



**l'école d'ingénierie  
informatique**

# Projet OPEN INNINATION

OUADGHIRI AHMED  
GBAMOU RAYMOND  
LOUIS STÉPHANE TCHANGA

## Table des matières

<b>1</b>	<b>Confidentialité</b>	<b>4</b>
<b>2</b>	<b>Téléchargement des sources</b>	<b>5</b>
<b>3</b>	<b>Ansible</b>	<b>5</b>
3.1	Préparation d'environnement	5
3.1.1	Installation d'Ansible	5
3.2	Création du projet SYMFONY	5
3.2.1	Déploiement avec Ansible	6
3.2.2	Installation du projet	10
3.3	Installation de docker avec Ansible	12
3.3.1	Lancement du playbook	13
<b>4</b>	<b>Gitlab</b>	<b>14</b>
4.1	Gitlab dockerisé	14
4.1.1	Les volumes	14
4.1.2	Accès au gitlab	15
4.2	Connexion	15
4.2.1	Gestion des mots de passe	15
4.2.2	Modification des mots de passes	15
4.2.3	Accès WEB	15
4.3	Menu utilisateur	16
4.4	Créer un projet	17
4.5	Permissions GitLab et protection des branches	19
4.6	Installation du Runner	26
4.6.1	Installation de GitLab Runner (Windows)	26
4.6.2	Installation de GitLab Runner (Linux)	33
4.6.3	Association d'un projet GitLab au Runner (Windows & Linux)	36
4.7	Gitlab-CI	38
4.7.1	Tests unitaires	38
4.7.2	Tests continus de vulnérabilités	47
4.7.3	Déploiement continu	50
4.8	Paramètres réseaux	51
4.8.1	DNS	51
4.8.2	SMTP	52
4.9	Intégration de l'annuaire Active Directory	52
4.9.1	Déclaration de l'annuaire	52

4.9.2 Nérification de votre config ldap	53
<b>5 Prometheus et grafana</b>	<b>53</b>
5.1 Prometheus et grafana dockerisé	53
5.1.1 Troubleshooting	54
5.1.2 Les volumes	54
5.2 Connexion	55
5.2.1 Mappage des ports	55
5.2.2 Gestion des mots de passe	55
5.2.3 Modification des mots de passes	55
5.2.4 Accès WEB	56
5.3 Configuration Prometheus	56
5.3.1 Les métriques du serveur Apache	60
5.3.2 Les métriques du serveur Nginx	62
5.4 Outil visualisation : Grafana	64
5.5 Création de panel docker	69
5.6 Métriques Hosts	73
5.6.1 Ajouter une source de données (Data sources)	73
5.6.2 Tableau de bord (Dashboard)	74
5.6.3 Monitorer une machine Windows	76
5.6.4 Monitorer une machine Linux	78
5.7 Intégration de l'annuaire Active Directory dans Grafana	80
Déclaration de l'annuaire	80

# **CONTEXTE**

---

Dans le cadre du projet innovation à L'epsi de Nantes mélangeant les filières et les niveaux d'étude, il nous a été demandé de répondre aux besoins de l'association Cité de l'Environnement spécialisée dans les domaines de l'environnement qui souhaite mettre en place une solution clé en main de chaîne DevSecOps pour le développement, la production et la maintenance de l'ensemble de l'exposition. La solution mise en place devra être sécurisée pour l'ensemble de la chaîne DevSecOps.

Dans ce contexte climatique actuel et parce que nous soutenons les projets écologiques, nous avons décidé de travailler en étroite collaboration avec l'association afin de leur proposer notre solution surnommée « Automatisator » répondant ainsi à leurs besoin et exigences.

## **2 LA SOLUTION AUTOMATISATOR**

---

### **2.1 Objectifs**

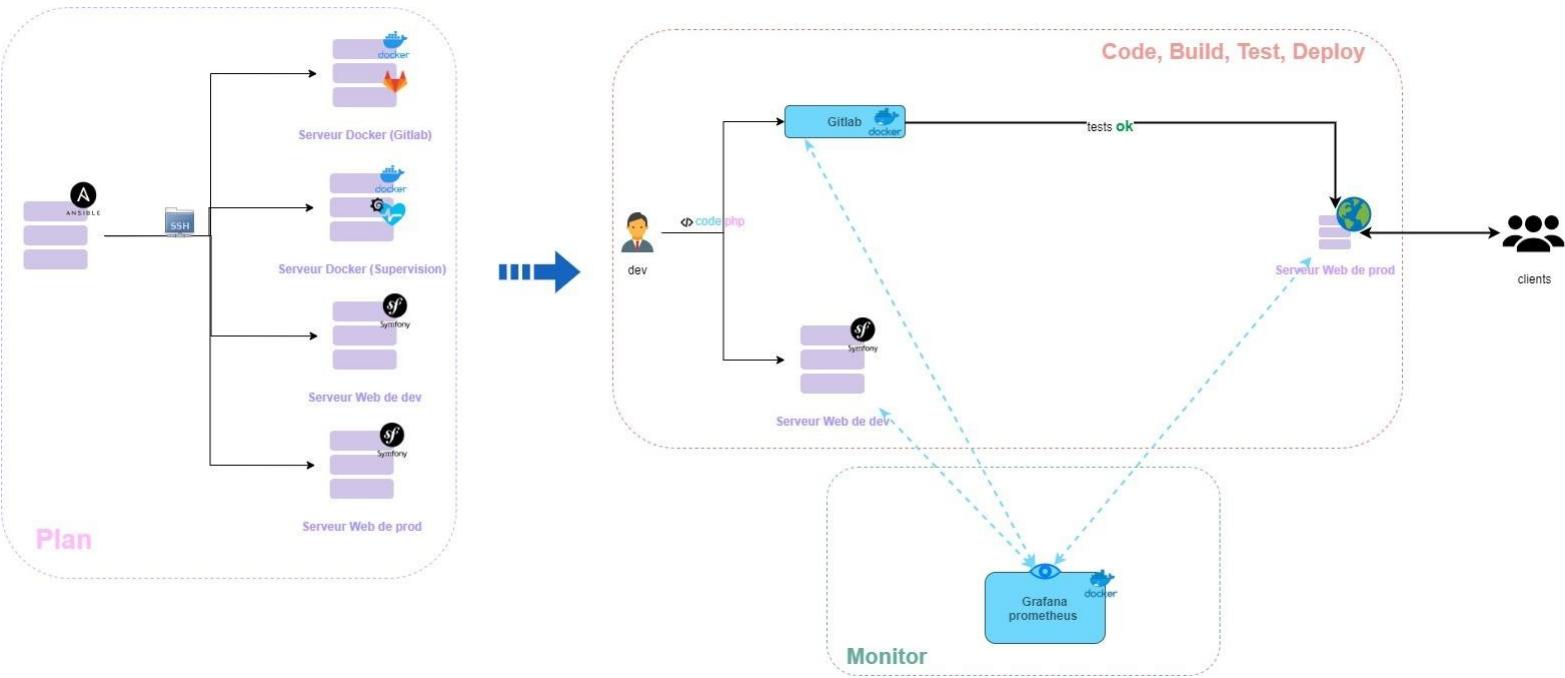
L'objectif des outils de notre solution est de rationaliser, raccourcir et automatiser davantage les différentes étapes du flux de travail de livraison de logiciels (ou « pipeline »). Ces outils promeuvent également les principes de base DevSecOps d'automatisation, de collaboration et d'intégration entre les équipes de développement, de sécurité et d'exploitation.

### **2.2 Les étapes**

Nous avons choisi de séparer notre solution sous forme de six étapes consécutives :



Ci-dessous les outils utilisés ainsi que leur fonctionnement à différentes étapes du cycle de vie DevSecOps :



### 2.2.1 Plan



Cette phase permet de définir les besoins de l'entreprise en termes d'infrastructure. Nous souhaitons par exemple installer une stack LAMP, ouvrir uniquement les ports 80/443, configurer votre application et utilisateurs utiliser, sur une multitude de serveurs sans vous y connecter manuellement ? C'est tout ici le but de cette étape, car sans même vous connectez et exécutez vos commandes sur votre parc de machines, vous exécuterez à la place uniquement un fichier yaml depuis une machine maître qui s'occupera d'automatiser la configuration de vos machines cibles.

Cette prouesse est possible depuis l'utilisation de l'outil Ansible qui simplifiera à l'équipe d'opérations les opérations répétitives, complexes et fastidieuses et cela permettra à votre équipe de gagner énormément de temps lorsque vous installez des packages ou configurez un grand nombre de serveurs.

Son architecture est simple et efficace. Il fonctionne en se connectant à vos nœuds et en y poussant de petits programmes. Ces programmes rendent le système conforme à l'état souhaité et, lorsqu'ils ont terminé leurs tâches, ils sont supprimés.

Cet outil fonctionne sur SSH et ne nécessite aucun démon, serveur spécial ou bibliothèque pour fonctionner. Un éditeur de texte et un outil en ligne de commande sont généralement suffisants pour faire votre travail. Nous décrivez simplement votre infrastructure dans un fichier YAML puis toutes les informations sur l'état souhaité de ces machines sont organisées en playbooks. Il est également capable de rassembler des informations sur les nœuds (telles que les adresses IP ou les détails du système d'exploitation) dans ce que l'on appelle des Facts, qui aident à l'approvisionnement sélectif et automatisé de différentes configurations sur la plate-forme. Tout cela avec un langage très lisible par l'homme (le YAML) sans avoir besoin d'écrire du code ou de déclarer des relations explicites.

## 2.2.2 Code, Build, Test, Deploy

Les étapes suivantes restent différentes les unes des autres mais nous avons choisi de les centraliser autour d'un seul outil appelé Gitlab.

### 2.2.2.1 Code



Cette phase implique la conception de logiciels et la création de code logiciel. Nos équipes ont besoin de stocker et de suivre les modifications apportées de leur code. Cela est possible depuis l'outil Git qui est le système de contrôle de version le plus utilisé au monde. Il suit les modifications que vous apportez aux fichiers, vous avez donc un enregistrement de ce qui a été fait et vous pouvez revenir à des versions spécifiques si vous en avez besoin. Git facilite également la collaboration, en permettant aux modifications de plusieurs personnes d'être fusionnées en une seule source.

Cependant, git ne peut s'exécuter qu'en local, vous avez donc besoin d'un outil qui concentre les fonctionnalités de git dans un même outil distant (on-premise ou cloud). Pour ce faire le choix s'est porté vers GitLab, une plate-forme entièrement intégrée basée sur Git pour le développement de logiciels. Outre les fonctionnalités de Git, GitLab possède de nombreuses fonctionnalités puissantes pour améliorer votre flux de travail que nous utiliserons dans les prochaines étapes du cycle DevSecOps.

### 2.2.2.2 Build et test



Au cours de cette phase de Build, vous gérez les versions et versions logicielles sur différentes branches (une branche test pour les équipes de développement et une branche pour les équipes de production) et utilisez l'intégration continue pour aider à compiler et à empaqueter automatiquement le code pour une version future en production.

Passe ensuite la phase de Test qui implique des tests automatisés continus pour assurer une qualité de code optimale. Nous avons les tests unitaires permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme (appelée « unité » ou « module ») et les tests de vulnérabilités automatisés afin de vérifier la sécurité de votre code. Le but de cette étape est de minimiser ainsi le risque de bugs et de vulnérabilités.

Cette automatisation est possible depuis la fonctionnalité d'intégration continue (CI) qui est intégré nativement dans Gitlab. C'est une pratique de fusionner tout le code produit par les développeurs. La fusion a généralement lieu plusieurs fois par jour dans un référentiel partagé. À partir du référentiel, des tests de construction et automatisés sont effectués pour garantir l'absence de problèmes d'intégration et l'identification précoce de tout problème avant son envoi en production.

### 2.2.2.3 Deploy



Une fois les tests d'intégration continuent réussis, la prochaine étape est la phase de déploiement qui inclut des outils qui aident à gérer, coordonner, planifier et automatiser les versions de produit en production. Cet exploit est possible grâce à la pratique de la livraison continue (CD) qui est inclus également nativement dans Gitlab et garantit la livraison du code validé CI à votre application au moyen d'un pipeline de déploiement structuré.

CI et CD agissent ensemble pour accélérer la rapidité avec laquelle votre équipe peut fournir des résultats à vos clients et parties prenantes. CI vous aide à détecter et à réduire les bogues au début du cycle de développement, et le CD déplace plus rapidement le code vérifié vers vos applications. CI et CD doivent fonctionner de manière transparente afin que votre équipe puisse se constituer rapidement et

Efficacement, tout en étant essentielle pour garantir une pratique de développement entièrement optimisée.

#### 2.2.2.4 Monitor



Cette phase sera gérée par les outils open source Prometheus et Grafana. Cette étape consiste à identifier et à collecter des informations sur les problèmes d'une version spécifique du logiciel en production et de surveiller les métriques et les logs de vos différentes applications afin de découvrir l'impact sur l'expérience de l'utilisateur.

## Planing prévisionnel

## **Intitulé du projet : Mise en place de la solution Automatisator**

Date : Jeudi 10 MARS 2022

# FACTURE No. 12365

16 Bd Général de Gaulle, 44200 Nantes  
France

Cité de l'environnement

Référence : 598

Date : 10/02/2022

N°client : 1234

Intitulé : Description du projet et/ou Produits facturés

Quantité	Désignation	Prix HT	Prix total HT
3	- <i>NPS – ONH (8 Go RAM, 80 Go SSD, 2vCores)</i>	11,99 €	35,97 €
2	- <i>NPS – ONH (4 Go RAM, 40 Go SSD, 1vCores)</i>	5,99 €	11,98 €
1	<i>- Provisionnement et configuration des machines virtuelles – 2 jours</i> <i>- Rédaction Livrables - 1 jour</i>	1250,00 € 425,00 €	1675,00 €
1	<i>- Installation et configuration Ansible – 5 jours</i> <i>- Rédaction Livrables - 1 jour</i> <i>- Licence Ansible</i>	2875,00 € 425,00 € Gratuite	3300,00 €
1	<i>- Mise en place et configuration de Gitlab + testsunitaires – 10 jours</i> <i>- Intégration des tests continus de vulnérabilités – 5 jours</i> <i>- Rédaction Livrables - 2 jours</i> <i>- Licence Gitlab</i>	6100,00 € 3025,00 € 850,00 € Gratuite	9975,00 €
1	<i>- Déploiement et configuration de la solution de supervision – 6 jours</i> <i>- Rédaction Livrables - 1 jour</i> <i>- Licence Grafana et Prometheus</i>	3750,00 € 425,00 € Gratuite	4175,00 €

<i>1</i>	<i>- Formation technicien - 3 jours</i>	<i>1575,00 €</i>	<i>1575,00 €</i>
----------	---	------------------	------------------

Total Hors Taxe	<b>20747,95 €</b>
TNA à 20%	<b>4 149,59 €</b>
<b>Total TTC en euros</b>	<b>24 897,54 €</b>

En votre aimable règlement,  
Cordialement,

Conditions de paiement : paiement à réception de facture, à 30 jours...Aucun escompte consenti pour règlement anticipé

Tout incident de paiement est possible d'intérêt de retard. Le montant des pénalités résulte de l'application aux sommes restant dues d'un taux d'intérêt légal en vigueur au moment de l'incident.

Indemnité forfaitaire pour frais de recouvrement due au créancier en cas de retard de paiement : 40€

Révision			
Nersion	Date	Auteur	Commentaires
1.0	10/02/2020	Equipe Automatisator	Création du document

## 1 CONFIDENTIALITE

AUTOMATISATOR et les membres de son personnel s'engagent à respecter les contraintes de sécurité et de confidentialité de l'association dans le cadre de ce projet.

AUTOMATISATOR et les intervenants s'engagent à considérer comme strictement confidentiel tant au sein de sa propre organisation que vis à vis de tiers, les informations, documents de toute nature et savoir-faire, qui lui seront transmis à l'association dans le cadre du projet.

Cet engagement vaut quel que soit le support utilisé pour cette transmission ou simplement les informations que AUTOMATISATOR aura pu obtenir ou eu connaissance au titre de la prestation. A cet effet, AUTOMATISATOR ne communiquera ces informations qu'aux personnes affectées à l'exécution de la prestation.

AUTOMATISATOR s'engage à ne pas utiliser les informations directement ou indirectement en tout ou partie, à quelque fin que ce soit en dehors de l'exécution de la prestation décrite dans le présent document.

## 2 ANSIBLE

### 2.1 Préparation d'environnement

Dans un premier temps nous mettons à jour le système :

```
apt-get update  
apt-get upgrade  
apt-get dist-upgrade
```

#### 3.1.1 Installation d'Ansible

Pour l'installation d'ansible, nous suivons la procédure de site officiel d'ansible ([https://docs.ansible.com/ansible/latest/installation\\_guide/intro\\_installation.html](https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html)). Nous commençons pour ajouter la ligne suivante à `/etc/apt/sources.list`.

```
deb http://ppa.launchpad.net/ansible/ansible/ubuntu trusty main
```

```
# stretch-updates, previously known as 'volatile'  
deb http://ftp.fr.debian.org/debian/ stretch-updates main  
deb-src http://ftp.fr.debian.org/debian/ stretch-updates main  
  
deb http://ppa.launchpad.net/ansible/ansible/ubuntu trusty main
```

Après, nous exécutons les commandes suivantes :

```
apt-get install dirmngr  
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 93C4A3FD7BB9C367  
apt-get update  
apt-get install ansible  
ansible -version
```

```
root@debian:/etc/apt# ansible --version  
ansible 2.9.6  
  config file = /etc/ansible/ansible.cfg  
  configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']  
  ansible python module location = /usr/lib/python2.7/dist-packages/ansible  
  executable location = /usr/bin/ansible  
  python version = 2.7.13 (default, Sep 26 2018, 18:42:22) [GCC 6.3.0 20170516]
```

## 2.2 Création du projet SYMFONY

Pour la création du projet Symfony, nous devons suivre les étapes suivantes :

Si ce n'est pas fait générer votre paire de clé sur le serveur maître/cible

```
ssh-keygen
```

Sur le serveur sur lequel, on autoriser Ansible à intervenir :

```
ssh-copy-id -i ~/.ssh/id_rsa.pub root@IP_SERVEUR_WEB
```

Maintenant que cela est fait, nous allons insérer notre serveur (Debian) dans le fichier `/etc/ansible/hosts` comme ceci :

```
[web]
192.168.1.27
```

Finalement pour tester qu'il fonctionne, nous lançons la commande suivante :

```
# ansible all -m ping -u root
```

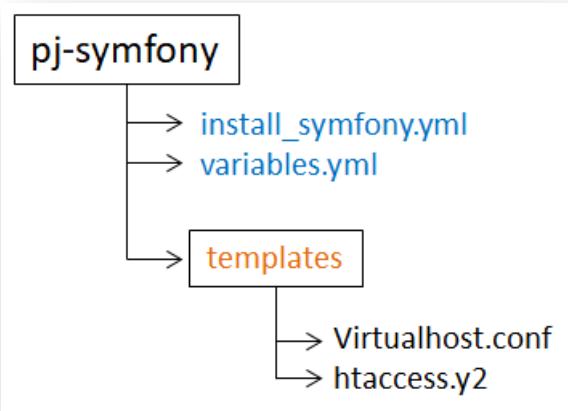
```
root@debian:~# ansible all -m ping -u root

[WARNING]: Platform linux on host 192.168.1.27 is using the discovered Python
future installation of another Python interpreter could change this. See
https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

### 3.2.1 Déploiement avec Ansible

Pour la structure du projet, nous allons créer un dossier appelé « **pj-symfony** » qui contiendra notre playbook et les templates nécessaires pour la réussite de l'installation.

```
mkdir /etc/ansible/pj-symfony
```



Nous retrouverez le playbook `install_symfony.yml` dans les sources livrées. Ce fichier est le cœur de notre projet, car il contiendra toutes les tâches nécessaires pour l'installation de Symfony. Il appellera aussi les templates et les variables créées dans le fichier `variables.yml`.

```

---
- name: create_projet_symfony
  hosts: web
  remote_user: root
  become: true
  vars_files: variables.yml
  tasks:

```

Dans les premières lignes du fichier `install_symfony.yml` on spécifie le groupe d'hôtes que nous souhaitons utiliser (web), l'utilisateur (root) et on s'assurent qu'il exécute les commandes par défaut. Nous appelons également le fichier `variables.yml` qui contiendra nos variables :

- **nomprojet** : le nom de l'application web
- **dossier** : le répertoire d'installation du projet
- **domainsite** : le nom de site à monter

Pour la première tâche « `installer_dependences` », nous allons installer toutes les dépendances nécessaires pour la bonne installation de « Composer ».

```

- name: installer_dependences
  apt: name={{ item }} state=latest
  with_items:
    - git
    - curl
    - unzip
    - php
    - php-fpm
    - php-mysql
    - php-curl
    - php-cli
    - php-bcmath
    - php-mbstring
    - php-xml
    - apache2
    - libapache2-mod-php

```

La **deuxième tâche « installer\_composer »** nous installerons « Composer », un gestionnaire de packages en PHP que facilita l'installation de Symfony et ses librairies dépendances.

```

- name: installer_composer
  shell: curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer
  args:
    creates: /usr/local/bin/composer

```

La **troisième tâche « create\_dossier/\_var/www\_si\_nexiste\_pas »**, nous créons le dossier www, dans le cas qu'il n'existe pas dans le système Debian.

```

- name: create_dossier/_var/www_si_nexiste_pas
  file: dest="{{ dossier }}" state=directory owner=www-data group=www-data mode=0700

```

Les **4me, 5me et 6me tâches**, nous créons le projet symfony :

```

- stat: path="{{ dossier }}/{{ nomprojet }}"/composer.lock
  register: composerlock

- name: create_pj_symfony
  composer:
    command: create-project
    arguments: symfony/website-skeleton "{{ nomprojet }}"
    working_dir: "{{ dossier }}"
  become: true
  become_user: www-data
  when: composerlock.stat.exists == False

- name: update_dependences_pj_symfony
  composer: command=update working_dir="{{ dossier }}/{{ nomprojet }}"
  become: true
  become_user: www-data
  when: composerlock.stat.exists == True

```

A partir de la **7me tâche « <<virtualhost >>**, nous allons créer des procédures pour la montée du projet dans le serveur web Apache. Nous créons un fichier de configuration Apache dans le répertoire `/etc/apache2/sites-available/` et nous ajoutons un Nirtual Host pour l'application Symfony. Pour le

virtualhost, nous utilisons un template appelé « `virtualhost.conf` ». Nous créons aussi un fichier appelé « `.htaccess` » dans le dossier du projet, car il nous aidera la montée dans l'apache. De la même façon, nous activons le site web avec `a2ensite` et finalement nous redémarrons le serveur Apache.

