

Mise en place et exploitation d'un serveur Grafana et Prometheus



Dans cette documentation nous installerons une solution de supervision matérielle avec le couple Grafana/Prometheus. Nous installerons aussi les agents sur les clients à surveiller.

Pour mieux s'y retrouver, cette documentation disposera de plusieurs screenshots illustrant les consignes.

Préambule

La supervision d'une infrastructure informatique consiste à surveiller et à analyser l'état de l'ensemble des composants de l'infrastructure pour garantir leur bon fonctionnement. Les intérêts sont :

1. Détection **rapide des problèmes** : La supervision permet de détecter rapidement les problèmes tels que les pannes, les dysfonctionnements et les erreurs, avant qu'ils ne deviennent des problèmes majeurs. Cela permet de réagir rapidement et de minimiser les temps d'arrêt et les perturbations pour les utilisateurs.
2. Amélioration de la **disponibilité** : En surveillant régulièrement les différents composants de l'infrastructure, la supervision permet de s'assurer que les systèmes sont disponibles en permanence. En cas de panne ou de défaillance, la supervision peut déclencher automatiquement des procédures de résolution pour minimiser le temps d'arrêt.
3. Optimisation des **performances** : La supervision permet de mesurer et d'analyser les performances des différents composants de l'infrastructure, ce qui peut aider à identifier les goulots d'étranglement et les points faibles. Les améliorations apportées à ces composants peuvent permettre d'optimiser les performances globales de l'infrastructure.
4. Réduction des **coûts** : En prévenant les temps d'arrêt et en optimisant les performances, la supervision peut aider à réduire les coûts liés à la maintenance et à la réparation de l'infrastructure informatique.
5. Amélioration de la **sécurité** : La supervision peut aider à détecter les tentatives d'intrusion et les attaques malveillantes sur l'infrastructure, permettant ainsi de prendre rapidement des mesures de sécurité pour les contrer.

En résumé, la supervision d'une infrastructure informatique est essentielle pour garantir le bon

fonctionnement de l'ensemble des composants de l'infrastructure. Elle permet de détecter rapidement les problèmes, d'optimiser les performances, de réduire les coûts et d'améliorer la sécurité.

Que superviser ?

1. Les **serveurs** : Les serveurs sont l'un des éléments les plus critiques de toute infrastructure informatique. Il est donc essentiel de superviser leur état de santé, leur charge de travail, leurs performances et leurs temps de réponse pour garantir leur disponibilité et leur bon fonctionnement.
2. **Les réseaux** : La supervision des réseaux permet de s'assurer que les connexions entre les différents composants de l'infrastructure sont rapides, fiables et sécurisées. La supervision peut inclure la surveillance des connexions Internet, des réseaux locaux (LAN) et des réseaux étendus (WAN).
3. **Les bases de données** : Les bases de données sont souvent le cœur de l'activité de nombreuses entreprises. Il est donc essentiel de superviser leur état de santé, leur utilisation des ressources et leur disponibilité pour garantir leur bon fonctionnement.
4. **Les applications** : Les applications sont souvent les interfaces les plus visibles pour les utilisateurs finaux. La supervision des applications peut inclure la surveillance de leur temps de réponse, de leur disponibilité et de leur utilisation des ressources.
5. **Les périphériques** : Les périphériques tels que les imprimantes, les scanners et les caméras peuvent également être supervisés pour garantir leur bon fonctionnement.

Pourquoi le duo Grafana/Prometheus?

Prometheus est un logiciel de supervision open-source créé par SoundCloud. En 2013, SoundCloud a décidé d'utiliser **Prometheus** pour ses infrastructures de production et a publié la version 1.0 en juillet 2016.

Prometheus, écrit en *GO*, s'impose depuis comme la solution de référence pour superviser une infrastructure de type *Cloud, SaaS/Openstack, OKD, K8S*.

Il existe plusieurs autres solutions de supervision sur le marché :

- Zabbix ;
- Nagios ;
- Centreon ;
- Sensu ;

Mais sont généralement assez coûteuses à déployer.

Grafana est un logiciel Open Source pour la visualisation et la supervision d'une infrastructure. Ce logiciel propose une connexion native à **Prometheus** et propose une liste de dashboards pré-générés pour récupérer les informations en provenance de **Prometheus**.

