma d'installation, les différentes briques sont installés et la configuration TLS est supporté par NSX Edge dans VSphere. Le Playbook dans son état actuel permet de déployer la stack de monitoring sans support TLS (car géré par NSX Edge). Il peut être intéressant d'installer un reverse proxy du type Apache ou Nginx afin de ne pas exposer trop de port et de gérer les certificats sur la machine.

Je choisi d'installer NGINX. Pour cela, nous pouvons modifier notre Playbook de 2 façons :

- Écrire un rôle qui va installer et configurer. Cela peut être une bonne solution lorsqu'on a une configuration atypique mais cela peut demander du temps pour l'écrire.
- Utiliser un des nombreux rôles disponibles dans Ansible Galaxy et simplement modifier les variables nécessaires pour la configuration des services.

Je fais le choix d'utiliser la deuxième option. Cela me permet de tirer bénéfice d'Ansible Galaxy sans avoir à réécrire un rôle en entier. Je vais le choix d'utiliser le rôle de **geerlinguy**. Dans un premier temps, je télécharge le rôle depuis Ansible Galaxy :

```
ansible-galaxy install geerlingguy.nginx
```

Dans mon Playbook, j'ai besoin d'appeler ce nouveau rôle :

```
- hosts: monit
  roles:
    - { role: geerlingguy.nginx }
```

De cette façon, le rôle sera exécuté sur la VM ou sont installés Graphana, Influxdb et Loki. Ce rôle dispose d'un fichier **/defaults/main.yml**. Il est très bien documenté et permet de comprendre rapidement les variables que nous devons utiliser.

Nous avons juste besoin de créer la configuration nécessaire pour Grafana et Influxdb.

Dans ce fichier, nous avons besoin de configurer les **Vhosts** et nous pouvons donner une liste d'argument dans la variable **nginx_vhosts**. Voici ce que nous pouvons lui ajouter par exemple.

```
nginx_vhosts:
  - listen: "443 ssl http2"
    server_name: "grafana.support-ent.fr
    root: "/appli/monitoring/grafana
    index: "index.php index.html index.htm"
    access_log: "/appli/monitoring/logs/access_log"
    error_log: "/appli/monitoring/logs/error_log"
    extra_parameters: |
      location ~ \.php$ {
          fastcgi_split_path_info ^(.+\.php)(/.+)$;
          fastcgi_pass unix:/var/run/php5-fpm.sock;
          fastcgi_index index.php;
          fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
          include fastcgi_params;
```

```
    }
    ssl_certificate     {{ my_certts_pem }};
    ssl_certificate_key {{ my_certs_key }};
    ssl_protocols       TLSv1.1 TLSv1.2;
    ssl_ciphers         HIGH:!aNULL:!MD5;
 - listen: "80 default_server"
   servername: "grafana.support-ent.fr"
   return 301 https://$server_name$request_uri;
```

Ici, je défini le Vhost en **80** avec redirection automatique ainsi que le Vhost en **443**. Les certificats sont référencés en variables. Il ne reste plus qu'à relancer le Playbook. Vu que la seule modification et l'ajout du rôle Nginx, les autres tâches apparaîtront en **OK** car aucun changement n'est réalisé.

Sur le rôle Nginx, les taches apparaîtront en **changed** car il y a eu un changement. Le rôle va également vérifier la configuration du Nginx, redémarrer le service grâce à un handler.

Une fois l'exécution terminée, nous pouvons accéder à Grafana en HTTPS sans avoir à passer par NSX Edge. Une autre évolution possible sera de gérer la montée de version automatiquement avec des tests unitaires. Cela peut être dangereux et causé des problèmes sur des infrastructures importantes. C'est pourquoi nous testons d'abord sur un environnement de pré-production avant de passer à la production. Il est important de faire les mises à jour afin de corriger les failles de sécurités quand des services sont exposés sur le web.

Il sera également intéressant de créer un Dashboard pour l'analyse des logs ainsi que la mise en place d'un système d'alerting. Comme Loki est développé par les même développeurs que Grafana, la mise en place d'un tel système est identique à celle décrite plus haut.