```
- name: virtualhost
  template: src=templates/virtualhost.conf dest=/etc/apache2/sites-available/{{ domainsite }}.conf

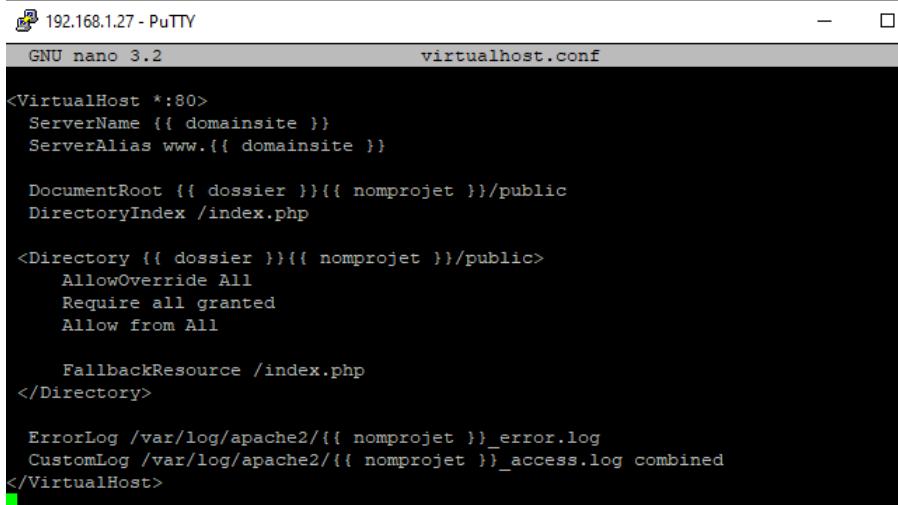
- name: htaccess_pour_pj_symfony
  template: src=templates/htaccess-dos.j2 dest={{ dossier }}{{ nomprojet }}.htaccess

- name: a2ensite
  command: a2ensite {{ domainsite }}
  args:
    creates: /etc/apache2/sites-enabled/{{ domainsite }}.conf

- name: restart_apache2
  apache2_module: name=rewrite state=present
  notify:
    - restart apache2
```

Nous trouverez également les fichiers de configuration apache qui seront utilisés dans les tâches du playbook dans le dossier templates.

- Le template : `virtualhost.conf`



The screenshot shows a PuTTY terminal window titled "192.168.1.27 - PuTTY". The title bar also displays "GNU nano 3.2" and "virtualhost.conf". The main pane of the terminal contains the following Apache configuration code:

```
<VirtualHost *:80>
  ServerName {{ domainsite }}
  ServerAlias www.{{ domainsite }}

  DocumentRoot {{ dossier }}{{ nomprojet }}/public
  DirectoryIndex /index.php

  <Directory {{ dossier }}{{ nomprojet }}/public>
    AllowOverride All
    Require all granted
    Allow from All

    FallbackResource /index.php
  </Directory>

  ErrorLog /var/log/apache2/{{ nomprojet }}_error.log
  CustomLog /var/log/apache2/{{ nomprojet }}_access.log combined
</VirtualHost>
```

- Le template : htaccess.j2

```
192.168.1.27 - PuTTY
GNU nano 3.2

# Use the front controller as index file. It serves as a fallback solution when
# every other rewrite/redirect fails (e.g. in an aliased environment without
# mod_rewrite). Additionally, this reduces the matching process for the
# start page (path "/") because otherwise Apache will apply the rewriting rules
# to each configured DirectoryIndex file (e.g. index.php, index.html, index.pl).
DirectoryIndex index.php

# By default, Apache does not evaluate symbolic links if you did not enable this
# feature in your server configuration. Uncomment the following line if you
# install assets as symlinks or if you experience problems related to symlinks
# when compiling LESS/Sass/CoffeeScript assets.
# Options FollowSymlinks

# Disabling MultiViews prevents unwanted negotiation, e.g. "/index" should not resolve
# to the front controller "/index.php" but be rewritten to "/index.php/index".
<IfModule mod_negotiation.c>
    Options -MultiViews
</IfModule>

<IfModule mod_rewrite.c>
    RewriteEngine On

    # Determine the RewriteBase automatically and set it as environment variable.
    # If you are using Apache aliases to do mass virtual hosting or installed the
    # project in a subdirectory, the base path will be prepended to allow proper
    # resolution of the index.php file and to redirect to the correct URI. It will
    # work in environments without path prefix as well, providing a safe, one-size
    # fits all solution. But as you do not need it in this case, you can comment
    # the following 2 lines to eliminate the overhead.
    RewriteCond %{REQUEST_URI}::$1 ^(/.+)/(.*)::\2$
    RewriteRule ^(.*) - [E=BASE:$1]

    # Sets the HTTP_AUTHORIZATION header removed by Apache
    RewriteCond %{HTTP:Authorization} .
    RewriteRule ^ - [E=HTTP_AUTHORIZATION:{HTTP:Authorization}]

    # Redirect to URI without front controller to prevent duplicate content
    # (with and without '/index.php'). Only do this redirect on the initial
    # rewrite by Apache and not on subsequent cycles. Otherwise we would get an
    # endless redirect loop (request -> rewrite to front controller ->
    # redirect -> request -> ...).
    # So in case you get a "too many redirects" error or you always get redirected
    # to the start page because your Apache does not expose the REDIRECT_STATUS
    # environment variable, you have 2 choices:
    # - disable this feature by commenting the following 2 lines or
    # - use Apache >= 2.3.9 and replace all L flags by END flags and remove the
    #   following RewriteCond (best solution)
    RewriteCond %{ENV:REDIRECT_STATUS} ^
    RewriteRule ^index\.php(?:/(.*))\$1 %{ENV:BASE}/\$1 [R=301,L]

    # If the requested filename exists, simply serve it.
    # We only want to let Apache serve files and not directories.
    RewriteCond %{REQUEST_FILENAME} -f
    RewriteRule ^ - [L]

    # Rewrite all other queries to the front controller.
    RewriteRule ^ %{ENV:BASE}/index.php [L]
</IfModule>
```

### 3.2.2 Installation du projet

A la fin quand nous avons fini de déployer tous les fichiers mentionnés, nous lançons le playbook avec la commande suivante :

```
ansible-playbook install_symfony.yml
```

```

root@debian:/etc/ansible/pj-symfony# ansible-playbook install_symfony.yml
PLAY [create_projet_symfony] ****
TASK [Gathering Facts] ****
[WARNING]: Platform linux on host 192.168.1.27 is using the discovered Python interpreter at /usr/bin/python, but future installation of another Python interpreter could change this. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more information.
ok: [192.168.1.27]

TASK [installer dependences] ****
[DEPRECATION WARNING]: Invoking "apt" only once while using a loop via squash_actions is deprecated. Instead of using a loop to supply multiple items and specifying "name: \"{{ item }}\"", please use 'php-fpm', 'php-mysql', 'php-curl', 'php-cli', 'php-bcmath', 'php-mbstring', 'php-xml', 'apache2', 'libapache2-mod-php')' and remove the loop. This feature will be removed in version 2.11. Deprecation setting deprecation_warnings=False in ansible.cfg.
ok: [192.168.1.27] => (item=u'git', u'curl', u'unzip', u'php', u'php-fpm', u'php-mysql', u'php-curl', u'php-cli', u'php-bcmath', u'php-mbstring', u'php-xml', u'apache2', u'libapache2-mod-php')

TASK [installer composer] ****
ok: [192.168.1.27]

TASK [create_dossier_/_var/www/_si_nexiste_pas] ****
ok: [192.168.1.27]

TASK [stat] ****
ok: [192.168.1.27]

TASK [create_pj_symfony] ****
changed: [192.168.1.27]

TASK [update_dependencies_pj_symfony] ****
skipping: [192.168.1.27]

TASK [virtualhost] ****
changed: [192.168.1.27]

TASK [htaccess_pour_pj_symfony] ****
changed: [192.168.1.27]

TASK [a2ensite] ****
changed: [192.168.1.27]

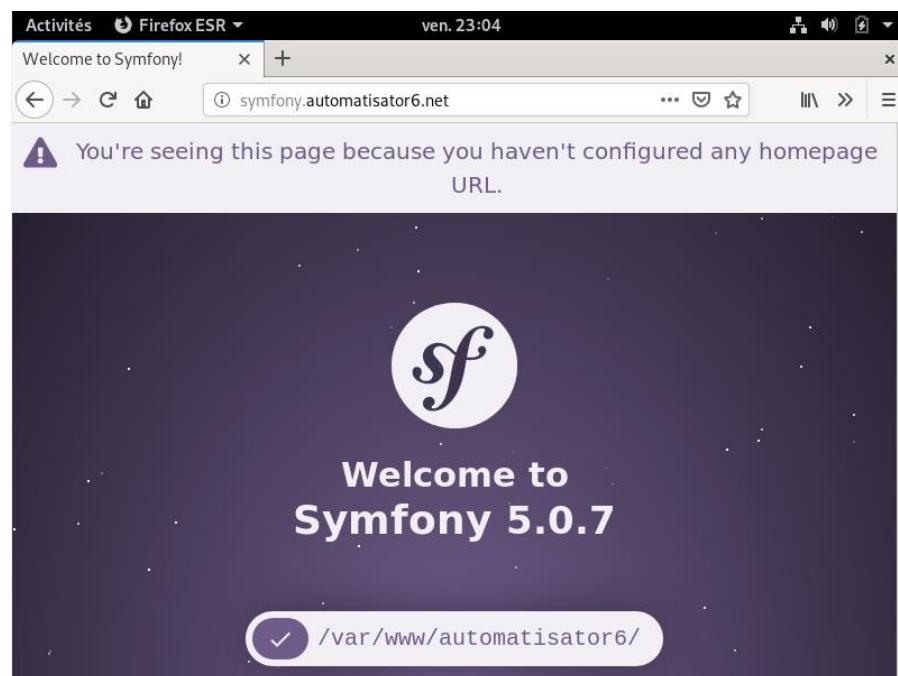
TASK [restart_apache2] ****
ok: [192.168.1.27]

PLAY RECAP ****
192.168.1.27      : ok=10    changed=4    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0

root@debian:/etc/ansible/pj-symfony# 

```

Finalement, nous vérifions la bonne installation du projet depuis notre inventaire en indiquant l'ip ou le nom dsn de notre machine cible:



## 2.3 Installation de docker avec Ansible

Selon le site officiel de Docker, nous devons préparer l'environnement approprié avant de procéder à l'installation. Le playbook fournit dans les sources prend en compte les procédures à faire avant l'installation.

Ouvrez le playbook `install_docker.yml`

Dans la première tâche, nous vérifions et nous supprimons les possibles anciennes versions de Docker dans le système cible.

```
--  
- name: installer_docker  
  hosts: web  
  remote_user: root  
  become: true  
  
  tasks:  
  
    - name: supprimer_possible_version_ancienne  
      apt:  
        name: "{{ item }}"  
        state: absent  
      with_items:  
        - docker  
        - docker-engine  
        - docker.io  
      become: true
```

Après dans la 2me tâche, nous installerons toutes les dépendances nécessaires pour la réussite de Docker.

```
- name: installer_dependences  
  apt: name={{ item }} state=latest  
  with_items:  
    - curl  
    - gnupg2  
    - software-properties-common  
    - ca-certificates  
    - apt-transport-https  
    - python-pip  
  become: true
```

Dans la 3me et 4me tâche, nous ajoutons la clé GPG officielle des dockers pour vérifier la signature des packages avant de les installer le gestionnaire de paquet de la machine cible. Ensuite nous les ajoutons les sources apt :

```
- name: importer_signature_au_systeme  
  shell: curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -  
  
- name: ajouter_signature  
  apt_repository:  
    repo: deb [arch=amd64] https://download.docker.com/linux/debian stretch stable  
    state: present
```

La 5me et 6me tache, nous mettons à jour la base de données des paquets et finalement nous installons Docker.

```
- name: Indexer_update
become: yes
apt:
  name: "*"
  state: latest
  update_cache: yes
  force_apt_get: yes

- name: install_docker
apt:
  name: "docker-ce"
  state: latest
```

### 3.3.1 Lancement du playbook

Pour lancer le **playbook**, nous exécutons la ligne suivante

```
ansible playbook install_docker.yml
```

```
root@debian:/etc/ansible/docker# ansible-playbook install_docker.yml
PLAY [installer_docker] ****
TASK [Gathering Facts] ****
[WARNING]: Platform linux on host 192.168.1.27 is using the discovered Python interpreter at
/usr/bin/python, but future installation of another Python interpreter could change this. See
https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more
information.
ok: [192.168.1.27]

TASK [remove possible old versions] ****
[DEPRECATION WARNING]: Invoking "apt" only once while using a loop via squash_actions is
deprecated. Instead of using a loop to supply multiple items and specifying `name: "{{ item }}``,
please use `name: ['docker', 'docker-engine', 'docker.io']` and remove the loop. This feature will
be removed in version 2.11. Deprecation warnings can be disabled by setting
deprecation_warnings=False in ansible.cfg.
ok: [192.168.1.27] => (item=['docker', 'u'docker-engine', 'u'docker.io'])

TASK [installer_dependencies] ****
[DEPRECATION WARNING]: Invoking "apt" only once while using a loop via squash_actions is
deprecated. Instead of using a loop to supply multiple items and specifying `name: "{{ item }}``,
please use `name: ['curl', 'gnupg2', 'software-properties-common', 'ca-certificates', 'apt-
transport-https', 'python-pip']` and remove the loop. This feature will be removed in version
2.11. Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
changed: [192.168.1.27] => (item=[u'curl', u'gnupg2', u'software-properties-common', u'ca-certificates', u'apt-transport-https', u'python-pip'])

TASK [importer_signature_au_systeme] ****
[WARNING]: Consider using the get_url or uri module rather than running 'curl'. If you need to
use command because get_url or uri is insufficient you can add 'warn: false' to this command task
or set 'command_warnings=False' in ansible.cfg to get rid of this message.
changed: [192.168.1.27]

TASK [ajouter_signature] ****
ok: [192.168.1.27]

TASK [Index new repo into the cache] ****
ok: [192.168.1.27]

TASK [actually install docker] ****
changed: [192.168.1.27]

PLAY RECAP ****
192.168.1.27 : ok=7    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Finalement, pour vérifier l'installation de Docker, nous exécutons la commande suivante sur le service cible

```
sudo systemctl status docker
```

```

root@debian:/etc/ansible/docker# sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2020-04-06 20:45:27 CEST; 56s ago
       Docs: https://docs.docker.com
Main PID: 21497 (dockerd)
   Tasks: 8
  Memory: 50.8M
    CGroup: /system.slice/docker.service
            └─21497 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

avril 06 20:45:22 debian dockerd[21497]: time="2020-04-06T20:45:22.50209674+02:00" level=warning msg="Your kernel does not support cgroup2 API"
avril 06 20:45:22 debian dockerd[21497]: time="2020-04-06T20:45:22.503488227+02:00" level=warning msg="Your kernel does not support cpuset cgroup feature"
avril 06 20:45:22 debian dockerd[21497]: time="2020-04-06T20:45:22.504077133+02:00" level=warning msg="Your kernel does not support memory limit in cgroups"
avril 06 20:45:22 debian dockerd[21497]: time="2020-04-06T20:45:22.504998422+02:00" level=info msg="Loading containers: start."
avril 06 20:45:23 debian dockerd[21497]: time="2020-04-06T20:45:23.398217658+02:00" level=info msg="Default bridge (docker0) is active"
avril 06 20:45:24 debian dockerd[21497]: time="2020-04-06T20:45:24.924623116+02:00" level=info msg="Loading containers: done."
avril 06 20:45:27 debian dockerd[21497]: time="2020-04-06T20:45:27.365207858+02:00" level=info msg="Docker daemon" commit=aafacb8
avril 06 20:45:27 debian dockerd[21497]: time="2020-04-06T20:45:27.366165507+02:00" level=info msg="Daemon has completed initialization"
avril 06 20:45:27 debian systemd[1]: Started Docker Application Container Engine.
avril 06 20:45:27 debian dockerd[21497]: time="2020-04-06T20:45:27.700042020+02:00" level=info msg="API listen on /var/run/docker.sock"
root@debian:/etc/ansible/docker# nano install_docker.yml
root@debian:/etc/ansible/docker# sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2020-04-06 20:45:27 CEST; 5min ago
       Docs: https://docs.docker.com
Main PID: 21497 (dockerd)
   Tasks: 8
  Memory: 50.9M
    CGroup: /system.slice/docker.service
            └─21497 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

```

## 3 GITLAB

### 3.1 Gitlab dockerisé

La plateforme gitlab sera déployée via une image docker. Nous retrouverez le fichier suivant dans le dossier `gitlab` dans les sources fournis :

- **docker-compose.yml** : fichier qui permet de créer votre conteneur
- Il suffit donc d'exécuter la commande suivante afin d'exécuter votre image :

```
docker-compose up
```

#### 4.1.1 Les volumes

Trois volumes seront créés :

Emplacement local	Emplacement du conteneur	Usage
/srv/gitlab/data	/var/opt/gitlab	Pour stocker les données d'application
/srv/gitlab/logs	/var/log/gitlab	Pour stocker les journaux
/srv/gitlab/config	/etc/gitlab	Pour stocker les fichiers de configuration Gitlab

5

### 5.1.1 Accès au gitlab

Gitlab est accessible maintenant sur <http://<IP-SERRNEUR>/>.

## 3.2 Connexion

### 4.2.1 Gestion des mots de passe

Mot de passe *web* pour l'administration via l'interface Web :

- Login : **root**
- Mot de passe : **root**

### 4.2.2 Modification des mots de passes

Cliquer sur votre profil qui est disponible dans la barre de menus située dans le coin supérieur droit (avatar de l'utilisateur) > **Settings > Password**

The screenshot shows the GitLab User Settings page for 'Edit Password'. The sidebar on the left has a red box around the 'Password' item, labeled '2'. The main content area shows the 'Password' section with fields for 'Current password', 'New password', and 'Password confirmation'. A red box labeled '1' highlights the 'Settings' option in the top right corner of the page, which is part of a user menu. The user is identified as 'Administrator @root'.

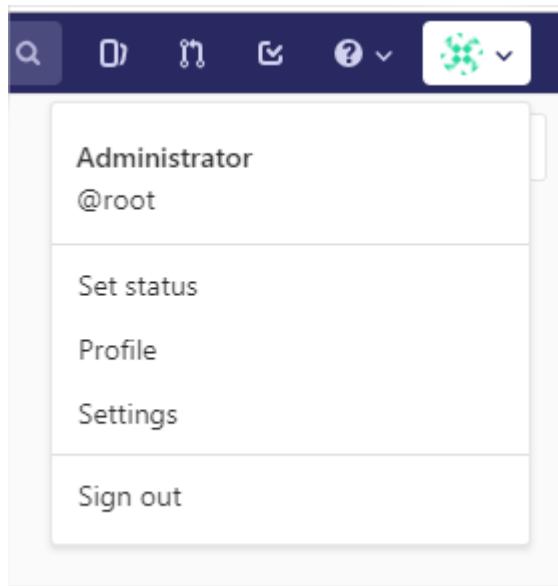
### 4.2.3 Accès WEB

L'interface d'administration est disponible aux adresses suivantes (différentes suivant que l'interface est en **HTTP ou HTTPS**) :

	<b>HTTP</b>	<b>HTTPS</b>
<b>Interface d'administration</b>	http://192.168.1.11:80	https://192.168.1.11:443

### 3.3 Menu utilisateur

Trois sections sont présentées dans le menu de paramétrage utilisateur située dans le coin supérieur droit : **Set status**, **Profile** et **Settings** :



Le menu **Set status** permet fournir un message d'état personnalisé pour votre profil utilisateur avec un emoji qui le décrit. Cela peut être utile lorsque vous êtes absent ou que vous n'êtes pas disponible. Les autres utilisateurs peuvent alors prendre en compte votre statut lorsqu'ils répondent à vos problèmes où vous attribuent un travail. Sachez que votre statut est visible publiquement, même si votre profil est privé.

Le menu **Profile** permet de :

- Consulter votre séquence d'activités et l'historique de vos contributions
- Connaitre les groupes dont vous êtes membre, projets auxquels vous avez contribué, vos projets personnels (en respectant le niveau de visibilité du projet) et vos extraits de code personnels

Le menu **Settings** permet de :

- Mettre à jour vos informations personnelles
- Définir un statut personnalisé pour votre profil
- Gérez votre email de validation pour votre profil
- Gérer 2FA
- Changer votre nom d'utilisateur et supprimer votre compte
- Gérer les applications pouvant utiliser GitLab en tant que fournisseur OAuth
- Gérez des jetons d'accès personnel pour accéder à votre compte via l'API et les applications autorisées.

- Ajouter et supprimer des emails liés à votre compte
- Choisissez le courrier électronique à utiliser pour les notifications, les validations Web et l'affichage dans votre profil public.
- Gérez les clés SSH pour accéder à votre compte via SSH
- Gérez vos préférences pour personnaliser votre propre expérience GitLab
- Nisualisez vos sessions actives et révoquez-en si nécessaire
- Accédez à votre journal d'audit, un journal de sécurité des événements importants impliquant votre compte

### 3.4 Créer un projet

Il est possible de créer un projet directement depuis la page d'accueil en cliquant sur le bouton vert New Project.

The screenshot shows the top navigation bar of the GitLab interface. On the left is the GitLab logo, followed by a dropdown menu labeled "Projects". To its right are "Groups", "More", and several other small icons. Below the navigation is a large header with the word "Projects" in bold. To the right of the header is a green button labeled "New project". Underneath the header, there are three tabs: "Your projects" (which is underlined), "Starred projects", and "Explore projects". Further down are two search/filter boxes: "Filter by name..." and "Last updated". At the bottom of the visible area, there are two buttons: "All" and "Personal".

Nous arrivons ensuite sur la page suivante :

Screenshot of the GitHub 'Create project' dialog for a 'Blank project'.

The dialog includes the following fields:

- Project name:** My awesome project
- Project URL:** http://192.168.1.12: h.ajdaini
- Project slug:** my-awesome-project
- Project description (optional):** Description format
- Visibility Level:**
  -  Private: Project access must be granted explicitly to each user.
  -  Internal: The project can be accessed by any logged in user.
  -  Public: The project can be accessed without any authentication.
- Initialize repository with a README**: Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Buttons at the bottom:

- Create project
- Cancel

Ci-dessous la description des différents champs de la section **Blank project** :

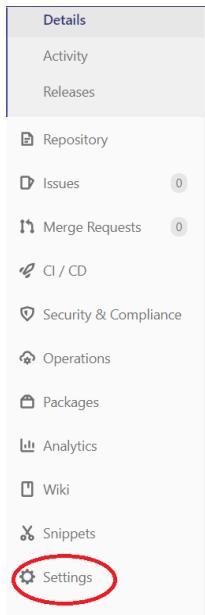
- Le nom de votre projet dans le champ **Project name**. Nous ne pouvez pas utiliser de caractères spéciaux, mais vous pouvez utiliser des espaces, des traits d'union, des traits de soulignement ou même des emoji.

- Le champ **Project description** ( facultatif) vous permet de saisir une description du tableau de bord de votre projet, ce qui aidera les autres utilisateurs à comprendre en quoi consiste votre projet. Bien que ce ne soit pas obligatoire, c'est une bonne idée de remplir ceci.
- Dans le champ **Nisibility Level**, il est possible de choisir son **niveau de visibilité** modifiables droits d'affichage et d'accès du projet pour les utilisateurs.
  - Public : Les projets publics peuvent être clonés **sans aucune** authentification
  - Internal : Les projets internes peuvent être clonés par n'importe quel utilisateur connecté.
  - Private : Les projets privés ne peuvent être clonés et visualisés que par les membres du projet. Ils apparaîtront uniquement aux membres du projet
- La sélection de l'option **Initialize repository with a README** crée un fichier README de sorte que le référentiel Git soit initialisé, possède une branche par défaut et puisse être cloné.

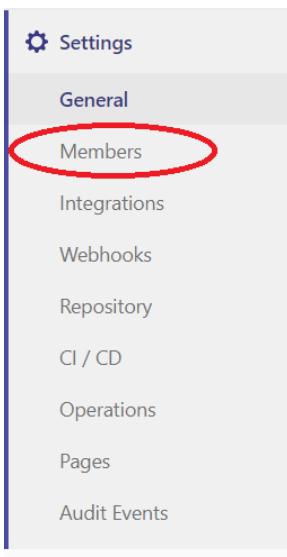
### 3.5 Permissions GitLab et protection des branches

Cette première partie de procédure a pour but d'expliquer la marche à suivre pour donner des permissions aux différents agents qui vont intervenir sur le GitLab et ainsi assurer une première barrière de sécurité contre les mauvaises manipulations des outils mis à disposition.

Se rendre sur GitLab et Ouvrir l'onglet « **Settings** »



Une fois ceci fait, il faut cliquer sur l'onglet « **Members** »



On arrive sur la fenêtre d'invitation des membres et du paramétrage des permissions

#### Project members

You can invite a new member to **automatisator** or invite another group.

Invite member	Invite group
<input type="text" value="GitLab member or Email address"/> <input type="text" value="Search for members to update or invite"/>	
<b>Choose a role permission</b> <input type="text" value="Guest"/> <a href="#">Read more about role permissions</a>	
<b>Access expiration date</b> <input type="text" value="Expiration date"/>	
<input type="button" value="Invite"/> <input type="button" value="Import"/>	

Ici on peut ajouter un membre et choisir son rôle ainsi que la date d'expiration de l'accès au GitLab (Si la Case « **Access expiration date** » est laissée vide, le compte garde l'accès indéfiniment)

On paramètre les rôles, chaque rôle à des droits prédéfinis :

- Le rôle « **Guest** » n'a aucun accès au code.
- Le rôle « **Reporter** » (donné ici au compte de test) à simplement des droits d'accès en lecture au code.
- Le rôle « **Developer** » à des droits de lecture et écriture sur le code.
- Le rôle « **Maintainer** » à des droits de lecture et écriture sur le code ainsi que des droits administrateurs.

On peut aussi noter la possibilité d'ajouter à tout moment une date d'expiration d'accès au GitLab.

On est ici avec le compte de test qui a le statut « **Reporter** »

On voit bien ici que l'on a accès au code, mais lorsque l'on clique sur « **Edit** », il est impossible d'éditer le code et un message nous en informant apparaît. Il se passe de même lorsque l'on clique sur «

Wed IDE », « Replace » ou « Delete ».

Ici, je suis avec mon compte qui a donc le statut « Maintainer », après avoir cliqué sur « Edit » je peux modifier le code.

Edit file

Write Preview changes

Y test main.cpp

```
1 // Your First C++ Program
2
3 #include <iostream>
4
5 int main() {
6     std::cout << "Salut Hatim je suis un test";
7     return 0;
8 }
```

Soft wrap text ▾

Dans la deuxième partie, nous allons voir qu'il y a aussi possibilité de protéger les branches et ainsi mettre en place une deuxième barrière contre les actes malveillants ou des fausses manipulations.

Se rendre sur GitLab et passer le curseur sur l'onglet « Repository » Puis sélectionner « Branches »

The screenshot shows the GitLab project overview for a repository named 'automatisator'. On the left sidebar, the 'Repository' link is circled in red. In the main content area, there is a 'Repository' card with a list of metrics: 0 Issues, 0 Merge Requests, 0 CI / CD, 0 Security & Compliance, 0 Operations, and 0 Packages. To the right of the repository card is a vertical sidebar with several options: Files, Commits, Branches (which is also circled in red), Tags, Contributors, Graph, Compare, and Locked Files. The 'Branches' option is clearly highlighted.

On arrive sur une page avec un lien menant vers le paramétrage des branches

The screenshot shows the 'Protected branches' section of a GitLab project settings. At the top, there are navigation links: 'Overview', 'Active', 'Stale', and 'All'. To the right are buttons for 'Delete merged branches' and 'New branch'. A note says 'Protected branches can be managed in [project settings](#)'. Below this, a table lists 'Active branches': 'master' (protected, default) and 'test' (protected). Each row includes a commit log, a 'Merge request' button, a 'Compare' button, and a trash bin icon. The 'master' row also has a dropdown arrow.

Une fois arriver sur le lien, une liste de possibilité est affichée, celle qui va nous intéresser ici est « **Protected Branches** », Cliquer sur « **Expand** »

#### Default Branch

Select the branch you want to set as the default for this project. All merge requests and commits will automatically be made against this branch unless you specify a different one.

#### Push Rules

Push Rules outline what is accepted for this project.

#### Mirroring repositories

Set up your project to automatically push and/or pull changes to/from another repository. Branches, tags, and commits will be synced automatically. [Read more](#)

#### Protected Branches

Keep stable branches secure and force developers to use merge requests.

#### Protected Tags

Limit access to creating and updating tags.

#### Repository cleanup

Clean up after running [BFG](#) on the repository [?](#)

On arrive sur un menu nous permettant de sélectionner la branche souhaitée ainsi que la possibilité d'affecter des droits à des personnes ou à un rôle sur ladite branche

Protect a branch

Branch:	Select branch or create wildcard Wildcards such as <code>*-stable</code> or <code>production/*</code> are supported		
Allowed to merge:	Select		
Allowed to push:	Select Only groups that <a href="#">have this project shared</a> can be added here		
Require approval from code owners:	<input checked="" type="checkbox"/> Pushes that change filenames matched by the CODEOWNERS file will be rejected		
<a href="#">Protect</a>			
Branch	Allowed to merge	Allowed to push	Code owner approval
master <a href="#">default</a>	4 users	4 users	<input type="checkbox"/>
<a href="#">Unprotect</a>			

Comme on peut le voir, seule la branche master est protégée pour le moment, c'est normal, la première branche sur projet qui est créée est protégée par défaut. On sélectionne la branche souhaitée, ici ça sera la branche « **test** »

Protect a branch

Branch:	test Wildcards such as <code>*-stable</code> or <code>production/*</code> are supported
Allowed to merge:	Select
Allowed to push:	Select Only groups that <a href="#">have this project shared</a> can be added here
Require approval from code owners:	<input checked="" type="checkbox"/> Pushes that change filenames matched by the CODEOWNERS file will be rejected
<a href="#">Protect</a>	

Maintenant que la branche est choisie, on peut appliquer des droits à des personnes et/ou à un rôle via le menu déroulant.

Protect a branch

Branch:	<input type="text" value="test"/>	<small>Wildcards such as <code>*-stable</code> or <code>production/*</code> are supported</small>
Allowed to merge:	<input type="text" value="Developers + Maintainers"/>	
Allowed to push:	<input type="text" value="Developers + Maintainers"/>	
Require approval from code owners:	<input checked="" type="checkbox"/> Pushes that change filenames matched by the CODEOWNERS file will be rejected	
<input type="button" value="Protect"/>		

Après avoir choisi qui pourra utiliser ou push du code sur cette branche, on peut cliquer sur « **Protect** » (Le bouton permet de dire si oui ou non on souhaite que le propriétaire du code donne son accord en cas de modification)

Branch	Allowed to merge	Allowed to push	Code owner approval
master <small>default</small>	Maintainers	Maintainers	<input checked="" type="checkbox"/> Unprotect
test	Developers + Mai...	Developers + Mai...	<input checked="" type="checkbox"/> Unprotect

On peut voir que maintenant la branche « **test** » est protégée elle aussi. Les droits ont ainsi été distribués de sorte que les développeurs puissent faire leurs tests sans encombre mais ne soient pas autorisés à fusionner du code ou encore le push dans la branche « **master** ». Si on souhaite ne plus protéger une branche, il suffit de cliquer sur le bouton « **Unprotect** ».

## 3.6 Installation du Runner

### 4.6.1 Installation de GitLab Runner (Windows)

Il faut au préalable installer Git et avoir un projet à associer à GitLab Runner téléchargeable ici : <https://git-scm.com/downloads>. Si vous sélectionnez Windows, vérifiez bien votre type d'architecture en faisant comme suit :

Ouvrir un explorateur de fichiers -> clic droit sur « Ce PC » -> Propriétés -> Dans Système -> Type du Système.

Dans cette première partie, on vous présente comment aller chercher GitLab Runner et les paramétrages à effectuer pour son bon fonctionnement. Pour commencer, il faut se rendre sur la page de sélection de l'OS : <https://docs.gitlab.com/runner/>

En parcourant la page, on va se retrouver face à ça, dans notre cas on va choisir « **Install on Windows** »

#### Install GitLab Runner

GitLab Runner can be [installed](#) and used on GNU/Linux, macOS, FreeBSD, and Windows. You can install it using Docker, download the binary manually or use the repository for rpm/deb packages that GitLab offers. Below you can find information on the different installation methods:

- Install using GitLab's repository for Debian/Ubuntu/CentOS/RedHat (preferred).
- Install on GNU/Linux manually (advanced).
- Install on macOS.
- Install on Windows.
- Install as a Docker service.
- Install in autoscaling mode using Docker machine.
- Install on FreeBSD.
- Install on Kubernetes.
- Install the nightly binary manually (development).

Une fois ceci fait, suivez les instructions du passage « **Installation** »

#### Installation

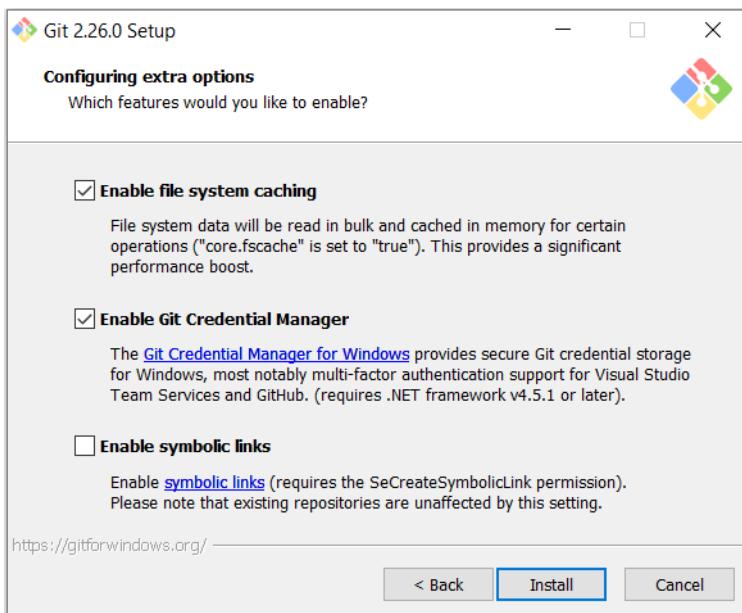
**Important:** With GitLab Runner 10, the executable was renamed to `gitlab-runner`. If you want to install a version prior to GitLab Runner 10, [visit the old docs](#).

1. Create a folder somewhere in your system, ex: `C:\GitLab-Runner`.
2. Download the binary for [x86](#) or [amd64](#) and put it into the folder you created. Rename the binary to `gitlab-runner.exe`. You can download a binary for every available version as described in [Bleeding Edge - download any other tagged release](#).
3. Run an [elevated command prompt](#):
4. [Register the Runner](#).
5. Install the Runner as a service and start it. You can either run the service using the Built-in System Account (recommended) or using a user account.

**Run service using Built-in System Account** (under directory created in step 1. from above, ex: `C:\GitLab-Runner`)

Après avoir téléchargé l'installateur du GitLab-Runner, lancer le et laisser les options par défaut

Cliquer sur « **Install** » et à la fin du chargement, cliquer sur « **Next** »



Maintenant que l'installation est terminée, on peut passer à la configuration et à l'association d'un projet. On peut reprendre la documentation GitLab.

## Installation

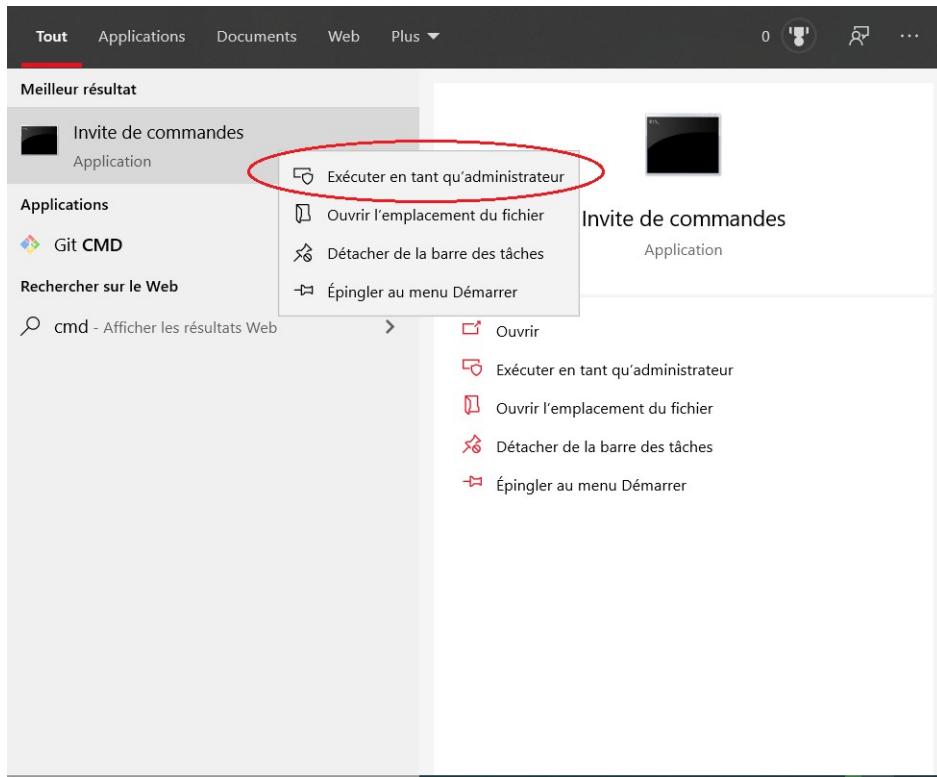
**⚠ Important:** With GitLab Runner 10, the executable was renamed to `gitlab-runner`. If you want to install a version prior to GitLab Runner 10, [visit the old docs](#).

1. Create a folder somewhere in your system, ex.: `C:\GitLab-Runner`.
2. Download the binary for [x86](#) or [amd64](#) and put it into the folder you created. Rename the binary to `gitlab-runner.exe`. You can download a binary for every available version as described in [Bleeding Edge - download any other tagged release](#).
3. Run an [elevated command prompt](#):
4. [Register the Runner](#).

5. Install the Runner as a service and start it. You can either run the service using the Built-in System Account (recommended) or using a user account.

**Run service using Built-in System Account** (under directory created in step 1. from above, ex.: `C:\GitLab-Runner`)

## On ouvre un invit  de commande en mode administrateur



Il va ensuite falloir acc der au dossier o u on a placer le gitlab-runner.exe par d faut sur `C:\GitLab-Runner`

```
cd C:\GitLab-Runner
```

On rentre la commande pour le lancer (il se peut qu'il soit lanc  par d faut).

```
gitlab-runner.exe start
```

```
C:\GitLab-Runner>gitlab-runner.exe start
Runtime platform          arch=amd64 os=windows pid=19380 revision=4c96e5ad version=12.9.0
```

Si on veut voir la version du Runner

```
gitlab-runner --version
```

Pour associer un projet, on va arr ter le service

```
gitlab-runner.exe stop
```

```
C:\GitLab-Runner>gitlab-runner.exe stop
Runtime platform          arch=amd64 os=windows pid=12960 revision=4c96e5ad version=12.9.0
```

Puis le service d'enregistrement

```
gitlab-runner.exe register stop
```

#### 4.6.2 Installation de GitLab Runner (Linux)

Cette partie permet d'installer GitLab Runner sous une machine **linux**. Il faut d'abord, comme avant chaque installation sous linux, vérifier qu'il n'y a pas **d'update** possible de l'OS.

On se met donc en administrateur avec la commande

```
su
```

Puis on lance la mise à jour avec la commande :

```
apt update
```

Il va falloir installer **Git** et **Curl** pour faire l'installation et la configuration de GitLab Runner.

```
apt install git
```

Une fois les deux outils installés, on peut faire la commande d'installation de GitLab Runner :

```
curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-  
runner/script.deb.sh | sudo bash
```

```
apt install curl
```

On va stopper le service **gitlab-runner** et directement lancer le service d'enregistrement

```
gitlab-runner stop  
gitlab-runner register
```

#### 4.6.3 Association d'un projet GitLab au Runner (Windows & Linux)

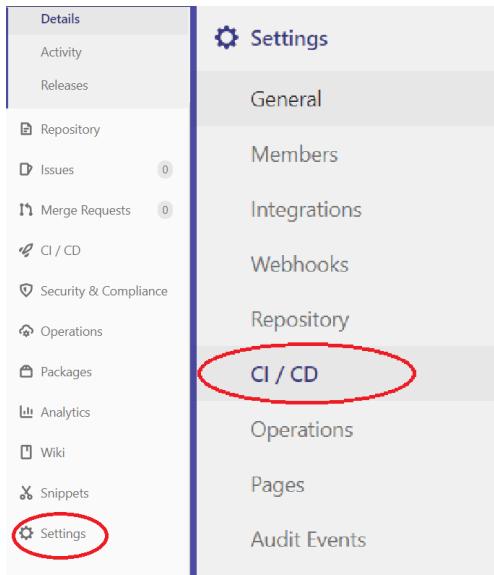
Directement, il demande de rentrer une URL, l'adresse qui doit être utilisée dans notre cas est l'adresse du serveur cible

```
@ip serveur cible
```

```
C:\GitLab-Runner>gitlab-runner.exe register stop  
Runtime platform arch=amd64 os=windows pid=15152 revision=4c96e5ad version=12.9.0  
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
```

Après avoir rentré l'URL, GitLab Runner demande un **Token**, ce token va lier notre projet au runner, on

le trouve en allant dans l'onglet « **Settings** » du projet puis dans « **CI/CD** »



Le menu qui nous intéresse est « **Runner** », cliquer sur « **Expand** »

## Runners

Runners are processes that pick up and execute jobs for GitLab. Here you can register and see your Runners for this project. [More information](#)

Expand

Ici, on voit la partie « **Set up a specific Runner manually** », dans le 3 de cette partie se trouve notre **Token**, il faut le copier et l'ajouter dans notre invité de commande

## Specific Runners

### Set up a specific Runner automatically

You can easily install a Runner on a Kubernetes cluster.  
[Learn more about Kubernetes](#)

1. Click the button below to begin the install process by navigating to the Kubernetes page
2. Select an existing Kubernetes cluster or create a new one
3. From the Kubernetes cluster details view, install Runner from the applications list

[Install Runner on Kubernetes](#)

### Set up a specific Runner manually

1. [Install GitLab Runner](#)
2. Specify the following URL during the Runner setup:
3. Use the following registration token during setup:
4. Start the Runner!

```
C:\GitLab-Runner>gitlab-runner.exe register stop
Runtime platform           arch=amd64 os=windows pid=9788 revision=4c96e5ad version=12.9.0
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
https://gitlab.com/
Please enter the gitlab-ci token for this runner:
VxCquHTQrG9E_VuLtra7
```

Il faut ensuite donner une description au Runner, on peut rentrer ce qu'on veut ici

```
C:\GitLab-Runner>gitlab-runner.exe register stop
Runtime platform          arch=amd64 os=windows pid=9788 revision=4c96e5ad version=12.9.0
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
https://gitlab.com/
Please enter the gitlab-ci token for this runner:
VxCquHTQrG9E_Vultra7
Please enter the gitlab-ci description for this runner:
[VAL]: Runner_Projet_Automatisator
```

Maintenant, il faut rentrer des tags, entrer **ssh, ci**

```
C:\GitLab-Runner>gitlab-runner.exe register stop
Runtime platform          arch=amd64 os=windows pid=9788 revision=4c96e5ad version=12.9.0
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
https://gitlab.com/
Please enter the gitlab-ci token for this runner:
VxCquHTQrG9E_Vultra7
Please enter the gitlab-ci description for this runner:
[VAL]: Runner_Projet_Automatisator
Please enter the gitlab-ci tags for this runner (comma separated):
ssh, ci
```

Après avoir fait ça, le Runner doit retourner comme résultat « **Registering runner... succeeded** »

```
C:\GitLab-Runner>gitlab-runner.exe register stop
Runtime platform          arch=amd64 os=windows pid=9788 revision=4c96e5ad version=12.9.0
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
https://gitlab.com/
Please enter the gitlab-ci token for this runner:
VxCquHTQrG9E_Vultra7
Please enter the gitlab-ci description for this runner:
[VAL]: Runner_Projet_Automatisator
Please enter the gitlab-ci tags for this runner (comma separated):
ssh, ci
Registering runner... succeeded           runner=VxCquHTQ
Please enter the executor: docker-ssh, parallels, shell, virtualbox, docker-ssh+machine, custom, docker, docker-windows,
ssh, docker+machine, kubernetes:
```

Il demande une dernière chose, choisir un exécuteur, on a choisi le « **Shell** ». Cet exécuteur va permettre d'exécuter des commandes linux sur un serveur cible, noter qu'on peut rajouter des exécuteurs par la suite (ex : **docker...**)

```
C:\GitLab-Runner>gitlab-runner.exe register stop
Runtime platform          arch=amd64 os=windows pid=9788 revision=4c96e5ad version=12.9.0
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
https://gitlab.com/
Please enter the gitlab-ci token for this runner:
VxCquHTQrG9E_Vultra7
Please enter the gitlab-ci description for this runner:
[VAL]: Runner_Projet_Automatisator
Please enter the gitlab-ci tags for this runner (comma separated):
ssh, ci
Registering runner... succeeded           runner=VxCquHTQ
Please enter the executor: docker-ssh, parallels, shell, virtualbox, docker-ssh+machine, custom, docker, docker-windows, ssh, docker+machine, kubernetes:
shell
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!
C:\GitLab-Runner>
```

Désormais, le GitLab-Runner, est associé et actif sur notre projet, pour s'en assurer on peut retourner sur la page où on a trouvé le **Token** et observé qu'un runner est actif sur le projet

## Runners activated for this project

The screenshot shows a list of runners for a specific project. At the top, there is a header with a warning icon, the runner's name ('KyhNp-3a...'), a lock icon, and edit, pause, and remove buttons. Below the header, the runner's details are listed: 'Runner\_Projet\_Automatisator' and '#1820641'. Underneath, there are two buttons: 'ci' (which is highlighted in blue) and 'ssh'.

## **3.7 Gitlab-CI**

Une fois le runner installé et configuré sur un serveur web, vous pouvez vous attaquer à la partie d'intégration continue dans Gitlab.