Une autre amélioration consisterai à utiliser **Ansible-Vault** afin de chiffrer les informations sensibles tels que les mots de passe, token et autres éléments d'identification.

Son fonctionnement est relativement simple. Il suffit de chiffrer les fichiers contenant des éléments sensible avec la commande :

```
ansible-vault encrypt FICHIER_A_CHIFFRER
```

Un dernier point intéressant serait d'utiliser le module pour la configuration d'Influxdb qui est disponible dans un pack de module complémentaire à télécharger. C'est une mise à jour que je vais entreprendre prochainement.

Conclusion sur ce projet

Nous avons ici un système de monitoring complet (métriques + logs système et applicatifs) avec des graphiques facilement compréhensibles et avec un système d'alerte en place. Ce qui est rassurant pour l'administrateur qui a définit ses seuils d'alertes afin de se laisser une marge de temps pour agir en conséquence.

En plus, comme Grafana est accessible depuis un simple navigateur internet, cela permet à l'Administrateur de pouvoir surveiller à distance l'infrastructure.

Cela a été pour moi un projet très enrichissant car j'ai pu construire sur des bases que j'avais en Ansible pour arriver à produire un script fonctionnel avec plusieurs briques logiciels. J'ai rencontré certaines difficultés dans la compréhension du fonctionnement de certain module d'Ansible mais en persévérant et avec l'aide de **Mr Thomas Coleno** et **Mr Arthur Bertinetti** j'ai pu réussir mes tâches.

Ansible est une technologie qui m'intéresse beaucoup et je suis très content d'avoir pu travailler dessus durant mon stage. J'ai par la suite créé d'autres script Ansible du type :

- installation / configuration d'un serveur Apache
- Configuration d'un pool de machine Big Blue Button
- Déploiement de machines à partir d'un template dans vSphere avec configuration du de base (ip, interface, hostname, lvm, …
- Déploiement d'une infrastructure complexe (Nginx, Apache, Drupal, Mariabd, Moodle, Python)

Sur cette dernière j'ai rencontré des difficultés sur certains points. Mon responsable a pu utiliser une partie du travail que j'ai fait pour arriver à un script qui fonctionne. Grâce à lui, j'ai appris de mes erreurs et pu grandement et efficacement améliorer mes compétences en Ansible notamment sur les notions de programmation en Python et sur la manipulation du format JSON. Ce sont ses notions qui m'ont manqué pour finir ce Playbook.

Conclusion

Ce stage correspondait parfaitement à ce que je recherché. Il m'a permis d'apprendre et de perfectionner certaines de mes connaissances, notamment tout ce qui touche à l'automatisation, au Scripting, et à la gestion de plusieurs VM. J'ai également pu faire un peu de programmation en Python.

Sur ce dernier point, j'ai encore beaucoup de travail à faire car je n'ai pas les connaissances suffisantes pour pouvoir travailler efficacement avec ce langage et je pense qu'il est important de savoir exploiter ce langage qui est un excellent langage de Scripting pour tout AdminSys.

Ce stage au sein d'une grande entreprise de service numérique de renommée mondial fut une expérience très enrichissante tant sur le plan personnel que professionnel. Cela m'a permis de conforter mon envie de travailler dans le secteur informatique en tant que DevOps. A 33 ans, en reconversion professionnelle, il faut être conscient de ses forces et faiblesse et je pense que j'ai fait le bon choix d'écouter ma passion pour en faire mon métier.

Le contexte actuel sanitaire a fait que j'étais en télétravail 99% du temps, ce qui ne rends pas forcement les choses faciles pour encadrer un stagiaire. **Mr Thomas Coleno** à parfaitement su me superviser et m'apporter l'aide nécessaire quand j'en avais besoin. Il m'a laissé une grande autonomie et m'a permis de progresser énormément.

En parallèle de ce stage, j'ai choisi de passer des certifications afin de valider mes compétences. J'ai pu obtenir les certifications suivantes :
- **Comptia Security +** : cette certification traite sur la cybersécurité.
- **CKA : Certified Kubernetes Administrator**. Une certification pour l'administration de clusters sous Kubernetes
- **RHCSA : Red Hat Certified System Administrator** : Administration système sur Red Hat / CentOS / Fedora

Je passe fin Septembre la certification **RHCE : Red Hat Certified Engeneer**.