### **4.7.1 Tests unitaires**

Pour les tests unitaires nous utiliserons « PHPUnit », dans lequel il nous fournit toute une suite de classes et méthodes utiles pour faciliter l'écriture de nos tests. Pour sa respective installation via Composer, nous devons exécuter la ligne suivante depuis de notre projet symfony sur votre serveur web :



```

root@debian:/var/www/automatisator6# composer require --dev phpunit/phpunit ^7.5.17
Do not run Composer as root/super user! See https://getcomposer.org/root for details
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Restricting packages listed in "symfony/symfony" to "5.0.*"

Prefetching 3 packages ⏳
  - Downloading (100%)

Package operations: 34 installs, 0 updates, 0 removals
- Installing sebastian/version (2.0.1): Loading from cache
- Installing sebastian/resource-operations (2.0.1): Loading from cache
- Installing sebastian/recursion-context (3.0.0): Loading from cache
- Installing sebastian/object-reflector (1.1.1): Loading from cache
- Installing sebastian/object-enumerator (3.0.3): Loading from cache
- Installing sebastian/global-state (2.0.0): Loading from cache
- Installing sebastian/exporter (3.1.2): Loading from cache
- Installing sebastian/environment (4.2.3): Loading from cache
- Installing sebastian/diff (3.0.2): Loading from cache
- Installing sebastian/comparator (3.0.2): Loading from cache
- Installing phpunit/php-timer (2.1.2): Loading from cache
- Installing phpunit/php-text-template (1.2.1): Loading from cache
- Installing phpunit/php-file-iterator (2.0.2): Loading from cache
- Installing theseer/tokenizer (1.1.3): Loading from cache
- Installing sebastian/code-unit-reverse-lookup (1.0.1): Loading from cache
- Installing phpunit/php-token-stream (3.1.1): Loading from cache
- Installing phpunit/php-code-coverage (6.1.4): Loading from cache
- Installing phpspec/prophecy (v1.10.3): Loading from cache
- Installing phar-io/version (2.0.1): Loading from cache
- Installing phar-io/manifest (1.0.3): Loading from cache
- Installing myclabs/deep-copy (1.9.5): Loading from cache
- Installing phpunit/phpunit (7.5.20): Loading from cache
- Installing symfony/web-profiler-bundle (v5.0.7): Loading from cache
- Installing symfony/profiler-pack (v1.0.4): Loading from cache
- Installing easycorp/easy-log-handler (v1.0.9): Loading from cache
- Installing symfony/debug-bundle (v5.0.7): Loading from cache
- Installing symfony/debug-pack (v1.0.7): Loading from cache
- Installing nikic/php-parser (v4.3.0): Loading from cache
- Installing symfony/maker-bundle (v1.14.6): Loading from cache
- Installing symfony/phpunit-bridge (v5.0.7): Loading from cache
- Installing symfony/css-selector (v5.0.7): Loading from cache
- Installing symfony/dom-crawler (v5.0.7): Loading from cache
- Installing symfony/browser-kit (v5.0.7): Loading from cache

* Instead:
  1. Remove it now: composer remove --dev phpunit/phpunit
  2. Use Symfony's bridge: composer require --dev phpunit

How to test?

* Write test cases in the tests/ folder
* Run php bin/phpunit

root@debian:/var/www/automatisator6#

```

Nous pouvons tester qu'il est bien installé, nous lançons la commande suivante :

```
# php bin/phpunit
```

Il installera toutes les packages nécessaires pour son bon fonctionnement

```

root@debian:/var/www/automatisator6# php bin/phpunit
Do not run Composer as root/super user! See https://getcomposer.org/root for details
Creating a "phpunit/phpunit" project at "./phpunit-8.3-0"
Installing phpunit/phpunit (8.3.5)
Plugins have been disabled.
- Installing phpunit/phpunit (8.3.5): Loading from cache
Created project in /var/www/automatisator6/bin/.phpunit/phpunit-8.3-0
Do not run Composer as root/super user! See https://getcomposer.org/root for details
Do not run Composer as root/super user! See https://getcomposer.org/root for details
Do not run Composer as root/super user! See https://getcomposer.org/root for details
composer.json has been updated
Do not run Composer as root/super user! See https://getcomposer.org/root for details
Do not run Composer as root/super user! See https://getcomposer.org/root for details
Loading composer repositories with package information
Updating dependencies
Package operations: 23 installs, 0 updates, 0 removals
- Installing phar-io/version (2.0.1): Loading from cache
- Installing phar-io/manifest (1.0.3): Loading from cache
- Installing phpuunit/php-timer (2.1.2): Loading from cache
- Installing sebastian/recursion-context (3.0.0): Loading from cache
- Installing sebastian/exporter (3.1.2): Loading from cache
- Installing sebastian/diff (3.0.2): Loading from cache
- Installing sebastian/comparator (3.0.2): Loading from cache
- Installing sebastian/object-reflector (1.1.1): Loading from cache
- Installing sebastian/global-state (3.0.0): Loading from cache
- Installing sebastian/object-enumerator (3.0.3): Loading from cache
- Installing sebastian/resource-operations (2.0.1): Loading from cache
- Installing sebastian/type (1.1.3): Loading from cache
- Installing doctrine/instantiator (1.3.0): Loading from cache
- Installing myclabs深深-copy (1.9.5): Loading from cache
- Installing theseer/tokenizer (1.1.3): Loading from cache
- Installing sebastian/version (2.0.1): Loading from cache
- Installing sebastian/environment (4.2.3): Loading from cache
- Installing sebastian/code-unit-reverse-lookup (1.0.1): Loading from cache
- Installing phpuunit/php-text-template (1.2.1): Loading from cache
- Installing phpuunit/php-token-stream (3.1.1): Loading from cache
- Installing phpuunit/php-file-iterator (2.0.2): Loading from cache
- Installing phpuunit/php-code-coverage (7.0.10): Loading from cache
- Installing symfony/phpunit-bridge (dev-master): Symlinking from /var/www/automatisator6/vendor/symfony/phpunit-bridge
Writing lock file
Generating optimized autoload files
PHPUnit 8.3.5 by Sebastian Bergmann and contributors.

Time: 7.52 seconds, Memory: 4.00 MB
No tests executed!

```

Nous pouvons commencer à tester phpunit avec la commande suivante :

```

root@debian:/var/www/automatisator6# php bin/phpunit
PHPUnit 8.3.5 by Sebastian Bergmann and contributors.

Testing Project Test Suite
.
1 / 1 (100%)

Time: 49 ms, Memory: 4.00 MB
OK (1 test, 1 assertion)

```

Dans la structure de Symfony, nous avons par défaut un dossier « Tests » qui contiendra l'ensemble de nos tests.

/var/www/automatisator6/		
Name	Size (KB)	Last modified
..		2020-0-
bin		2020-0-
config		2020-0-
public		2020-0-
src		2020-0-
templates		2020-0-
tests		2020-0-

Pour la création de notre 1er test, nous allons créer un fichier appelé *AppTest.php*

```
nano AppTest.php
```

Ensuite nous ajouterons les lignes suivantes, où nous testerons la fonction « **testTestAreWorking** »

```
<?php

namespace App\Tests;

use PHPUnit\Framework\TestCase;

class AppTest extends TestCase{

    public function testTestAreworking () {
        $this->assertEquals(2, 1+1);

    }
}
```

Nous relançons ensuite la commande **phpunit**

```
root@debian:/var/www/automatisator6# php bin/phpunit
PHPUnit 8.3.5 by Sebastian Bergmann and contributors.

Testing Project Test Suite
.
1 / 1 (100%)

Time: 131 ms, Memory: 4.00 MB

OK (1 test, 1 assertion)
root@debian:/var/www/automatisator6#
```

Pour le test avec une base de données, nous utiliserons les Fixtures. Nous installons le package avec composer :

```
composer require --dev orm-fixtures
```

Après l'installation, nous aurons un nouveau dossier appelé *DataFixtures*. C'est dossier contiendra différentes fixtures que nous pourrons utiliser.

/var/www/automatisator6/src/		
Name	Size (KB)	Last modified
..		2020-0
Controller		2020-0
DataFixtures		2020-0
Entity		2020-0
Migrations		2020-0
Repository		2020-0
Kernel.php	2	2020-0

Pour notre test, nous créerons un Entity `User` pour simuler un compte utilisateurs :

```
# php bin/console make:user
```

```
root@debian:/var/www/automatisator6# php bin/console make:user
The name of the security user class (e.g. User) [User]:
>

Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
>

Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:
>

Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by some other system.
Does this app need to hash/check user passwords? (yes/no) [yes]:
>

created: src/Entity/User.php
created: src/Repository/UserRepository.php
updated: src/Entity/User.php
updated: config/packages/security.yaml

Success!

Next Steps:
- Review your new App\Entity\User class.
- Use make:entity to add more fields to your User entity and then run make:migration.
- Create a way to authenticate! See https://symfony.com/doc/current/security.html
```

Après nous allons créer une nouvelle fixture appelée `UserFixtures.php` où nous ajouterons les lignes suivantes :

```
<?php

namespace App\DataFixtures;
use App\Entity\User;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Common\Persistence\ObjectManager;

class UserFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {
        for ($i = 0; $i < 10; $i++){
            $user = (new User())
                ->setEmail("user$i@domain.fr")
                ->setPassword("0000");
        }
    }
}
```

```

        $manager->persist($user);
    }
    $manager->flush();
}
}

```

De la même manière, nous allons créer un nouveau test nommé `UserRepositoryTest.php` qui nous permettra tester notre Entity User, dans notre exemple nous allons par exemple que le nombre d'utilisateurs est égale à 19 :

/var/www/automatisator6/tests/		
Name	Size (KB)	Last modified
..		
AppTest.php	1	2020-04-14 11:43:10
bootstrap.php	1	2020-04-14 11:43:10
UserRepositoryTest.php	0	2020-04-14 11:43:10

```

<?php

namespace App\Test\Repository;

use App\Repository\UserRepository;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;

class UserRepositoryTest extends KernelTestCase
{
    public function testcount(){
        self::bootKernel();
        $users = self::$container->get(UserRepository::class)->count([]);
        $this->assertEquals(10, $users);
    }
}

```

Nous modifierons également notre **fichier** `.test.env` et nous allons indiquer les informations de connexions à notre base de données Mysql (Nous devons au préalable avoir installé Mysql et configurer l'accès au compte root).

```

# define your env variables for the test env here
KERNEL_CLASS='App\Kernel'
APP_SECRET='$secretf0rt3st'

```

```
SYMFONY_DEPRECATIONS_HELPER=999999  
  
PANTHER_APP_ENN=panther  
DATABASE_URL=mysql://root:root@127.0.0.1:3306/tuto_test?serverVersion=10.3
```

Cela étant, nous créons la structure de notre base de données avec la commande suivante :

```
php bin/console doctrine:database:create --env=test  
php bin/console doctrine:schema:update --env=test --force
```

```
root@debian:/var/www/automatisator6# php bin/console doctrine:database:create --env=test  
Created database `tuto_test` for connection named default  
root@debian:/var/www/automatisator6# nano .env.test  
root@debian:/var/www/automatisator6# php bin/console doctrine:schema:update --env=test --force  
  
Updating database schema...  
  
    1 query was executed  
  
[OK] Database schema updated successfully!
```

Nous installerons aussi « `liip test fixtures` », un bundle que nous permettra tester notre test `UserRepositoryTest.php`

```
composer require --dev liip/test-fixtures-bundle:^1.0.0
```

Nous ajoutons le bundle dans le test `UserRepositoryTest.php`

```
nano tests/ UserRepositoryTest.php
```

```
1 <?php  
2  
3 namespace App\Test\Repository;  
4  
5 use App\DataFixtures\UserFixtures;  
6 use App\Repository\UserRepository;  
7 use Liip\TestFixturesBundle\Test\FixturesTrait;  
8 use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;  
9  
10 class UserRepositoryTest extends KernelTestCase  
11 {  
12     use FixturesTrait;  
13  
14     public function testcount()  
15     {  
16         self::bootKernel();  
17         $this->loadFixtures([UserFixtures::class]);  
18         $users = self::$container->get(UserRepository::class)->count([]);  
19         $this->assertEquals(10, $users);  
20     }  
21 }
```

Finalement, nous vérifions que le test fonctionne :

```
php bin/phpunit
```

```
root@debian:/var/www/automatisator6# php bin/phpunit
PHPUnit 8.3.5 by Sebastian Bergmann and contributors.

Testing Project Test Suite
..
2 / 2 (100%)

Time: 432 ms, Memory: 20.00 MB

OK (2 tests, 2 assertions)
root@debian:/var/www/automatisator6#
```

Il suffit maintenant de créer un fichier `.gitlab-ci.yml` à la racine du projet symfony dans le Gitlab, ainsi pour chaque commit ou push de votre projet vous déclencherez votre pipeline CI, vous devez:

Assurez-vous que votre projet est configuré pour utiliser un Runner.

Le fichier `.gitlab-ci.yml` indique au GitLab Runner quoi faire. Notre pipeline comportera trois étapes :

- test\_unit
- test\_vuln
- deploy

Nous utiliserons un Runner de type shell afin de lancer directement la commande `php bin/phpunit` sur notre serveur web.

Noici à quoi ressemble donc la première étape de notre `.gitlab-ci.yml`:

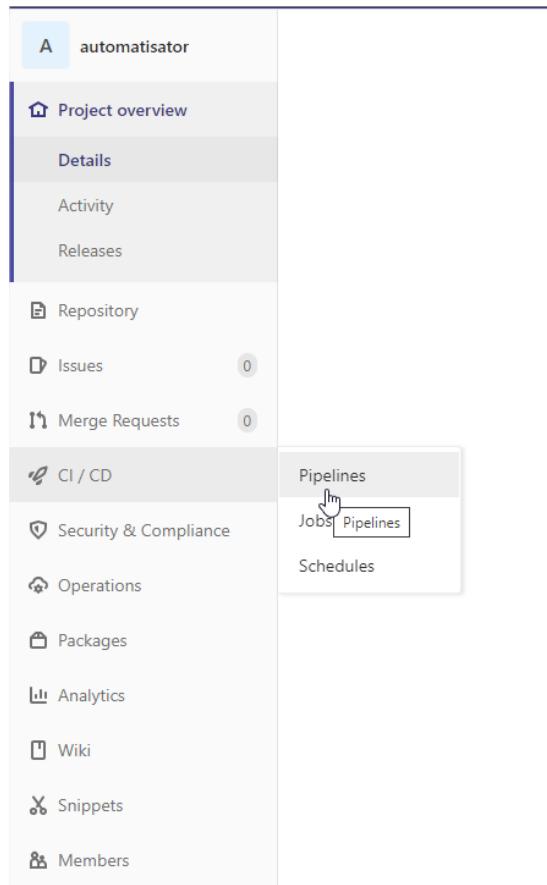
```
stages:
- test_unit
- test_vuln
- deploy

test_unit:
  tags:
    - nom_du_runner
  stage: test_unit
  script: php bin/phpunit
  only:
    - test

test_vuln:
  tags:
    - nom_du_runner
  stage: test_vuln
  script: à rajouter
  only:
    - test
```

```
deploy:  
  tags:  
    - nom_du_runner  
  stage: deploy  
  script: à rajouter  
  only:  
    - test
```

Dorénavant pour chaque commit une pipeline CI se lancera, pour voir votre pipeline il suffit de survoler **CI/CD** et cliquer sur **Pipelines** :



Nous obtiendrez ainsi le pipeline suivant :

automatisator > Pipelines > #54508

running Pipeline #54508 triggered just now by AJDAINI Hatim

## Add new file

3 jobs for test

latest

8e700861 ...

No related merge requests found.

Pipeline Jobs 3 Tests 0

Test\_unit

Test\_vuln

Deploy

test\_unit



test\_vuln



deploy



### 4.7.2 Tests continus de vulnérabilités

Ce guide décrit comment intégrer des tests continus de vulnérabilités dans votre pipeline d'intégration continue. L'outil de scan **Gemnasium** est utilisé dans le projet Automatisator.

Si vous utilisez GitLab CI / CD, vous pouvez analyser vos dépendances pour les vulnérabilités connues à l'aide de l'analyse des dépendances. Toutes les dépendances sont analysées, y compris les dépendances transitives (également appelées dépendances imbriquées).

Nous pouvons tirer parti de l'analyse des dépendances en incluant le travail CI dans votre fichier `.gitlab-ci.yml` existant.

**! Pour exécuter ces tests de vulnérabilités, par défaut, vous avez besoin de GitLab Runner avec l'exécuteur docker ou kubernetes exécutant en mode privilégié.**

Nous allons donc procéder à remplir la deuxième étape de notre fichier `.gitlab-ci.yml` :

```
stages:  
- test_unit  
- test_vuln  
- deploy
```

```
test_unit:
  tags:
    - nom_du_runner
  stage: test_unit
  script: php bin/phpunit

test_vuln:
  tags:
    - nom_du_runner
  stage: test_vuln
  script: à rajouteronly:
    - test

dependency_scanning:
  tags:
    - nom_du_runner
  image: docker:stable
  variables:
    DOCKER_DRINER: overlay2
  services:
    - docker:stable-dindscript:
      - export SP_NERSION=$(echo "$CI_SERNER_NERSION" | sed 's/^([0-9]*\.)\.\([0-9]*\)\.*\//1-2-stable/')
      - docker run
      --
  env DEP_SCAN_DISABLE_REMOTE_CHECKS="${DEP_SCAN_DISABLE_REMOTE_CHECKS:-false}"
    --volume "$PWD:/code"
    --volume /var/run/docker.sock:/var/run/docker.sock"URL du serveur
    Web du Client:$SP_NERSION" /code
  artifacts:
    paths: [gl-dependency-scanning-report.json]only:
      - test

deploy:
  tags:
    - nom_du_runner
  stage: deploy
  script: à rajouteronly:
    - test
```

Nous obtiendrez ainsi le pipeline suivant :

automatisator > Pipelines > #54508

⌚ running Pipeline #54508 triggered just now by 🧑 AJDAINI Hatim

## Add new file

⌚ 3 jobs for test

⚡ latest

⌚ 8e700861 ⋮ ⌂

⚠ No related merge requests found.

Pipeline Jobs 3 Tests 0

```
graph LR; A((Test_unit)) --> B(( )); B --> C((Test_vuln)); C --> D((Deploy));
```

The screenshot shows a CI pipeline interface. At the top, it says "Pipeline #54508 triggered just now by AJDAINI Hatim". Below that is a section titled "Add new file" with a status message "⌚ 3 jobs for test". Underneath are buttons for "latest" and a specific commit hash "8e700861". A note says "⚠ No related merge requests found.". At the bottom, there's a summary "Pipeline Jobs 3 Tests 0" and a visual representation of the pipeline stages: Test\_unit, Test\_vuln, and Deploy. Each stage has a circular icon with a checkmark or a refresh symbol, indicating its current status.

Les résultats sont peuvent être triés selon la gravité de la vulnérabilité :

1. Critique
2. Haute
3. Moyen
4. Faible
5. Inconnue
6. Tout le reste

Severity	Confidence	Report type	
All severities	All confidence levels	All report types	<input checked="" type="checkbox"/> Hide dismissed
Severity	Confidence	Vulnerability	
CRITICAL	Unknown	CVE-2017-18269 in glibc GitLab / security-reports	
MEDIUM	High	Cipher with no integrity GitLab / security-reports	
MEDIUM	High	ECB mode is insecure GitLab / security-reports	
MEDIUM	Medium	Predictable pseudorandom number generator GitLab / security-reports	
MEDIUM	Unknown	CVE-2016-10228 in glibc GitLab / security-reports	
LOW	Medium	X-Content-Type-Options Header Missing GitLab / security-reports	
LOW	Medium	X-Content-Type-Options Header Missing GitLab / security-reports	
LOW	Unknown	CVE-2010-4052 in glibc GitLab / security-reports	
LOW	Unknown	CVE-2018-18520 in elfutils GitLab / security-reports	

### 4.7.3 Déploiement continu

La dernière étape manquante et l'étape deploy soit le déploiement continu de notre code en production c'est-à-dire dans le branche de test et dans le serveur de production.

```

stages:
  - test_unit
  - test_vuln
  - deploy

test_unit:
  tags:
    - nom_du_runner
  stage: test_unit
  script: php bin/phpunit

test_vuln:
  tags:
    - nom_du_runner
  stage: test_vuln
  script: à rajouter
  only:
    - test

```

```

dependency_scanning:
  tags:
    - nom_du_runner
  image: docker:stable
  variables:
    DOCKER_DRINER: overlay2
  services:
    - docker:stable-dind
  script:
    - export SP_NERSION=$(echo "$CI_SERNER_NERSION" | sed
's/^([0-9]*\.)\.\(([0-9])*\)\.*/\1-\2-stable/')
    - docker run
    --
env DEP_SCAN_DISABLE_REMOTE_CHECKS="${DEP_SCAN_DISABLE_REMOTE_CHECKS:-false}"
    --volume "$PWD:/code"
    --volume /var/run/docker.sock:/var/run/docker.sock
    "URL du serveur Web du Client:$SP_NERSION" /code
  artifacts:
    paths: [gl-dependency-scanning-report.json]
only:
  - test

deploy:
  tags:
    - nom_du_runner
  stage: deploy
  script:
    - git push origin master
    - rsync -av --exclude '/var' --exclude './.git' --exclude '.gitignore' --
exclude './.env' --exclude '/src/DataFixtures'. SERNER_CIBLE:CHEMIN_SOURCE_CIBLE

only:
  - test

```

## 3.8 Paramètres réseaux

### 4.8.1 DNS

Les pages GitLab s'attendent à s'exécuter sur leur propre hôte virtuel.

---

#### 4.8.2 SMTP

Pour rajouter votre serveur SMTP vous pouvez soit modifier le fichier en dehors de votre conteneur Docker **/srv/gitlab/config/gitlab.rb** soit directement le fichier **/etc/gitlab/gitlab.rb** dans votre conteneur.

### 3.9 Intégration de l'annuaire Active Directory

#### 4.9.1 Déclaration de l'annuaire

La configuration initiale de LDAP dans GitLab nécessite des modifications soit dans le fichier en Dehors de votre conteneur Docker **/srv/gitlab/config/gitlab.rb** soit directement dans le fichier

**/etc/gitlab/gitlab.rb** dans votre conteneur. Une section dédiée à LDAP est disponible dans le fichier ce configuration, elle a été configuré par l'ingénieur devops, elle ressemble à ceci :

```
gitlab_rails['ldap_enabled'] = true

gitlab_rails['ldap_servers'] = YAML.load <<-EOS
main:
  label: 'votre LDAP'
  host: 'IP DU SERNEUR LDAP'
  port: 389
  method: 'plain'
  uid: 'sAMAccountName'
  bind_dn: 'CN=ldap-user,OU=NOSTRE ORGINISATION, DC=NOSTRE DC,DC=LOCAL'
  password: NOTRE MOT DE PASSE
  timeout: 10
  active_directory: true
  allow_username_or_email_login: false
  block_auto_created_users: false
  base: 'DC=NOSTRE DC,DC=LOCAL'
  user_filter: '(memberOf:1.2.840.113556.1.4.1941:=CN=GITLAB-USERS,CN=Users,DC=CORP,DC=COM)'
EOS
```

Quand vous passerez sous LDAP, il suffira de changer la méthode de 'plain' à 'tls'.

#### 4.9.2 Nérification de votre config ldap

Pour tester une modification de votre configuration ldap il est possible de lancer la commande suivante directement depuis votre conteneur :

```
gitlab-rake gitlab:ldap:check
```

Nous aurez un message de succès si votre configuration est bonne sinon vous aurez directement un message d'erreur vous indiquant d'où vient l'erreur.

## 4 PROMETHEUS ET GRAFANA

### 4.1 Prometheus et grafana dockerisé

Il a choisi été choisie de déployer la plateforme Prometheus et grafana via une image docker. Afin de lancer les conteneurs Prometheus et Grafana, il suffit de se positionner à la racine du projet comme suit :

```
root@ubuntu:~/poc_monitoring/apache-nginx-grafana# ls
apache2 docker-compose.yml grafana nginx prometheus var web
root@ubuntu:~/poc_monitoring/apache-nginx-grafana# █
```

Et de lancer la commande suivante :

```
docker-compose up
```

*Nous pouvez ajouter l'option -d pour lancer les conteneurs en background.*

Nous obtiendrez ainsi le résultat suivant :

```
root@ubuntu:~/poc_monitoring/apache-nginx-grafana# docker-compose up -d
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the current node.

To deploy your application across the swarm, use the bundle feature of the Docker experimental build.

More info:
https://docs.docker.com/compose/bundles

nginx is up-to-date
Starting Nginx-exporter1
Starting prometheus
new-nginx is up-to-date
Starting Nginx-exporter2
Starting apache-exporter
Starting Serveur-apache
Starting node-exporter
Starting grafana
```

Pour lister tous les conteneurs déployés, utilisez la commande docker ps, comme suit :

```
root@ubuntu:~/poc_monitoring/apache-nginx-grafana# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
7443ff8d8af2        grafana/grafana:6.7.1   "/run.sh GF_INSTALL_..."   2 minutes ago      Up 34 seconds      0.0.0.0:3000->3000/tcp   grafana
e442fd84cc6e        httpd:2.4                 "httpd-foreground"    2 minutes ago      Up 34 seconds      0.0.0.0:8088->80/tcp    Serveur-apache
fbecfc8c34aa3       prom/node-exporter          "/bin/node_exporter"  2 minutes ago      Up 35 seconds      0.0.0.0:9100->9100/tcp   node-exporter
c061eb9eb747        bitnami/apache-exporter      "apache_exporter -sc..."  2 minutes ago      Up 33 seconds      0.0.0.0:9117->9117/tcp   apache-exporter
8a973d23243e        fish/nginx-exporter         "/usr/local/bin/nginx"  2 minutes ago      Up 32 seconds      0.0.0.0:9114->9113/tcp   Nginx-exporter2
2f40173536aa        nginx:1.17                "nginx -g 'daemon of..."  2 minutes ago      Up 36 seconds      0.0.0.0:8087->80/tcp    new-nginx
af12136b125a        prom/prometheus           "/bin/prometheus --c..."  2 minutes ago      Up 33 seconds      0.0.0.0:9090->9090/tcp   prometheus
1249b8caa65d        nginx:1.17                "nginx -g 'daemon of..."  2 minutes ago      Up 31 seconds      0.0.0.0:8000->80/tcp    nginx
b0b10b57e706        fish/nginx-exporter         "/usr/local/bin/nginx"  2 minutes ago      Up 35 seconds      0.0.0.0:9113->9113/tcp   Nginx-exporter1
root@ubuntu:~/poc_monitoring/apache-nginx-grafana# █
```

Afin d'afficher également les conteneurs qui ne sont pas en mode UP, veuillez rajouter l'option -a.

### 5.1.1 Troubleshooting

Dans cet exemple, nous allons simuler une erreur afin d'exécuter quelques commandes de dépannage.

Dans le cas où vous obtenez une erreur de syntaxe, il suffit de lancer la commande docker-compose config afin de l'identifier :

```
root@ubuntu:~/poc_monitoring/apache-nginx-grafana# docker-compose config
ERROR: yaml.parser.ParserError: while parsing a block mapping
  in "./docker-compose.yml", line 1, column 1
expected <block end>, but found '<block mapping start>'
  in "./docker-compose.yml", line 15, column 4
root@ubuntu:~/poc_monitoring/apache-nginx-grafana#
```

Dans le cas où vous obtenez une erreur sur l'une de vos applications, il suffit de lancer la commande docker-compose log afin d'afficher les logs de vos applications « :

```
root@ubuntu:~/poc_monitoring/apache-nginx-grafana# docker-compose logs
Attaching to grafana, node-exporter, Serveur-apache, new-nginx, apache-exporter, Nginx-exporter1, nginx, Nginx-exporter2, prometheus
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Starting Grafana" logger=server version=6.7.1 commit=ca6d08d5cb branch=HEAD
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config loaded from" logger=settings file=/usr/share/grafana/conf/default
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config loaded from" logger=settings file=/etc/grafana/grafana.ini
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config overridden from command line" logger=settings arg="default.paths.
data=/var/lib/grafana"
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config overridden from command line" logger=settings arg="default.paths.
logs=/var/log/grafana"
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config overridden from command line" logger=settings arg="default.paths.
plugins=/var/lib/grafana/plugins"
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config overridden from command line" logger=settings arg="default.paths.
provisioning=/etc/grafana/provisioning"
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config overridden from command line" logger=settings arg="default.log.mo
de=console"
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config overridden from Environment variable" logger=settings var="GF_PAT
HS_DATA=/var/lib/grafana"
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config overridden from Environment variable" logger=settings var="GF_PAT
HS_LOGS=/var/log/grafana"
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config overridden from Environment variable" logger=settings var="GF_PAT
HS_PLUGINS=/var/lib/grafana/plugins"
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config overridden from Environment variable" logger=settings var="GF_PAT
HS_PROVISIONING=/etc/grafana/provisioning"
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config overridden from Environment variable" logger=settings var="GF_SEC
URITY_ADMIN_USER=admin"
grafana    | t=2020-04-05T21:41:03+0000 lvl=info msg="Config overridden from Environment variable" logger=settings var="GF_SEC
URITY_ADMIN_PASSWORD=*****"
node-exporter | time="2020-04-05T21:41:05Z" level=info msg="Starting node_exporter (version=0.18.1, branch=HEAD, revision=3db7773
2e925c08f675d7404a8c46466b2ece83e)" source="node_exporter.go:156"
node-exporter | time="2020-04-05T21:41:05Z" level=info msg="Build context (go=gol.12.5, user=root@b50852a1acba, date=20190604-16:
41:18)" source="node_exporter.go:157"
node-exporter | time="2020-04-05T21:41:05Z" level=info msg="Enabled collectors:" source="node_exporter.go:97"
node-exporter | time="2020-04-05T21:41:05Z" level=info msg=" - arp" source="node_exporter.go:104"
node-exporter | time="2020-04-05T21:41:05Z" level=info msg=" - bcache" source="node_exporter.go:104"
node-exporter | time="2020-04-05T21:41:05Z" level=info msg=" - bonding" source="node_exporter.go:104"
```

### 5.1.2 Les volumes

Trois volumes ont été créés :

Emplacement local	Emplacement du conteneur	Usage
./grafana/grafana.db	/etc/grafana/grafana.ini	Configuration par défaut
./grafana/provisioning/dashboards	/etc/grafana/provisioning/dashboards	Conservation les tableaux de bord
./grafana/provisioning/datasources	/etc/grafana/provisioning/datasources	Conservation les sources de données

<code>./prometheus/prometheus.yml</code>	<code>/etc/prometheus/prometheus.yml</code>	Pour stocker les métriques collectées et configurer les targets
--	---	---

## 4.2 Connexion

### 5.2.1 Mappage des ports

Les ports par défaut de prometheus grafana sont respectivement :

- 9000:9090
- 3000:3000

Prometheus est accessible maintenant sur <http://<IP-SERNEUR>:9090/>

Grafana est accessible maintenant sur <http://<IP-SERNEUR>:3000/>

### 5.2.2 Gestion des mots de passe

#### 5.2.2.1 Grafana

Mot de passe *web* pour l'administration via l'interface Web :

- Login : **admin**
- Mot de passe : **admin**

### 5.2.3 Modification des mots de passes

Nous pouvez réinitialiser le mot de passe sur Grana de l'utilisateur soit

- Depuis le fichier de config en changeant la variable **password** soit dans :
  - Le fichier `/etc/grafana/grafana.ini` dans le conteneur
  - Le fichier `/var/lib/docker/volumes/grafana_grafana-config/_data/grafana.ini` dans votre machine local
  - Cliquer sur votre profil qui est disponible dans la barre de menus située dans le coin supérieur droit (avatar de l'utilisateur) > **Settings > Password**
- Depuis une commande curl :

```
curl -X PUT -H "Content-Type: application/json" -d '{
  "oldPassword": "admin",
  "newPassword": "newpass",
  "confirmNew": "newpass"
}' http://admin:admin@<your_grafana_host>:3000/api/user/password
```

- Depuis la cli avec la commande suivante (il faut être dans le conteneur) :

```
grafana-cli admin reset-admin-password
```

#### 5.2.4 Accès WEB

L'interface d'administration est disponible aux adresses suivantes (différentes suivant que l'interface est en **HTTP** ou **HTTPS**) :

	<b>HTTP</b>	<b>HTTPS</b>
<b>Interface d'administration Prometheus</b>	<a href="http://NOTRE_IP:9000">http://NOTRE_IP:9000</a>	<a href="https://NOTRE_IP:9090">https://NOTRE_IP:9090</a>
<b>Interface d'administration Grafana</b>	<a href="http://NOTRE_IP:3000">http://NOTRE_IP:3000</a>	<a href="https://NOTRE_IP:3000">https://NOTRE_IP:3000</a>

### 4.3 Configuration Prometheus

Pour bien collecter les métriques et les stocker sur le serveur Prometheus, il faut impérativement configurer le fichier « **prometheus.yml** ». Il permet également de configurer :

- Les alertes les intervalles « scrape »
- Le temps d'intervalle d'évaluation
- La configuration des targets « scrape »

```
# my global config
global:
  scrape_interval:    15s # Set the scrape interval to every 15 seconds. Default is every 1 minute
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
```

Ici le temps de `scrape_interval` est configuré à 15 secondes mais il peut bien être modifié selon le besoin et l'intervalle d'évaluation à 15 secondes également.

Ce fichier prend également en charge tous les exporter qui collectent les métriques. Dans notre cas nous avons rajouter un exporter pour grafana, apache docker et nginx :

```

Quick connect...
1. Home
3. 192.168.64.128 (lavilla)

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ['192.168.64.128:9090']

  - job_name: 'grafana'
    static_configs:
      - targets: ['192.168.64.128:3000']

  - job_name: 'nginx-exporter'
    static_configs:
      - targets: ['192.168.64.128:9113', '192.168.64.128:9114']

  - job_name: 'node-exporter'
    static_configs:
      - targets: ['192.168.64.128:9100']

  - job_name: 'apache-exporter'
    static_configs:
      - targets: ['192.168.64.128:9117']

  - job_name: 'docker'
    static_configs:
      - targets: ['192.168.64.128:9323']

```

Ensuite, il faut redémarrer le container prometheus avec la commande :

`docker restart prometheus`

Ou redémarrer le docker-compose avec la commande :

`docker-compose restart`

Prometheus est maintenant accessible sur l'adresse IP du serveur avec le port 9090 :

Prometheus collecte les données des systèmes monitorés en se connectant à des « endpoints » et

Élément	Valeur
pas de données	

il collecte ses propres métriques automatiquement. Un système de labels ou cibles avec la nouvelle version permet une identification fine des métriques.

[Non sécurisé | 192.168.64.128:9090/targets](#)

Prométhée Alertes Graphique Statut ▾ Aidez-moi

Endpoint	Etat	Étiquettes	Dernière éraflure	Durée de raclage	Erreur
http://192.168.64.128:3000/metrics	EN HAUT	instance = "192.168.64.128:3000" job = "grafana"	Il y a 2.619s	2,336 ms	

**exportateur nginx (2/2 up)** [Montrer moins](#)

Endpoint	Etat	Étiquettes	Dernière éraflure	Durée de raclage	Erreur
http://192.168.64.128:9113/metrics	EN HAUT	instance = "192.168.64.128:9113" job = "nginx-exporter"	Il y a 3.128s	1,945 ms	
http://192.168.64.128:9114/metrics	EN HAUT	instance = "192.168.64.128:9114" job = "nginx-exporter"	Il y a 4.09s	4.634ms	

**exportateur de noeuds (1/1 et plus)** [Montrer moins](#)

Endpoint	Etat	Étiquettes	Dernière éraflure	Durée de raclage	Erreur
http://192.168.64.128:9100/metrics	EN HAUT	instance = "192.168.64.128:9100" job = "node-exporter"	Il y a 14.767s	10,59 ms	

Sur la page d'accueil on retrouve notamment la configuration fournie et on peut bien tester la validité des sources monitorées. Il est également possible de les visualiser via la console ou le moteur de graphe de Prometheus (menu Graph).

## □ Statut-->configuration

[Non sécurisé | 192.168.64.128:9090/config](#)

Prométhée Alertes Graphique Statut ▾ Aidez-moi

**Configuration** [Copier dans le presse-papier](#)

```

global:
  scrape_interval: 15s
  scrape_timeout: 10s
  evaluation_interval: 15s
alerte:
  gestionnaires d'alerte:
    - static_configs:
        - cibles: []
          schéma: http
          délai: 10 s
          api_version: v1
scrape_configs:
  - nom_travail: prometheus
    honor_timestamps: true
    scrape_interval: 15s
    scrape_timeout: 10s
    metrics_path: / metrics
    schéma: http
    static_configs:
      - cibles:
          - 192.168.64.128:9090
      - nom_travail: grafana
        honor_timestamps: true
        scrape_interval: 15s
        scrape_timeout: 10s
        metrics_path: / metrics
        schéma: http
        static_configs:
          - cibles:
              - 192.168.64.128:9090

```

Le serveur Prometheus fournit son propre moteur de requêtes PromQL. Il permet également de fournir, à partir de métriques Prometheus, différents types de graphiques paramétrables selon les requêtes et les besoins. Cette image montre bien l'instance de Nginx.

Exemple de requête PromQL :

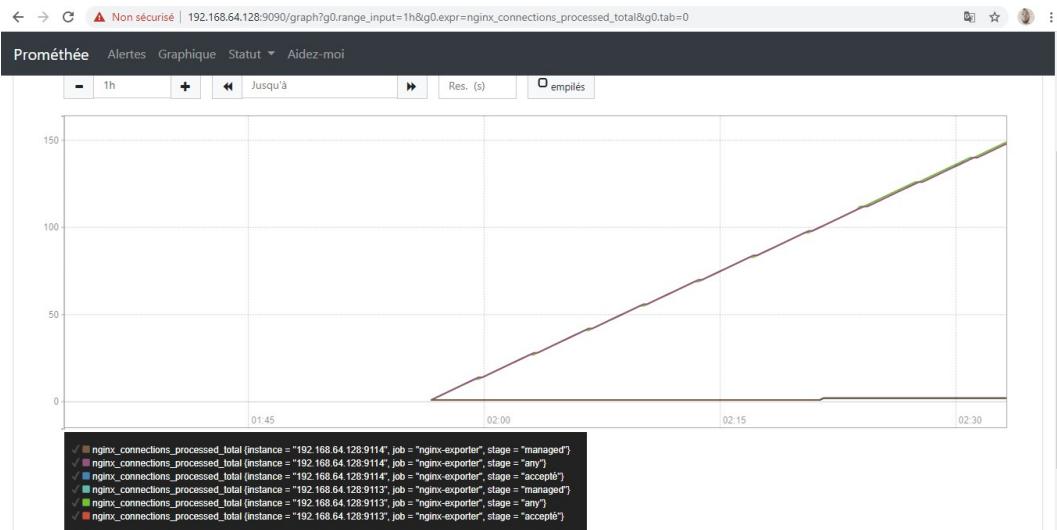
`nginx_connections_processed_total`

The screenshot shows the Prometheus web interface with the URL `192.168.64.128:9090/graph?g0.range_input=1h&g0.expr=nginx_connections_processed_total&g0.tab=1`. The interface includes a navigation bar with links for Non sécurisé, Prométhée, Alertes, Graphique, Statut, and Aidez-moi. Below the search bar, there is a button labeled "Exécuter" (Execute) and a "Graphique" tab selected. The results table shows the following data:

Élément	Valeur
nginx_connections_processed_total {instance = "192.168.64.128:9114", job = "nginx-exporter", stage = "accepté"}	2
nginx_connections_processed_total {instance = "192.168.64.128:9114", job = "nginx-exporter", stage = "any"}	136
nginx_connections_processed_total {instance = "192.168.64.128:9114", job = "nginx-exporter", stage = "managed"}	2
nginx_connections_processed_total {instance = "192.168.64.128:9113", job = "nginx-exporter", stage = "accepté"}	2
nginx_connections_processed_total {instance = "192.168.64.128:9113", job = "nginx-exporter", stage = "any"}	137
nginx_connections_processed_total {instance = "192.168.64.128:9113", job = "nginx-exporter", stage = "managed"}	2

At the bottom right, there is a link "Supprimer le graphique".

□ Nu sa performance, il est également capable de zoomer le tout en faisant un clic sur graphe :



Les différentes métriques récupérées par Prometheus sont accessibles à leurs url respectives, associées à l'adresse IP du serveur Prometheus ainsi que le port attribué.

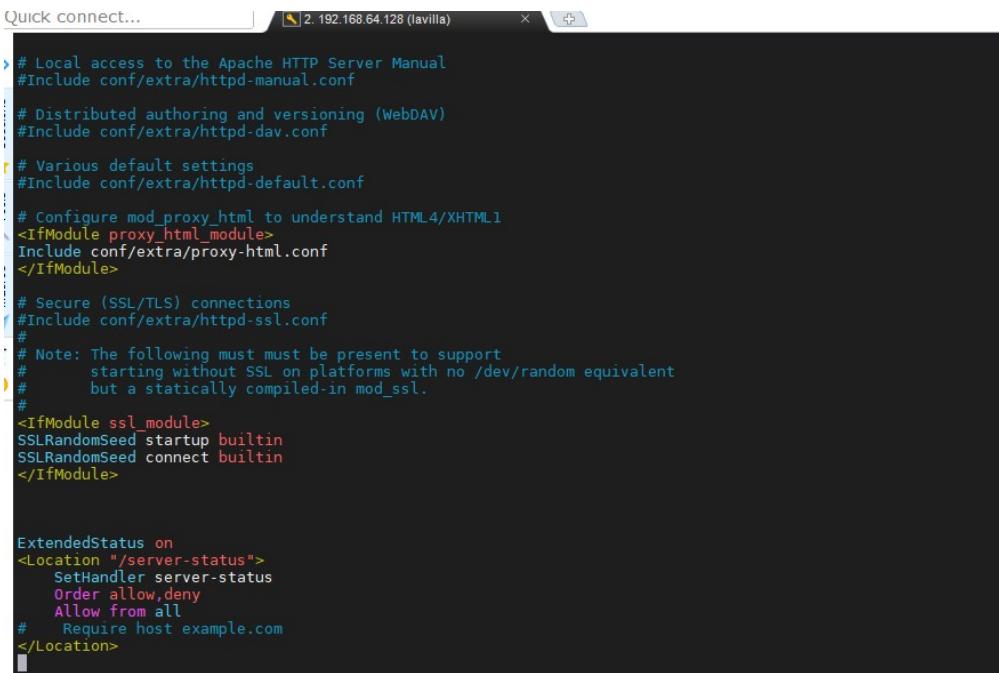
### 5.3.1 Les métriques du serveur Apache

Pour pouvoir collecter les métriques du serveur Apache, il est nécessaire de configurer l'exporter qui scrape sur l'IP de la machine hôte et du port avec la commande ci-dessous :

```
- "-nginx.scrape uri=http://192.168.64.128:8088/status"
```

Le fichier de `httpd.conf` permet de configurer le statut du serveur. Cela permet de collecter les métriques et mettre à la disposition de Prometheus et ces métriques sont accessibles à travers l'url :

`http://192.168.64.128:8088/server-status`



```
> # Local access to the Apache HTTP Server Manual
#Include conf/extra/httpd-manual.conf

# Distributed authoring and versioning (WebDAV)
#Include conf/extra/httpd-dav.conf

# Various default settings
#Include conf/extra/httpd-default.conf

# Configure mod_proxy_html to understand HTML4/XHTML1
<IfModule proxy_html module>
Include conf/extra/proxy-html.conf
</IfModule>

# Secure (SSL/TLS) connections
#Include conf/extra/httpd-ssl.conf
#
# Note: The following must be present to support
#       starting without SSL on platforms with no /dev/random equivalent
#       but a statically compiled-in mod_ssl.
#
<IfModule ssl_module>
SSLRandomSeed startup builtin
SSLRandomSeed connect builtin
</IfModule>

ExtendedStatus on
<Location "/server-status">
    SetHandler server-status
    Order allow,deny
    Allow from all
    # Require host example.com
</Location>
```

Après cette configuration, il faut impérativement redémarrer le container avec la commande suivante pour prendre en compte ladite configuration.

```
docker-compose restart
```

Ensuite le statut peut être affiché dans l'url avec le résultat suivant :

`http://192.168.64.128:8088/server-status`

## État du serveur Apache pour 192.168.64.128 (via 192.168.112.10)

Version du serveur: Apache / 2.4.43 (Unix)

Serveur MPM: événement

Serveur construit: 31 mars 2020 03:45:22

Heure actuelle: samedi, 04-avr-2020 22:18:10 UTC

Heure de redémarrage: samedi 04 avril 2020 21:57:56 UTC

Configuration du serveur parent. Génération: 1

Génération MPM du serveur parent: 0

Temps de disponibilité du serveur: 20 minutes 14 secondes

Charge du serveur: 0,00 0,06 0,07

Accès totaux: 81 - Trafic total: 91 kB - Durée totale: 36

Utilisation du processeur: u.24 s.09 cu0 cs0 - 0,0272% de charge du processeur

.0667 requêtes / sec - 76 B / seconde - 1150 B / requête - .444444 ms / requête

1 demandes en cours de traitement, 74 travailleurs inactifs

Fente	PID	Arrêt	Connexions		Fils		Connexions asynchrones	
			total	acceptant	occupé	tourner au ralenti	l'écriture	rester en vie
0	sept	non	1	Oui	1	24	0	0
1	8	non	0	Oui	0	25	0	0
2	9	non	1	Oui	0	25	0	0
Somme	3		3		1	74	0	0

.....  
.....  
.....  
.....  
.....

Clé du tableau de bord:

"\_ " En attente de connexion ", s" Démarrage "," rDemande de lecture

### 5.3.1.1 Connection sur le serveur Apache

Après toutes configurations c'est-à-dire les fichiers :

- httpd.conf
- htdocs

### 5.3.2 Les métriques du serveur Nginx

L'exporter Nginx procède également de la même manière que celui d'Apache mais les fichiers de configurations ne sont pas les mêmes. Donc pour Nginx, l'exporter scrape sur :

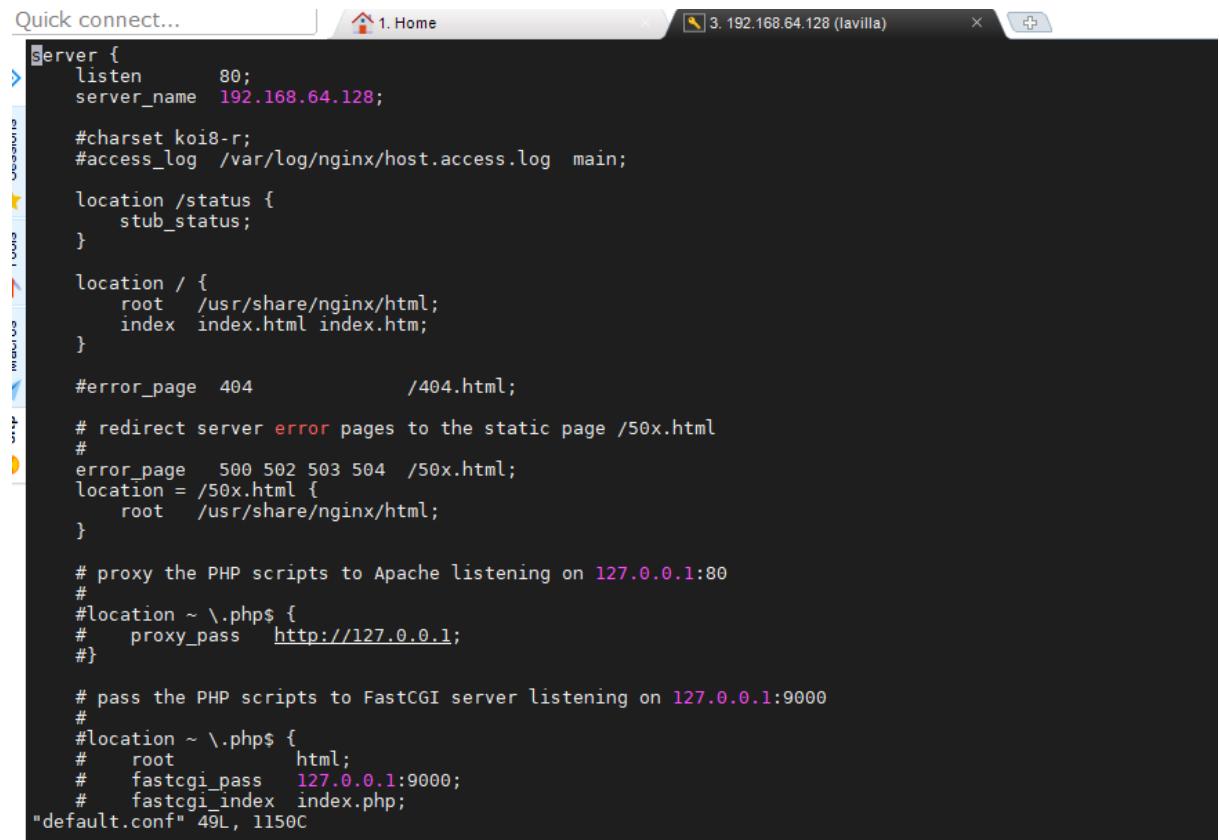
`-nginx.scrape uri=http://192.168.64.128:8000/status"`

Ce qui lui permet d'aller sur le serveur Apache collecter les métriques et mettre à la disposition du serveur Prometheus.

Le serveur Nginx dispose deux fichiers dont il faut configurer à l'occurrence de :

- Default.conf
- html

Le fichier de default.conf permet de configurer le statut du serveur Nginx. Cela va permettre de collecter les métriques et mettre à la disposition de Prometheus et ces métriques sont accessibles à travers l'url : <https://192.168.64.128:8000/status> dont le contenu du fichier permet de choisir l'adresse IP et d'exposer le port 80.