Cette dernière certification est le prolongement logique de ce que j'ai fait durant mon stage. Elle est très pointue et elle est orientée sur l'automatisation et la très bonne maitrise d'Ansible pour administrer un S.I.

Pour terminer, j'ai eu une proposition d'embauche en CDI en tant que Cadre Ingénieur et j'ai accepté.

Je vais pouvoir évoluer au sein d'une équipe dynamique, sur des projets et des technologies intéressantes.

Annexes

image ENT

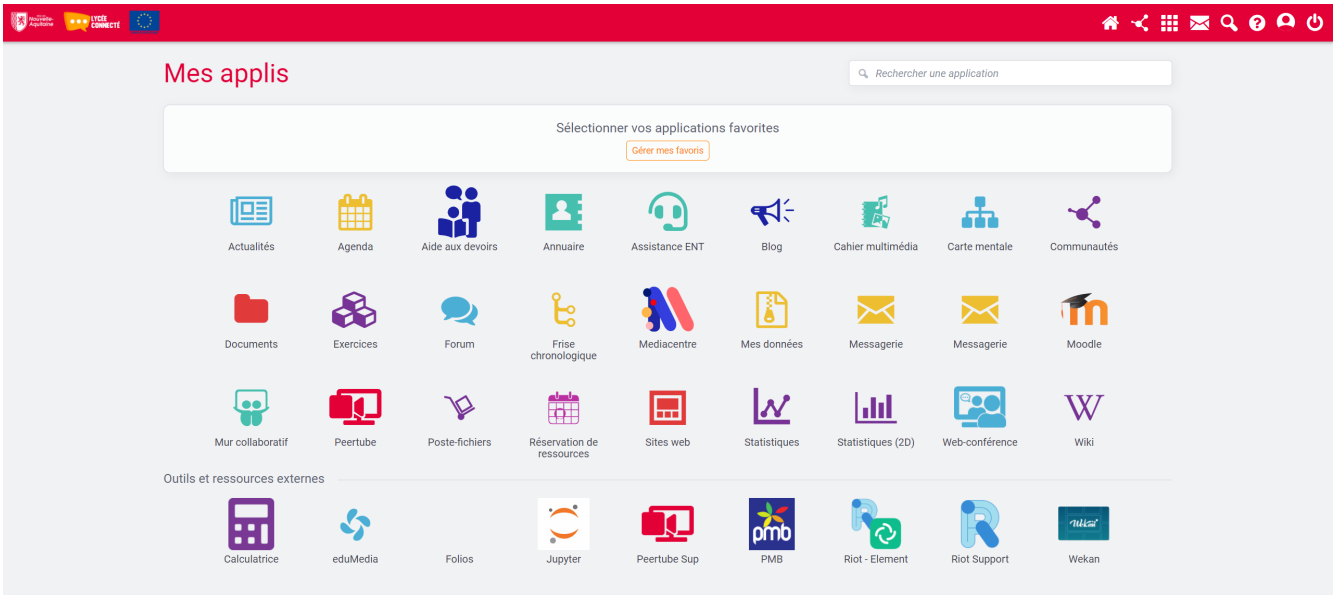

Figure 2 – ENT

influxdb bucket

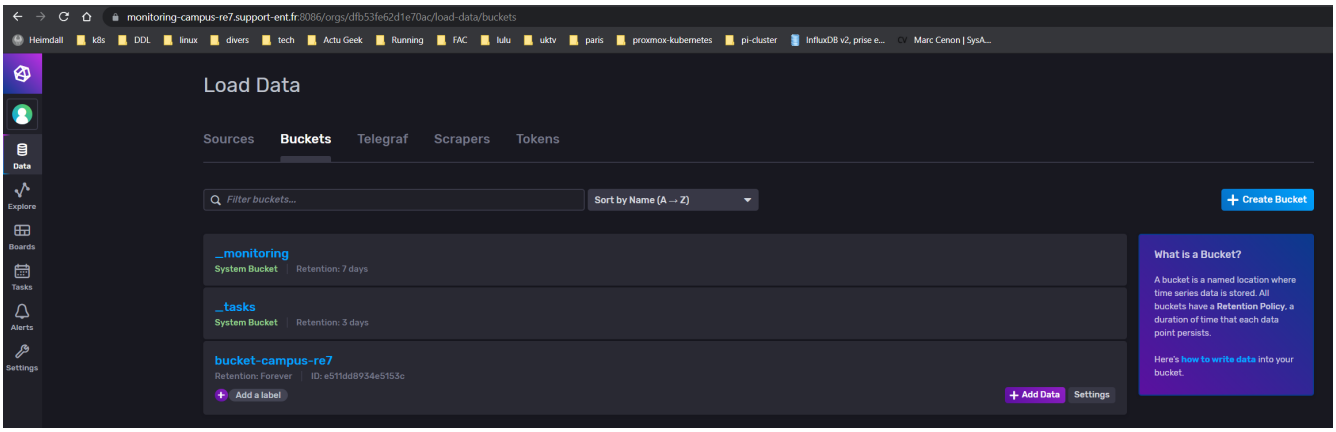

Figure 3 – bucket Influxdb

grafana dashboard

Figure 4 – Grafana dashboard

grafana data sources

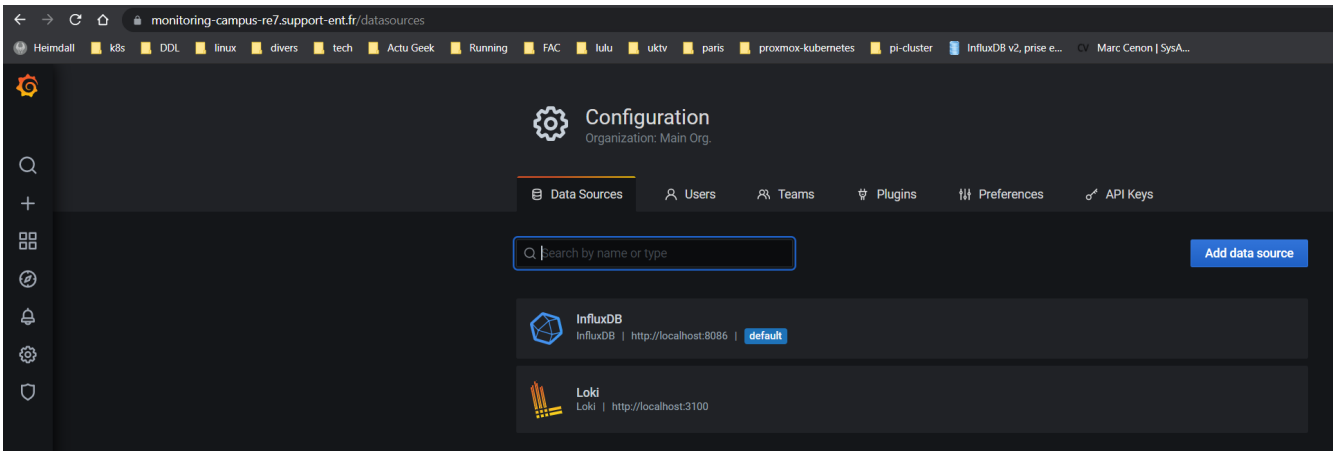

Figure 5 – Grafana datasources

requête influxdb

Figure 6 – Influxdb query

alerte grafana

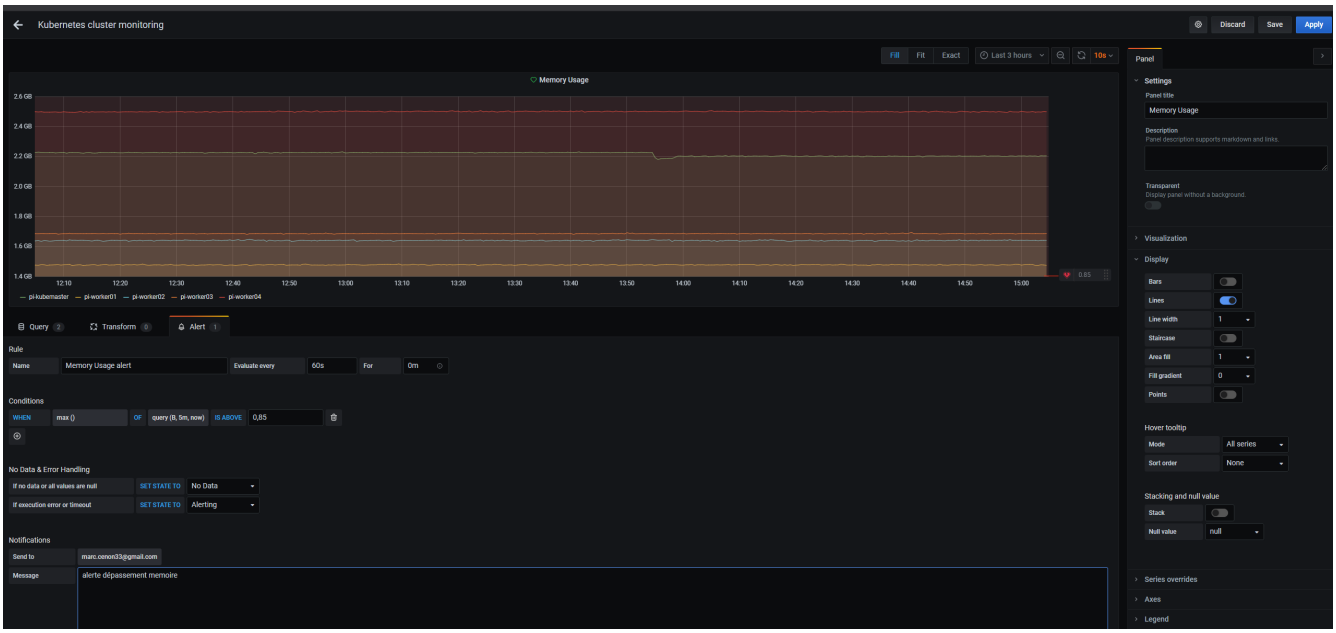

Figure 7 – Alerte

tableau du travail semaine par semaine