The screenshot shows a terminal window with the title "Quick connect...". It contains the Nginx configuration file "default.conf". The file defines a single server block listening on port 80. It includes basic logging, error handling (404 and 500-504 errors), and PHP proxying via FastCGI. The configuration is well-structured with comments and standard Nginx syntax.

```
server {
    listen      80;
    server_name 192.168.64.128;

    #charset koi8-r;
    #access_log  /var/log/nginx/host.access.log  main;

    location /status {
        stub_status;
    }

    location / {
        root   /usr/share/nginx/html;
        index index.html index.htm;
    }

    #error_page  404          /404.html;

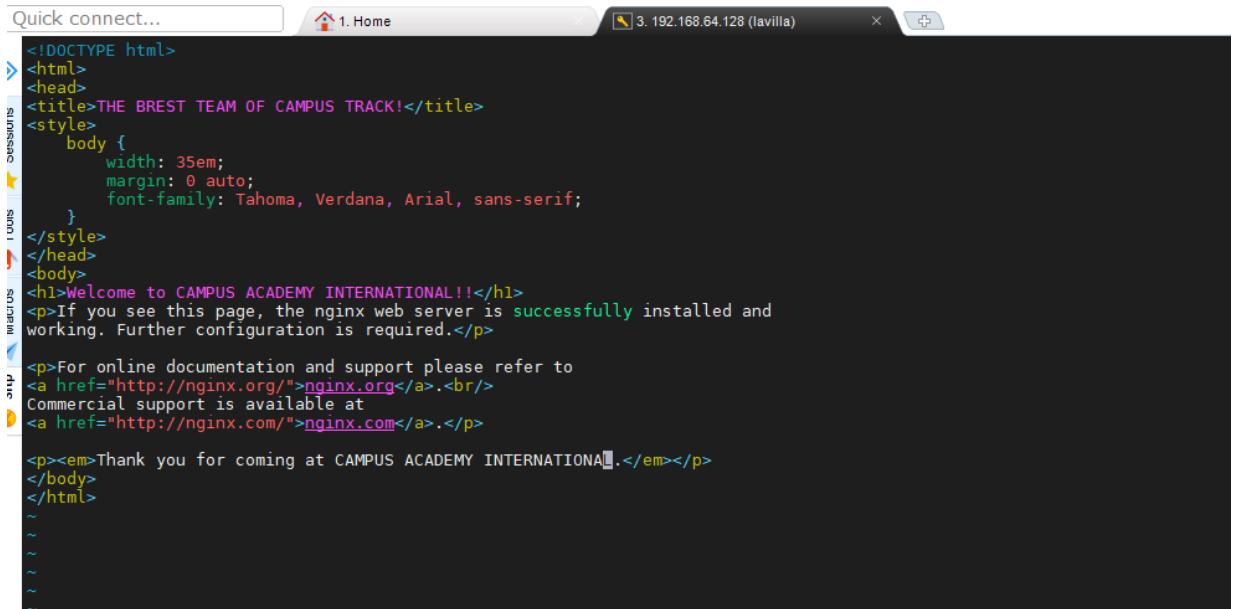
    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass   http://127.0.0.1;
    #

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root           html;
    #    fastcgi_pass  127.0.0.1:9000;
    #    fastcgi_index index.php;
    #}
}

"default.conf" 49L, 1150C
```

Ensuite, le fichier html celui qui permet de faire l'affichage de la page web aux utilisateurs (clients). C'est un fichier qui peut être modifié selon les besoins en affichage avec des messages spécifiques comme celui de notre solution Automatisator.



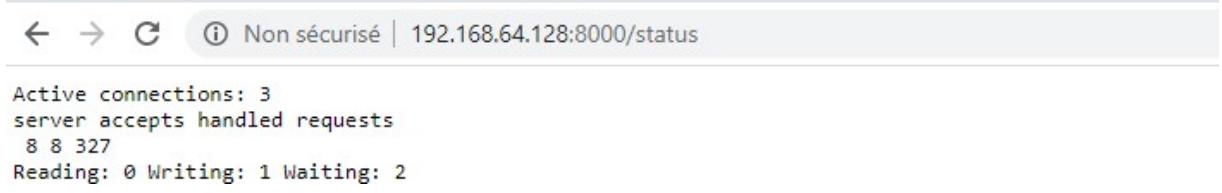
The screenshot shows a terminal window with the title "1. Home". The content of the terminal is the Nginx configuration file, which includes HTML code for a welcome page. The code defines a body with a width of 35em, centered margin, and a sans-serif font-family. It features a heading "Welcome to CAMPUS ACADEMY INTERNATIONAL!!" and several paragraphs providing links to nginx.org and nginx.com, along with a note of thanks.

```
<!DOCTYPE html>
<html>
<head>
<title>THE BREST TEAM OF CAMPUS TRACK!</title>
<style>
body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
}
</style>
</head>
<body>
<h1>Welcome to CAMPUS ACADEMY INTERNATIONAL!!</h1>
<p>If you see this page, the nginx web server is successfully installed and working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org">nginx.org</a>. <br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for coming at CAMPUS ACADEMY INTERNATIONAL!</em></p>
</body>
</html>
~
```

Exemple :

Après toutes ces configurations, il est nécessaire de redémarrer le docker-compose avec la commande précédente et les configurations sont prises en compte. On peut donc afficher le statut du serveur Nginx à travers l'adresse IP de la machine hôte, suivi du port attribué ainsi que son statut comme configuré dans le fichier. Dans notre cas :

<https://192.168.64.128:8000/status>



The screenshot shows a browser window with the URL "192.168.64.128:8000/status". The page displays the Nginx status information, including active connections (3), handled requests (8 8 327), and current activity (Reading: 0 Writing: 1 Waiting: 2).

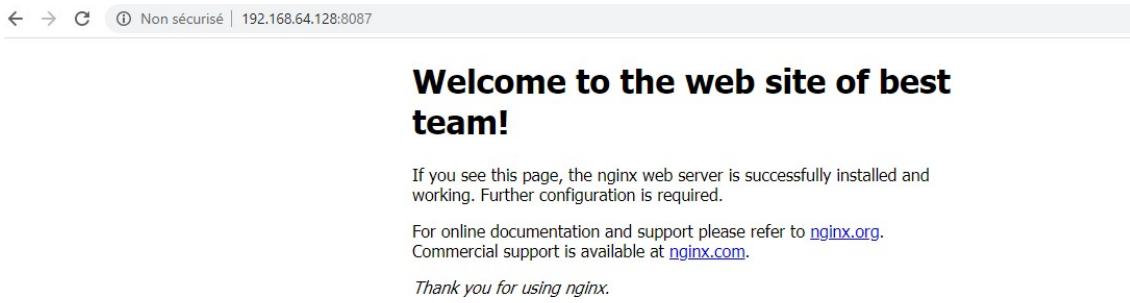
```
Active connections: 3
server accepts handled requests
 8 8 327
Reading: 0 Writing: 1 Waiting: 2
```

### 5.3.2.1 Connexion sur la page web Nginx

Pour la connexion sur la page web su serveur Nginx après toutes les configurations, il faut simple introduire dans l'url l'adresse IP de la machine hôte suivi du port attribuer.

Dans notre cas, l'adresse IP de la machine hôte est **192.168.64.128** et le serveur Nginx par exemple est accessible sur le port **8087** donc il sera :

<https://192.168.64.128:8087>

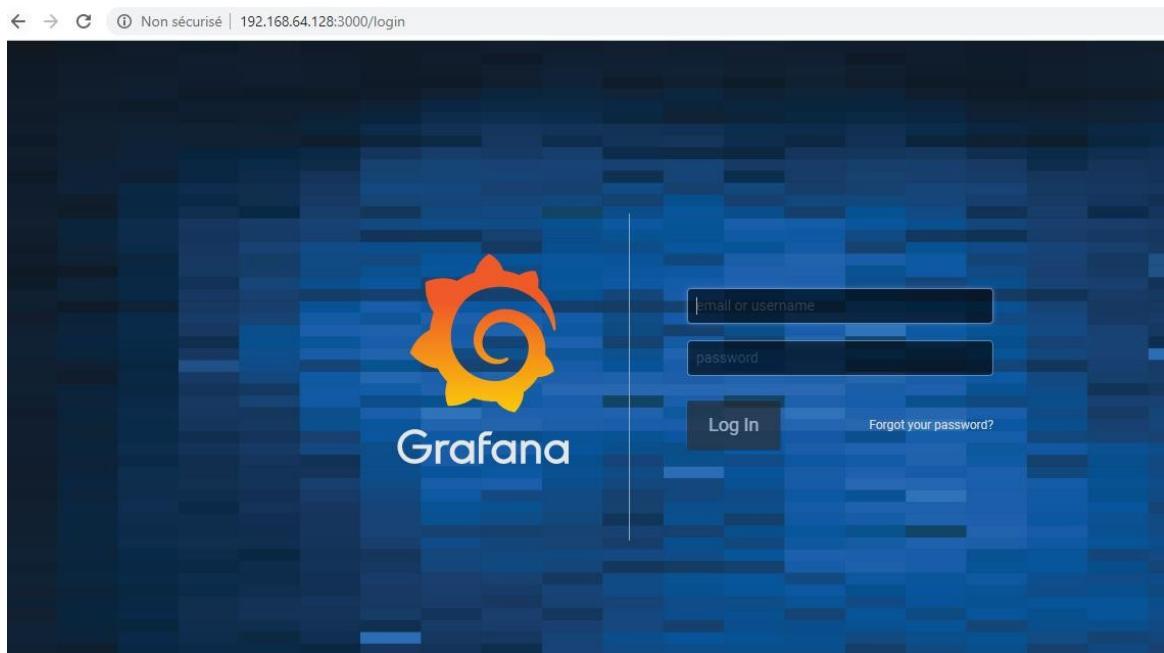


## 4.4 Outil visualisation : Grafana

Par défaut, l'interface web de Grafana est publiée sur le port **3000** mais il reste changeable dans la configuration selon les disponibilités des ports sur le serveur.

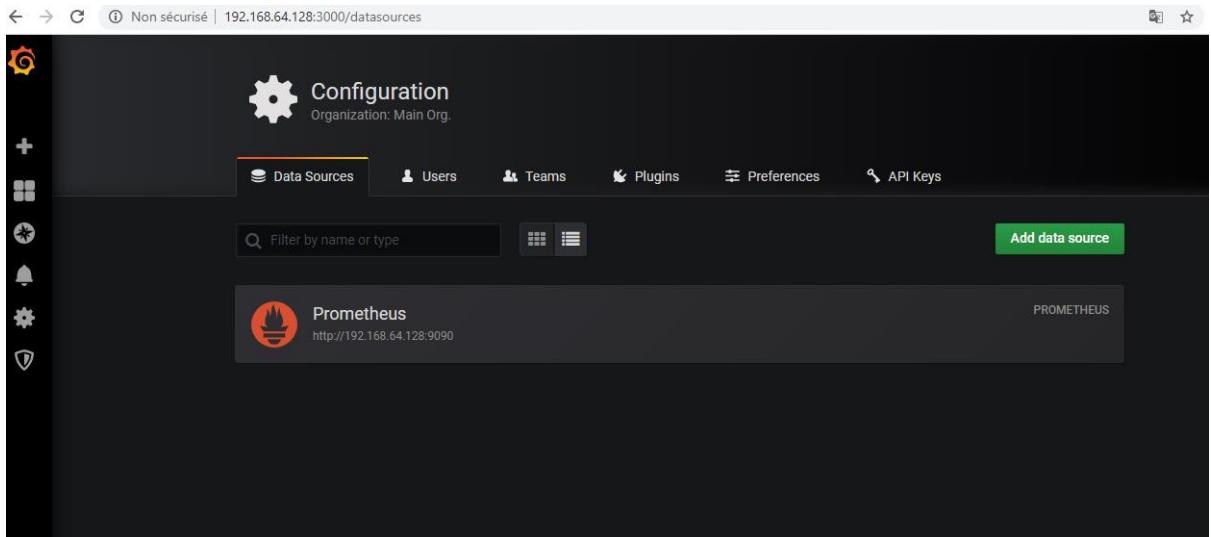
L'interface de Grafana permet d'accéder via l'adresse IP du serveur et le port attribué pour confirmer que ça fonctionne correctement.

Le nom d'utilisateur et le mot de passe dans le docker compose sont « **admin** » et « **root19** ».

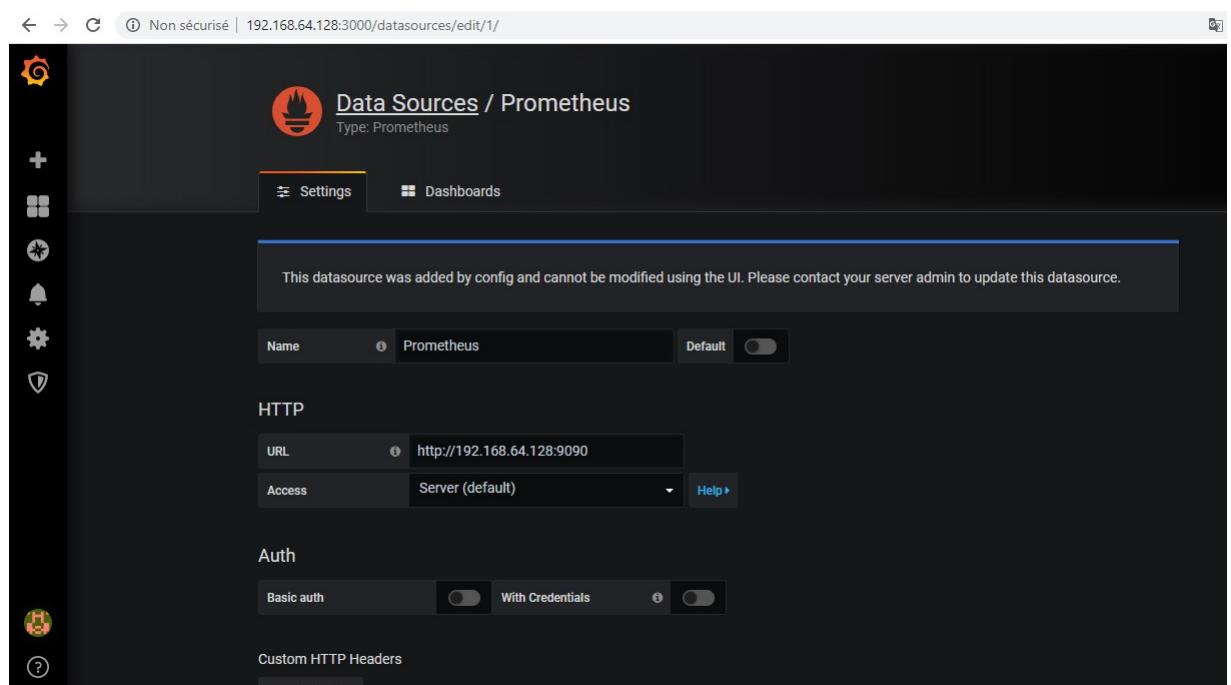


Après déploiement et connexion sur Grafana, il faut donc configurer la **datasource Prometheus** dans Grafana, et la configurer par « par défaut » pour qu'il soit directement connecté avec le serveur Prometheus.

- Pour cela on clique sur Prometheus pour l'ajouter

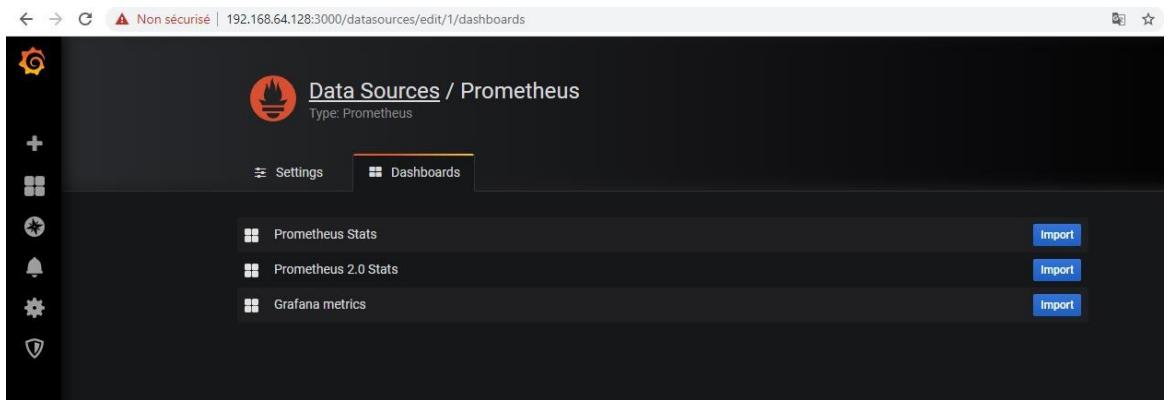


En cliquant sur Prometheus comme il se voit sur l'image, on ajoute Prometheus dans Grafana.



On peut ensuite, importer les différentes métriques pour qu'ils soient accessibles à l'affichage dans les Dashboards et créer des utilisateurs.

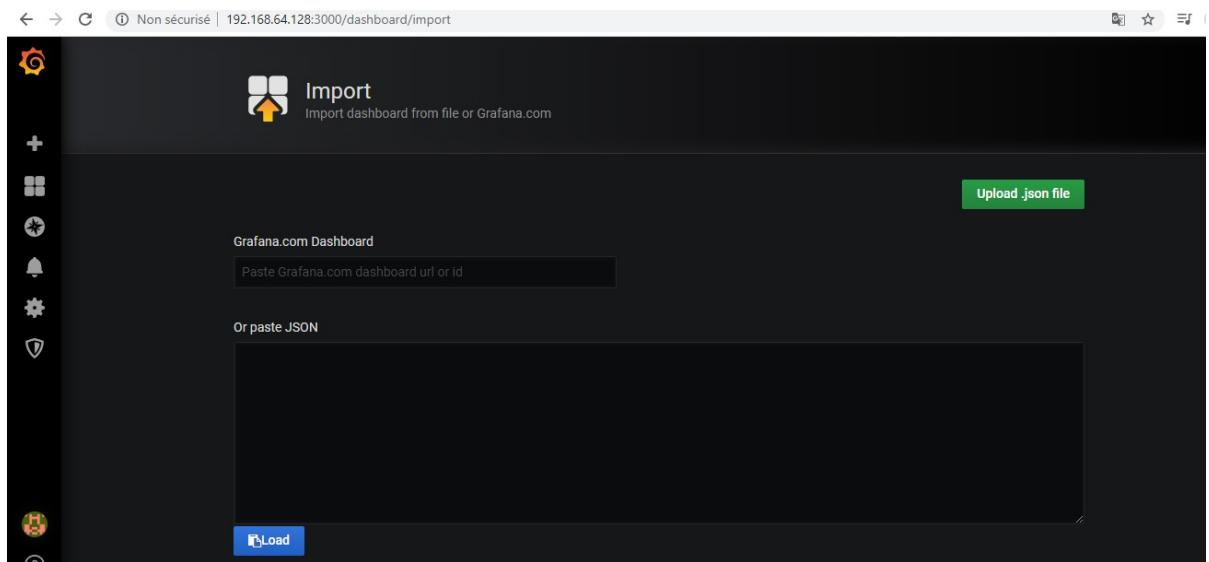
- En cliquant sur import, on importe les Dashboard par défaut de Prometheus et Grafana



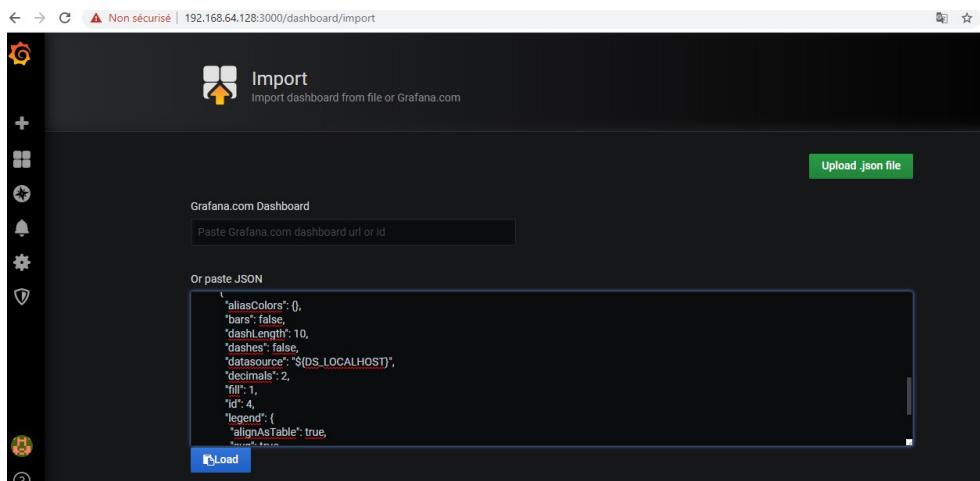
A présent place à la création du Dashboard. La communauté Grafana est très active et propose en téléchargement des Dashboard dont beaucoup se basent sur des données Prometheus. Partir d'un Dashboard existant est un moyen très rapide pour initier des écrans spécifiques à des composants standards à travers des fichiers **json**.

#### Dashboard Grafana/Serveur Apache

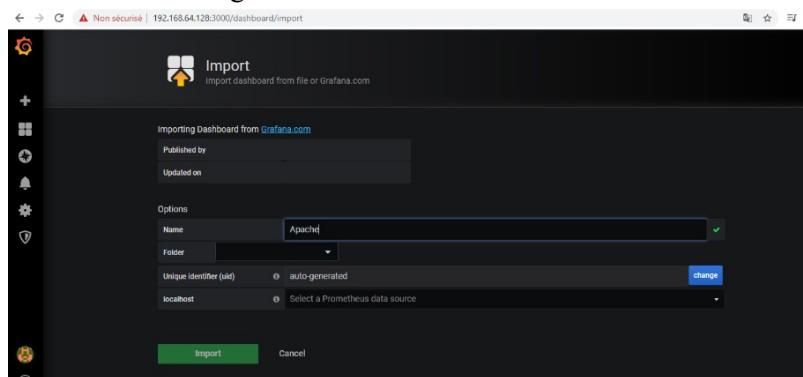
Dans cet exemple nous allons importer un fichier JSON pour créer un Dashbord Apache, pour cela ont clic sur le signe +, ensuite import



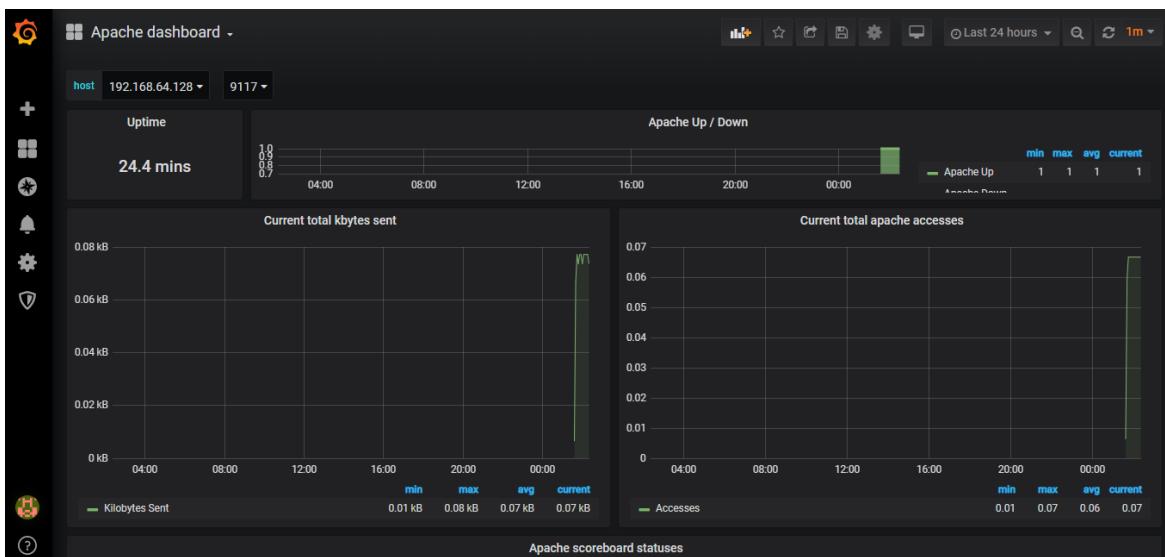
- On colle le fichier



- Ensuite il faut charger le fichier



- Afin on donne un Nom au Dashbord et on charge les métriques

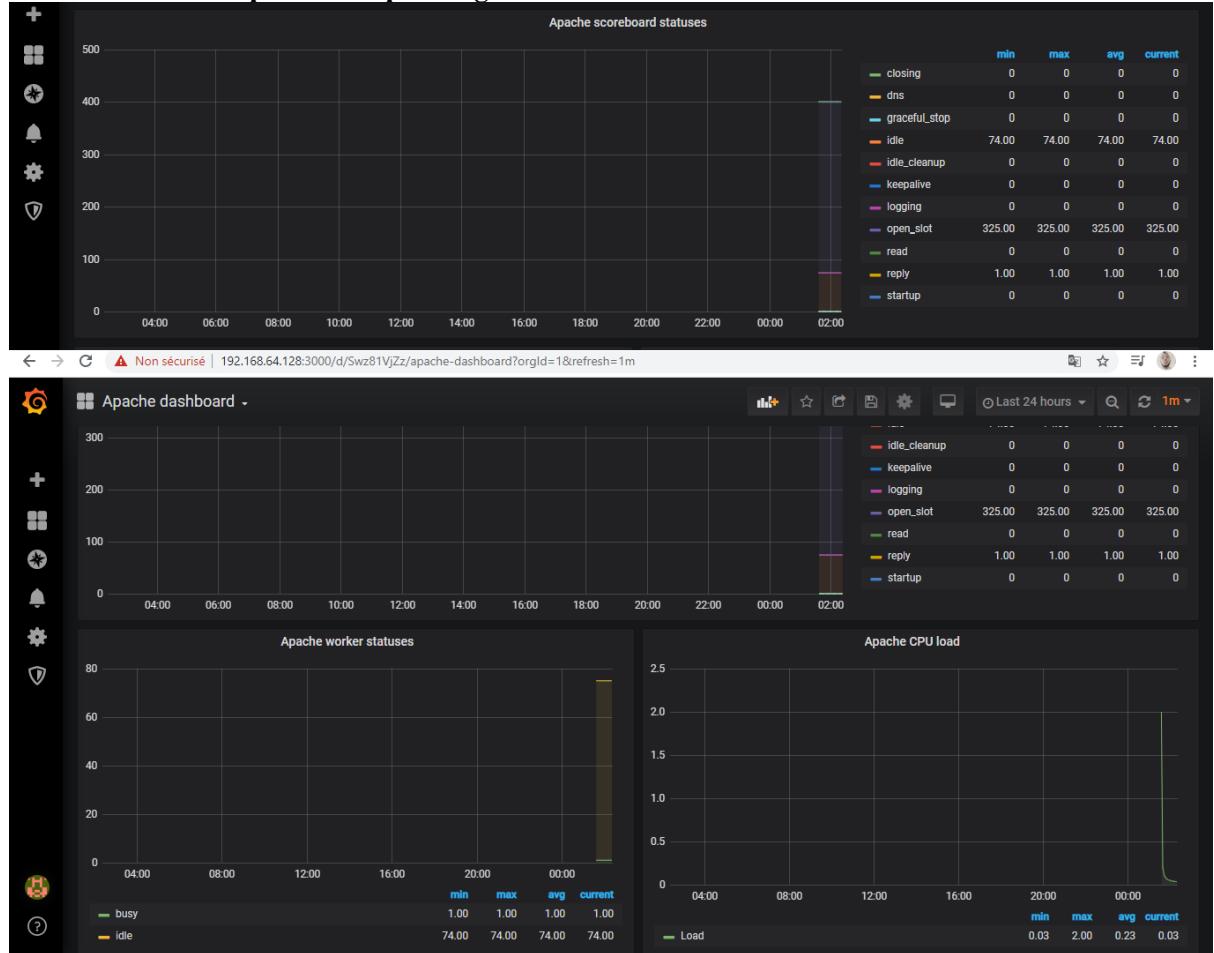


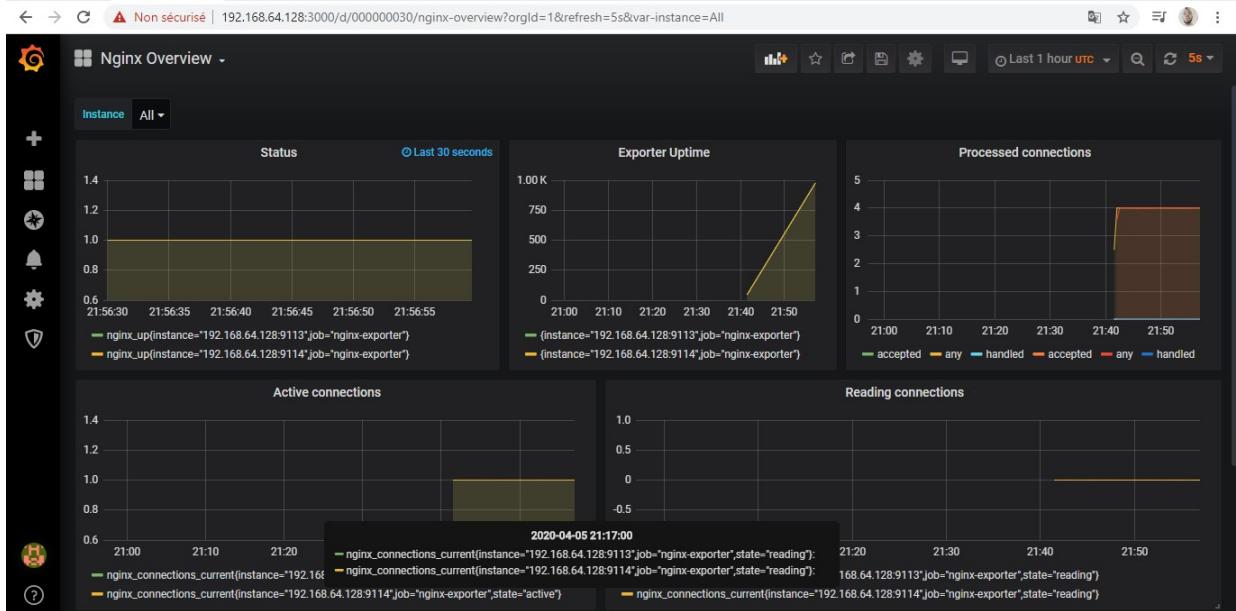
On voit bien que Grafana affiche les métriques suivantes :

- Kilobytes sent** : le nombre d'information transférée

- Uptime** : le temps fonctionnement depuis son démarrage
- Apache UP/Down** : l'état du serveur depuis son démarrage
- Current Total Accesses** : le nombre d'accès
- Apache Scoreboard statuses** : les connexions entrantes, en attente et complète
- Apache Worker statuses** : le nombre d'utilisateur actif et inactif sur le serveur
- Apache CPU Load** : l'utilisation du CPU du serveur

NB : c'est la même procédure pour Nginx



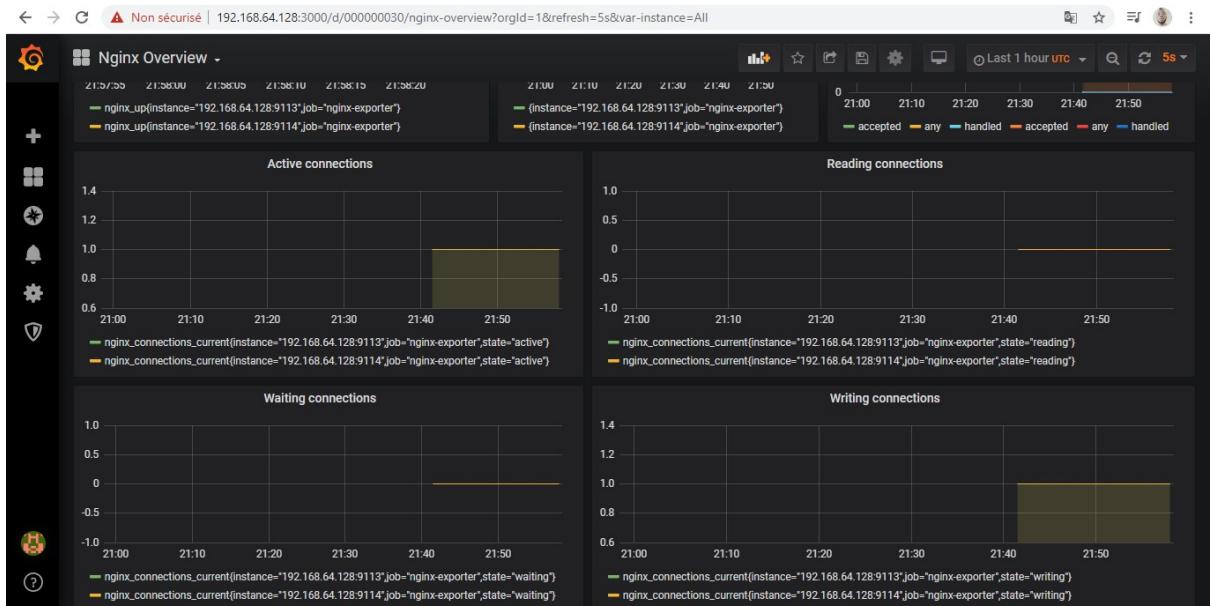


#### 5.4.1.1 Nginx Dashboard

Il est également possible d'utiliser des requêtes PromQL de Prometheus pour pouvoir créer des panels dans Grafana (Memory par exemple).

## 4.5 Crédation de panel docker

La création des panels se fait avec quel serveur ou métriques collectées et stocké sur Prometheus. Il nous sert de création des Dashboards spécifiques à travers des requêtes PromQL.



Exemple :

Nous effectuons une requête PromQL de Prometheus sur le daemon docker pour créer le panel Grafana avec la commande suivante :

```
engine_daemon_engine_cpus cpus
```

The screenshot shows the Prometheus web interface at <http://192.168.64.128:9090/graph>. The URL bar indicates it's an unsecured connection. The top navigation bar includes links for Prometheus, Alerts, Graph, Status, and Help. A message at the top right says "Try experimental React UI". Below the navigation, there's a search bar with the query "engine\_daemon\_engine\_cpus cpus". An "Execute" button is highlighted in blue. To its right is a dropdown menu with the placeholder "- insert metric at cursor -". On the far right, performance metrics are displayed: Load time: 29ms, Resolution: 14s, and Total time series: 1. Below the search bar, there are two tabs: "Graph" (selected) and "Console". The main area shows a single data point: "Element engine\_daemon\_engine\_cpus\_cpus(instance=\"192.168.64.128:9323\",job=\"docker\") Value 1". At the bottom left is a "Remove Graph" link, and at the bottom center is a "Add Graph" button.

Il affiche bien le résultat en cliquant sur « Execute » et il donne la possibilité de l'afficher en **console** tout comme en **graphe**.

Cette commande va être exécutée Grafana en procédant comme suit :

- En cliquant toujours dans Grafana sur le +, ensuite **create**

<http://192.168.64.128:3000/dashboard/new?orgId=1>

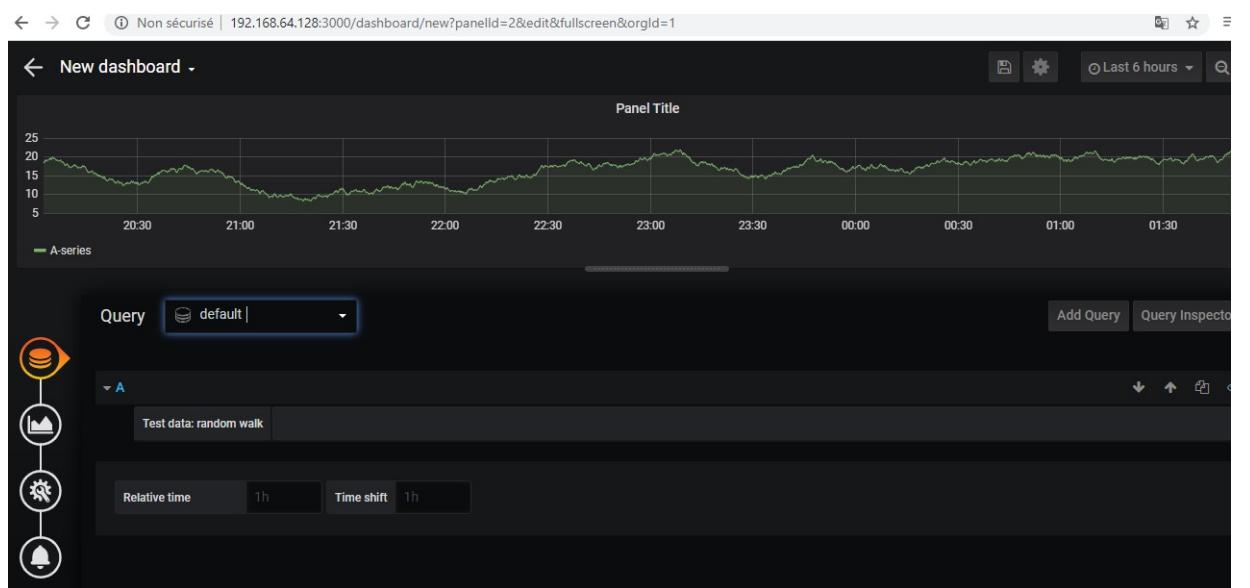
The screenshot shows the Grafana dashboard creation interface. On the left, there's a sidebar with icons for New dashboard, New Panel, Add Query, Choose Visualization, Convert to row, and other dashboard management options. The main area is titled "New Panel" and contains two buttons: "Add Query" and "Choose Visualization". A "Convert to row" button is located below these buttons. The overall interface is dark-themed.

- Add Query

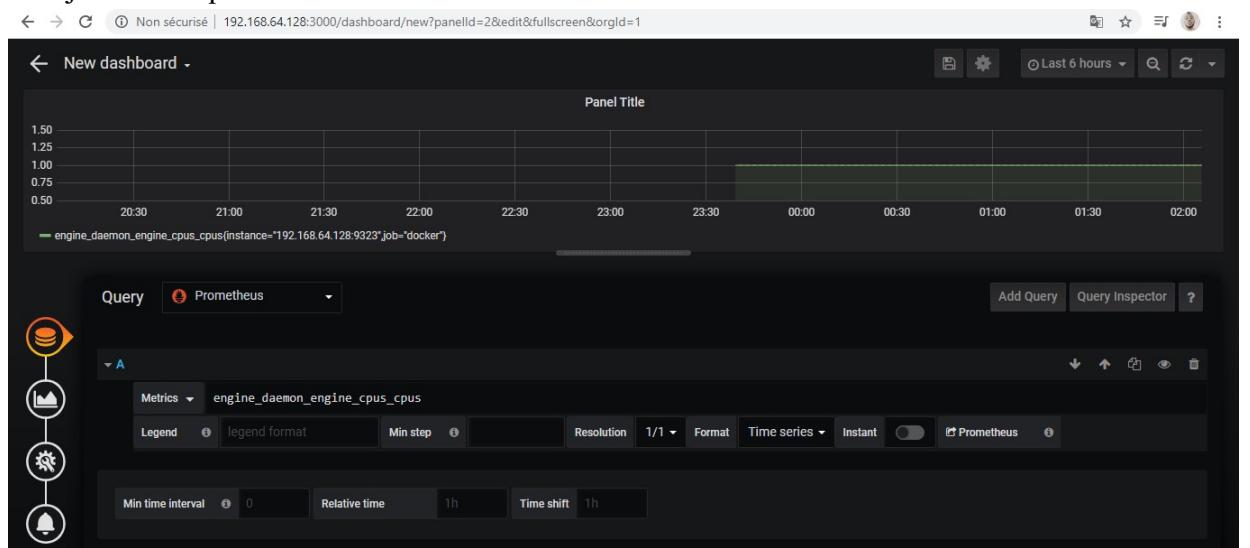
On ajoute la requête PromQL suite :

```
engine_daemon_engine_cpus cpus
```

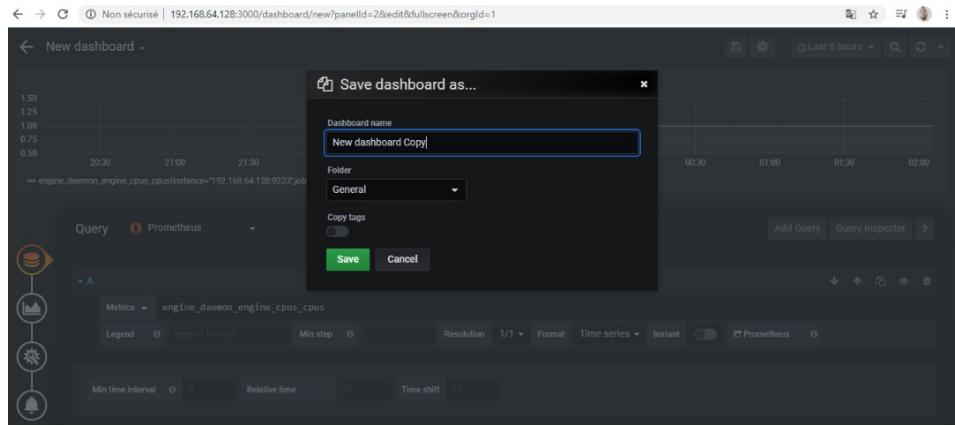
- Choisir le détalesource « default » qui correspond à Prometheus



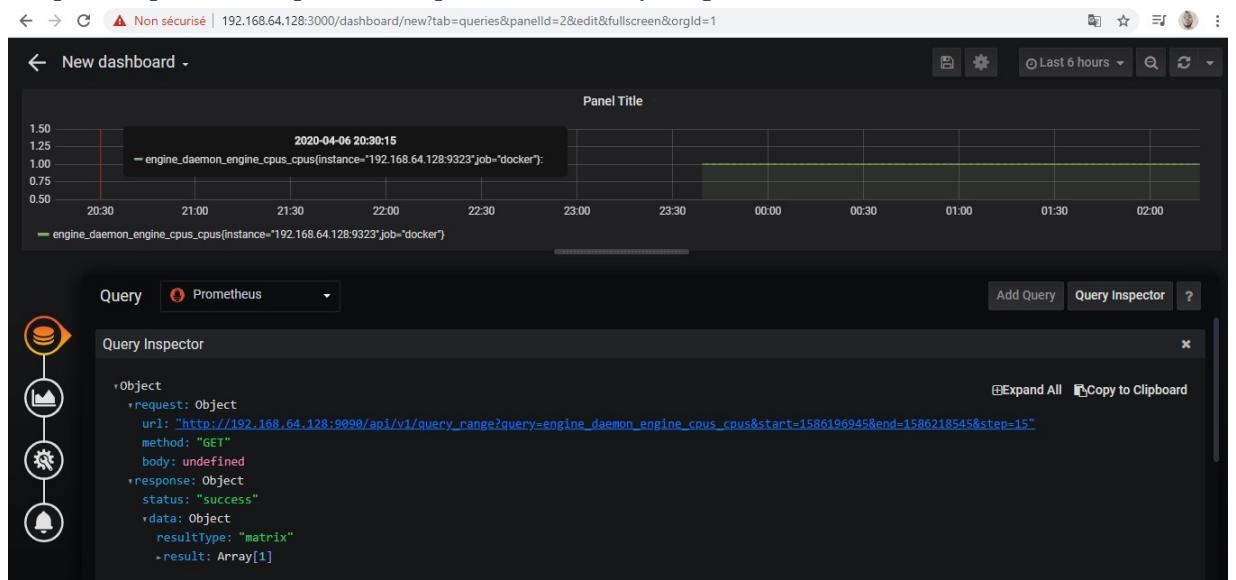
□ On ajouter la requête



□ Le panel est créé, il donc donnée un nom et enregistrer

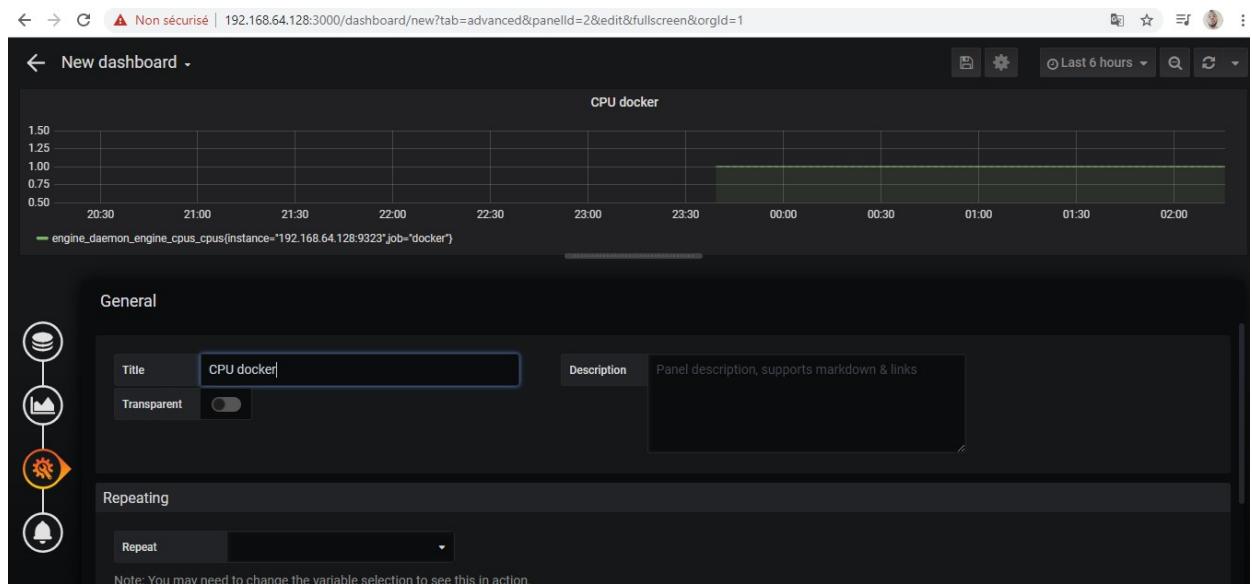


- On peut inspecter la requête en cliquant sur « **Query Inspector** »

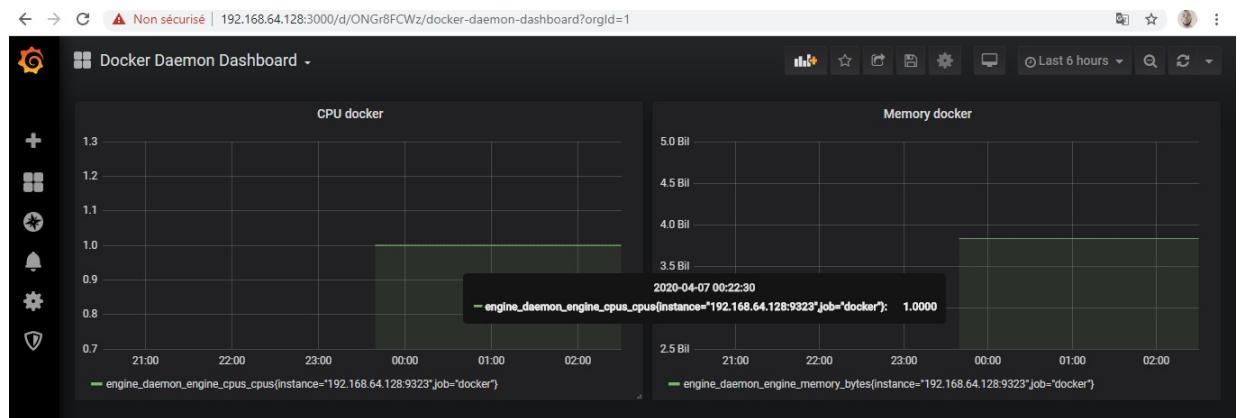


Cela permet de voir s'il existe des erreurs dans la requête ou pas.

- Nommer le panel



Enfin nous avons notre panel de dashboard créé. Mais ce qu'il faut noter est que, ce sont plusieurs panels qui donnent un Dashboard.



En résumé, L'intérêt du couple Prometheus / Grafana est de donner de la visibilité aux administrateurs systèmes, d'avoir des informations sur l'état de leurs serveurs. Les indicateurs CPU, RAM, disque vont permettre de gérer la capacité planifiée et d'anticiper les incidents relatifs à la plateforme.

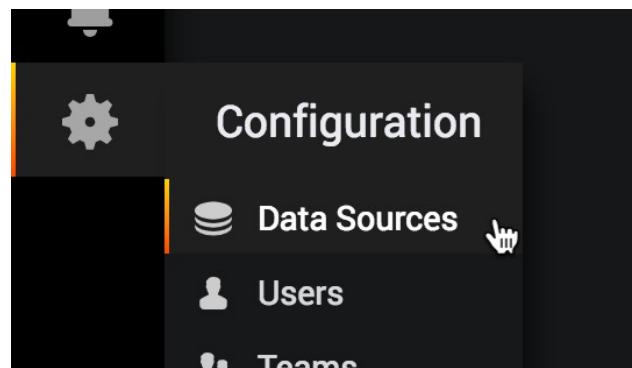
## 4.6 Métriques Hosts

Ici nous vous proposons une autre solution pour montrer les métriques de vos machines hôtes, en utilisant cette fois-ci influxdb.