semaine	actions
Semaine 1	repérage, prise en main de l’infrastructure et des outils de travail
Semaine 2	creation de machines BBB et configuration (Openstack - BigBlueButton)
Semaine 3	creation de machines BBB et configuration (Openstack - BigBlueButton)
Semaine 4	scrit Ansible pour la modification de la configuration des BBB
Semaine 5	script Ansible Monitoring Grafana Influxdv Promtail Loki Telegraf
Semaine 6	script Ansible Monitoring Grafana Influxdv Promtail Loki Telegraf
Semaine 7	script Ansible Apache
Semaine 8	script Bash, Python et Ansible pour déploiement centre de formation
Semaine 9	script Python et Ansible pour centre de formation - suppression mail Zimbra
Semaine 10	script Ansible pour Nginx, Apache, Moodle, Drupal, Mariadb, Python
Semaine 11	script Ansible pour Nginx, Apache, Moodle, Drupal, Mariadb, Python
Semaine 12	script Ansible pour Nginx, Apache, Moodle, Drupal, Mariadb, Python
Semaibe 13	script Ansible pour Nginx, Apache, Moodle, Drupal, Mariadb, Python
Semaine 14	créationde VLANS dans VSPHERE et NSX Edge, firewall, et VM - installation de Jupyter hub
Semaine 15	renouvellement certificats sur vm et sur NSX Edge, creation de vm Moodle et Jupyter
Semaine 16	création d’instances moodle preprod et prod (Postgres, Apache, Moodle)
Semaine 17	Montée en version de Peertube - Mise à jour docker Riot - Jupyter Hub sous Kubernetes
Semaine 18	Création d’un proxy pour BigBlueButton, configuration php pour Moodle - configuration NSX Edge - troubleshooting Jupyter
Semaine 19	Montée en version de Moodle, Ansible pour mise à jours Zimbra, modification des specs des machines Zimbra dans Vsphere, Mise en Place de Jupyter Hub sous Kubernetes
Semaine 20	Mise à jours Zimbra, troubleshooting Moodle authentification CAS - Jupyter Hub sous Kubernetes