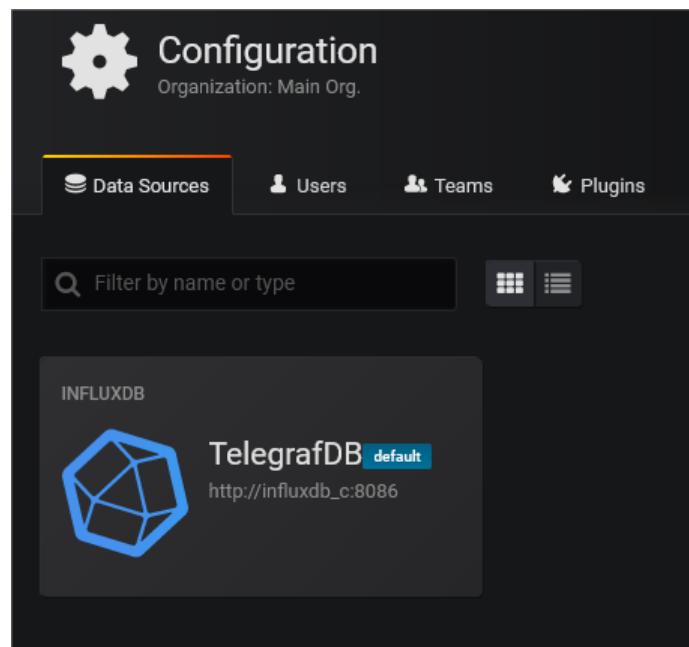
### 5.6.1 Ajouter une source de données (Data sources)

Avant de créer vos tableau de bord, vous devez ajouter votre source de données.

Déplacez d'abord votre curseur sur le menu latéral qui vous montrera le menu de configuration. Si le menu latéral n'est pas visible, cliquez sur l'icône Grafana dans le coin supérieur gauche. Le premier élément du menu de configuration concerne les sources de données. Cliquez dessus pour accéder à la page des sources de données où vous pouvez ajouter et modifier des sources de données.



Une source de données de type influxdb a été créé pour les besoins du projet nommé TelegrafDB



## 5.6.2 Tableau de bord (Dashboard)

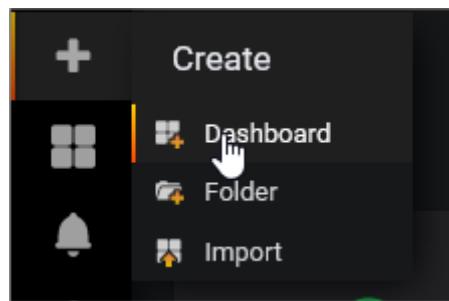
### 5.6.2.1 Description

Les tableaux de bord sont composés de panneaux individuels disposés sur une grille. Grafana est livré avec une variété de panneaux.

Grafana facilite la construction des bonnes requêtes et la personnalisation des propriétés d'affichage afin que vous puissiez créer le tableau de bord parfait pour vos besoins. Chaque Panel peut interagir avec les données de toutes les sources de données Grafana configurées (actuellement Graphite, Prometheus, Elasticsearch, InfluxDB, OpenTSDB, MySQL, PostgreSQL, Microsoft SQL Server et AWS Cloudwatch).

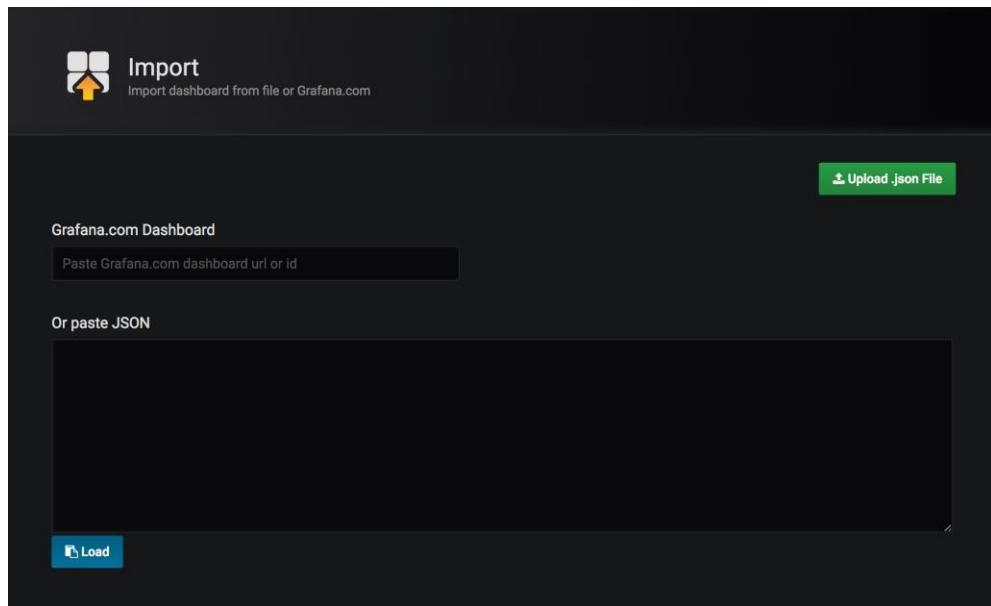
#### 5.6.2.2   Créer un dashboard

Cliquez sur le signe + dans le menu à gauche puis sur Dashboard

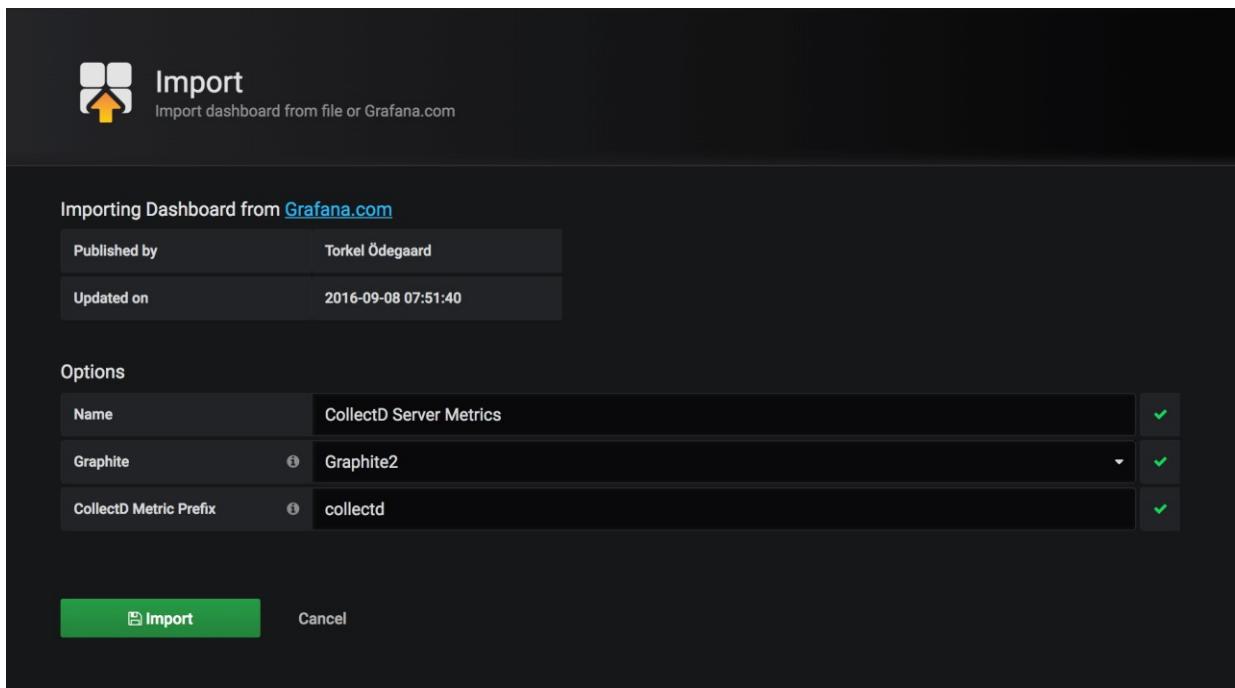


#### 5.6.2.3    Importer un dashboard

Pour importer un tableau de bord, ouvrez la recherche dans le tableau de bord, puis cliquez sur le bouton Importer.



À partir de là, vous pouvez télécharger un fichier json de tableau de bord, coller une URL de tableau de bord Grafana.com ou coller du texte json de tableau de bord directement dans la zone de texte.



#### 5.6.2.3.1 Les dashboards réalisés

L'équipe Automatisor a réalisé pour vous deux dashboards prêtes à l'emploi nommés respectivement :

- Dashboard -Windows
- Dashboard – Linux

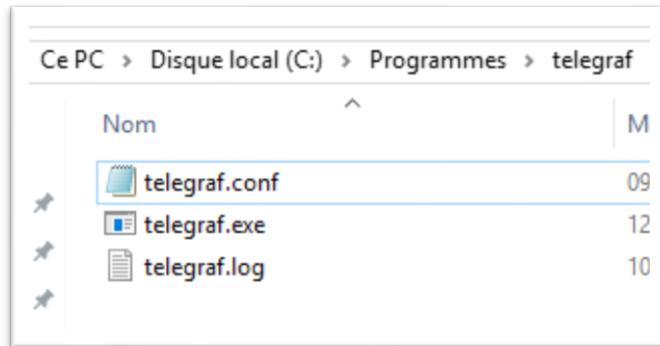
Ces dashboards permettent de superviser :

- Information système (Uptime, nombre de process et de threads et d'utilisateurs connectés)
- Usage de la ram et du cpu
- Disque I/
- Espace Disque restant
- Utilisation de la bande passante des différentes cartes réseau
- Paquets par second des différentes cartes réseau
- Erreurs de paquets des différentes cartes réseau
- Consommation de chaque process ainsi que le uptime

#### 5.6.3 Monitorer une machine Windows

- Télécharger telegraf sur la machine cible <https://portal.influxdata.com/downloads/>
- Dézipper le fichier téléchargé

- Créez un dossier *telegraf* dans *C:\Program Files*
- Copiez le contenu du fichier dézipper dans le dossier dans *C:\Program Files\telegraf*



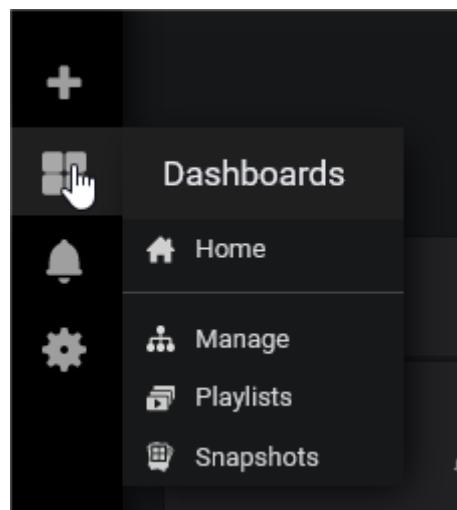
- Écrasez le fichier *C:\Program Files\telegraf\telegraf.conf* par le fichier ci-dessous



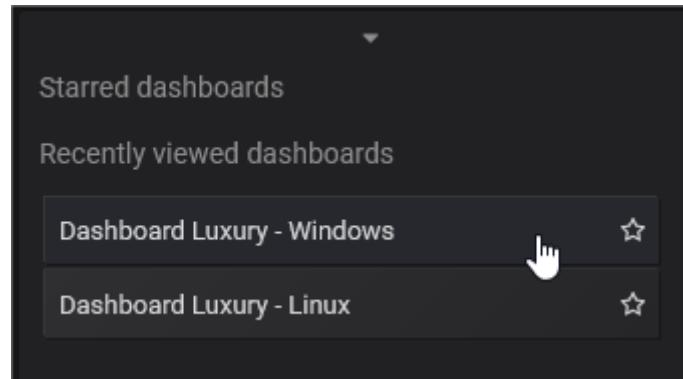
- Ouvrez la ligne de commande en administrateur et tapez les commandes suivantes pour lancer le service *telegraf*:

```
cd "C:\Program Files\telegraf"
telegraf.exe
```

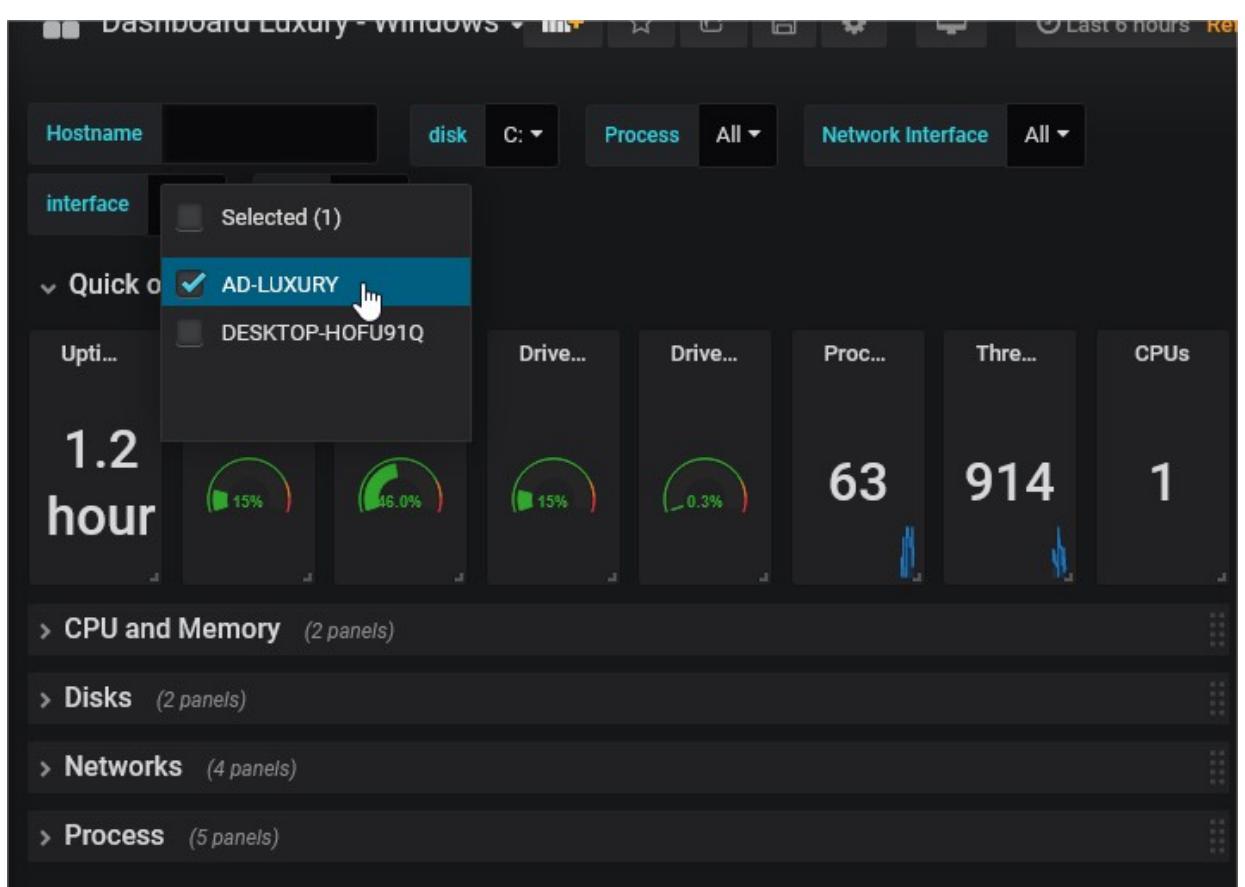
- Allez ensuite l'interface grafana depuis votre navigateur
- Cliquez sur le menu dashboard



- Cliquez sur le Dashboard – Windows



- Sélectionnez votre hostname :



#### 5.6.4 Monitorer une machine Linux

- Télécharger telegraf sur la machine cible avec la commande suivante

```
sudo apt install telegraf
```

- Sauvegarder la configuration par défaut

```
sudo mv /etc/telegraf/telegraf.conf /etc/telegraf/telegraf.conf.old
```

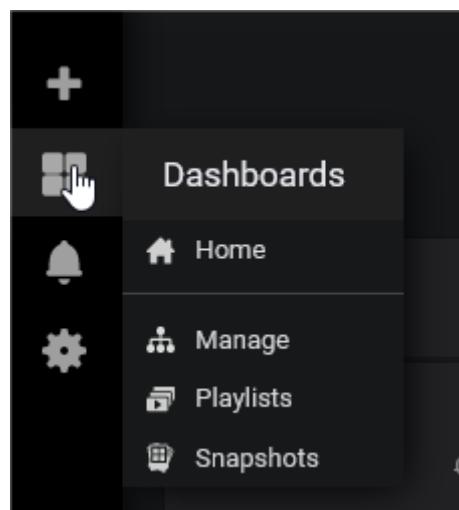
- Écraser le fichier `/etc/telegraf/telegraf.conf` par le fichier ci-dessous



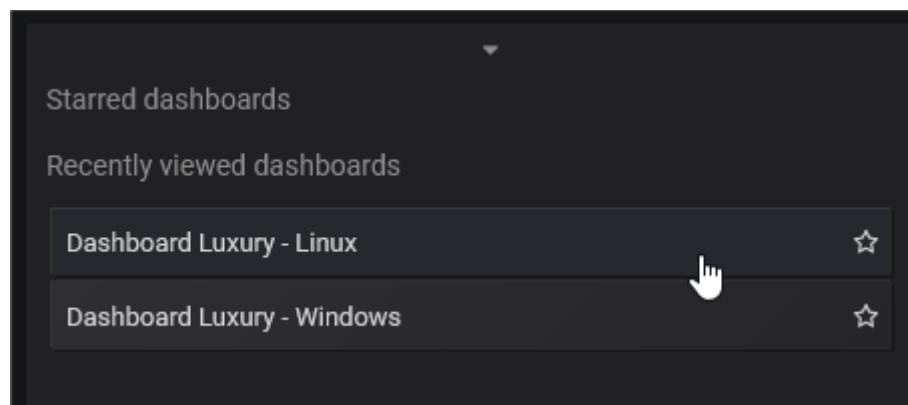
- Rechargez ensuite la config telegraf

```
sudo service telegraf restart
```

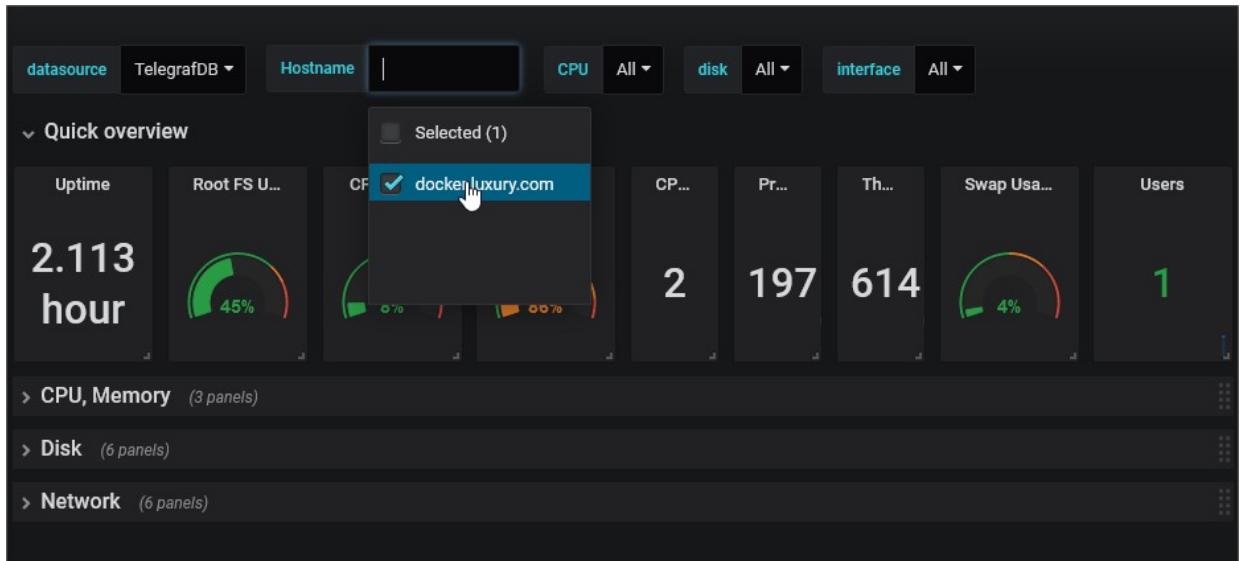
- Allez ensuite l'interface grafana depuis votre navigateur
- Cliquez sur le menu dashboard



- Cliquez sur le Dashboard – Linux



- Sélectionnez votre hostname :



## 4.7 Intégration de l'annuaire Active Directory dans Grafana

### 5.7.1 Déclaration de l'annuaire

L'intégration LDAP dans Grafana permet à vos utilisateurs Grafana de se connecter avec leurs informations d'identification LDAP. Nous pouvez également spécifier des mappages entre les appartenances à un groupe LDAP et les rôles d'utilisateur Grafana Organization.

Pour utiliser l'intégration LDAP, vous devez d'abord activer LDAP dans le fichier de configuration principal à savoir :

- Le fichier `/etc/grafana/grafana.ini` dans le conteneur
- Le fichier `/var/lib/docker/volumes/grafana_grafana-config/_data/grafana.ini` dans votre machine local

Et changez les valeurs suivantes :

```
[auth.ldap]
enabled = true
config_file = /etc/grafana/ldap.toml
allow_sign_up = true
```

Ensuite pour configurer votre LDAP il faut modifier le fichier :

- Le fichier `/etc/grafana/ldap.toml` dans le conteneur
- Le fichier `/var/lib/docker/volumes/grafana_grafana-config/_data/ldap.toml` dans votre machine local

Voici à quoi ressemble le fichier ldap.toml:

```
[[servers]]
host = "IP DU SERVEUR LDAP"
```

```

port = 389 use_ssl =
false start_tls = false
ssl_skip_verify = true
bind_dn = "CN=NOTRE CN,OU=votre OU,OU=Utilisateurs,OU=Notre
organisation,DC=votre DC,DC=LOCAL"
bind_password = '[YOUR_PASSWORD]' search_filter =
"(sAMAccountName=%s)" search_base_dns = ["DC=NOTRE
DC,DC=LOCAL"]

[servers.attributes] name =
"givenName" surname =
"sn"
username = "sAMAccountName"
member_of = "memberOf" email =
"mail"

# [[servers.group_mappings]]
# group_dn = "cn=admins,dc=grafana,dc=org"# org_role =
"Admin"
# To make user an instance admin (Grafana Admin) uncomment line below# grafana_admin = true
# The Grafana organization database id, optional, if left out the defaultorg (id 1) will be used
# org_id = 1

# Users that can view and edit dashboards
[[servers.group_mappings]]
group_dn = "CN=GP_votre CN,OU=votre OU,OU=Notre OU,OU==Notre OU,DC==Notre
DC,DC=LOCAL"
org_role = "Editor"

# Users that can only view dashboards
[[servers.group_mappings]]
group_dn = "CN=GP_votre CN,OU=votre OU,OU=Notre OU,OU==Notre OU,DC==NotreDC,DC=LOCAL"
org_role = "Niewer"

```

Un utilisateur peut appartenir à une ou plusieurs organisations et se voir attribuer différents niveaux de priviléges par le biais de rôles.

Ces groupes possèdent des rôles bien spécifiques dans grafana :

<b>Nom du role Grafana</b>	<b>Description du role</b>
Niewer	Les viewers peuvent modifier / inspecter les paramètres du tableau de bord dans le navigateur mais ne peuvent pas sauvegarder le tableau de bord
Editor	Les éditeurs peuvent administrer des tableaux de bord, des dossiers et des équipes qu'ils créent
Admin	Les administrateurs ont tous les droits