configuration data sources

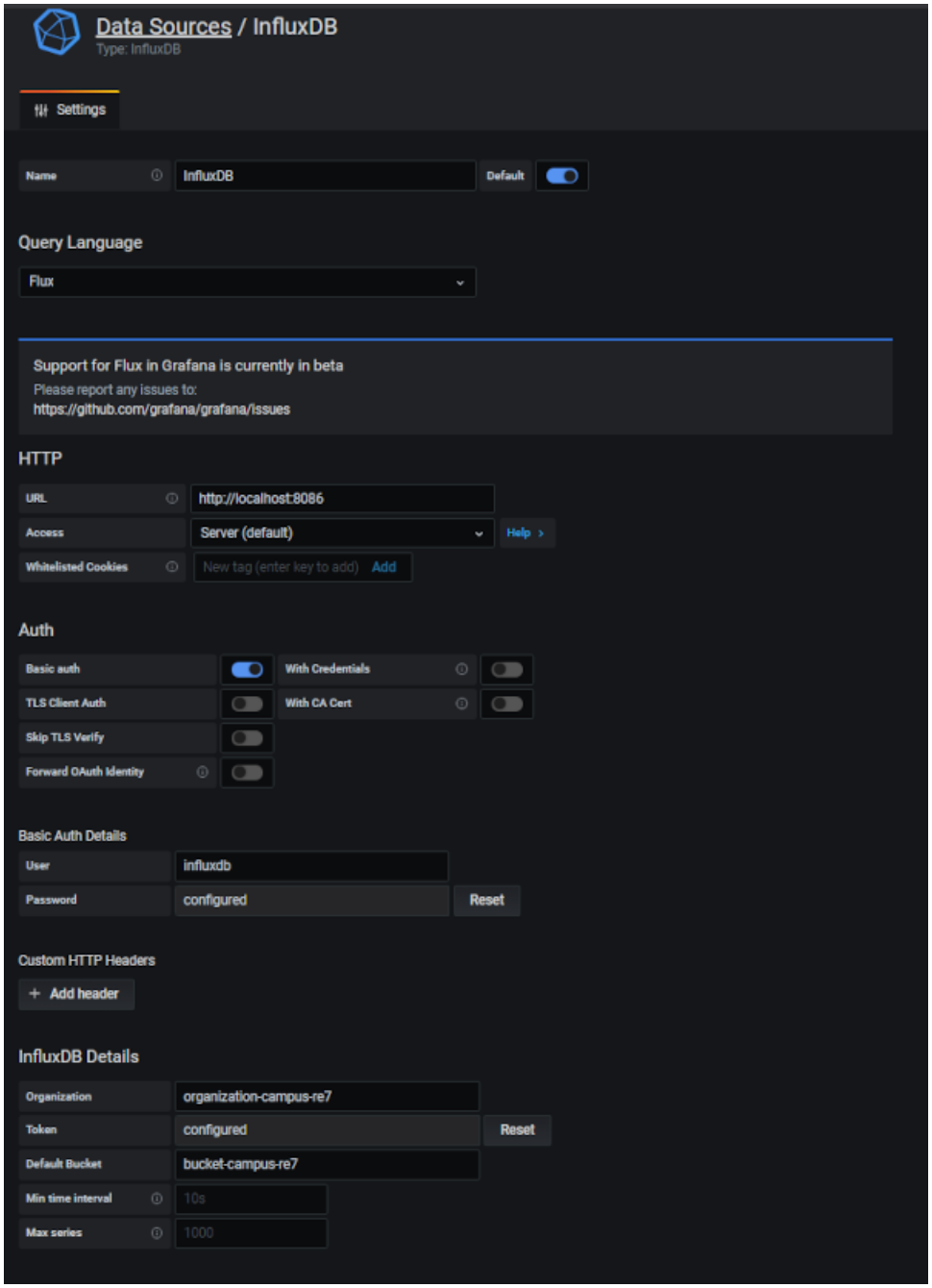

Figure 8 – Datasources configuration