Comparaison avec Kibana, une alternative à Grafana :

La principale nuance est l'utilisation d'**Elastic Search** pour récupérer les données par **Kibana**. (basé sur le **logs**) **Grafana** prends en charge plusieurs autres méthodes de stockage de metrics. (basé sur

les **metrics**)

Points of Difference	Kibana	Grafana
Compatibility	With Elasticsearch data source.	With influx DB, Graphite, Logzio and AWS Cloudwatch.
Access	Public	Based on the job responsibilities of the users.
Tool Functionalities	Geographical data relevance, Bar, Heatmap and Pie Charts.	Data visualization through Heatmap and mixing of data from multiple sources.
Working	Log based.	Metrics based.
Alerts	<ul style="list-style-type: none"> • Same user different location logging. • Meeting the demands if your content is boosting on social platforms. • If credit card numbers are visible in application logs. 	User can define its alert visually for the most important metrics
Installations	Must for the user to connect with Elasticsearch because of YAML files.	Configured with .ini files.

Prérequis

Nous considérons que vous avez déjà montés plusieurs serveurs afin de pouvoir les surveiller.

Nous considérons que vous êtes équipé de cette manière : - Une VM sous Debian 11 vierge **[Grafana/Prometheus]** - Une VM sous Debian 11 vierge **[Le serveur à monitorer]**

Les allocations de matériel (CPU/RAM...) sont à allouer selon vos envies, attention à respecter la configuration minimale. C'est à dire :

Pour le duo **Grafana/Prometheus** : - 2GB de ram - 2 cœurs de CPU - 20GB d'espace disque - *Debian 11*

Nos IP pour notre infrastructure seront : - [Grafana]: **10.192.43.12** (:3000 pour le port WEB) - [Supervision] : **10.192.43.13** (Le serveur à surveiller)

Mot de passe par défaut sur toutes les sessions : **Not24get**

Rappel des deux commandes essentiels : - ip a (connaitre son adresse IP) - nano /etc/network/interfaces (configuration de l'interface réseau)

<WRAP center round tip 60%> Ajouter les deux machines dans un logiciel tel que mRemoteNG pour faciliter l'administration.

Installation de Prometheus



Depuis les sources provenant de Prometheus

Ajout d'un compte de service "prometheus"

- Ajouter le compte de service :

```
groupadd --system prometheus
```

- Assigner un groupe et retirer la possibilité de se connecter

```
useradd -s /sbin/nologin --system -g prometheus prometheus
```

Création des dossiers pour Prometheus

- Créer les dossiers que Prometheus aura besoin pour enregistrer les configurations.

```
mkdir /etc/prometheus  
mkdir /var/lib/prometheus
```

Téléchargement et installation à partir de la source

Changer de répertoire temporaire pour le téléchargement, exemple `/home/adminlocal`.

- Télécharger la source avec la commande `curl` :

<WRAP center round tip 60%> La commande vient directement chercher la dernière version de Prometheus, il n'est pas nécessaire de renseigner la version.

[snippet.bash](#)

```
curl -s  
https://api.github.com/repos/prometheus/prometheus/releases/latest | grep
```

```
browser_download_url|grep linux-amd64|cut -d '"' -f 4|wget -qi -
```

- Extraire l'archive :

Trouver le nom du fichier télécharger avec la commande `ls`.

```
tar -xvf prometheus-2.42.0.linux-amd64.tar.gz
```

- Copier les fichiers dans le dossier créé précédemment

```
mv prometheus-2.42.0.linux-amd64 /etc/prometheus
```

- Attribuer les permissions à l'utilisateur Prometheus des dossiers

```
chown prometheus:prometheus /etc/prometheus
chown prometheus:prometheus /var/lib/prometheus
chown -R prometheus:prometheus /etc/prometheus/consoles
chown -R prometheus:prometheus /etc/prometheus/console_libraries
```

Copie des fichiers

```
cp /etc/prometheus/prometheus /usr/local/bin/
cp /etc/prometheus/promtool /usr/local/bin/
```

Ajout au démarrage automatique

- Création du service

```
nano /etc/systemd/system/prometheus.service
```

- Ajouter la configuration dans le fichier de service

[snippet.bash](#)

```
[Unit]
Description=Prometheus
Documentation=https://prometheus.io/docs/introduction/overview/
Wants=network-online.target
After=network-online.target
[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
--config.file /etc/prometheus/prometheus.yml \
```

```
--storage.tsdb.path /var/lib/prometheus/ \  
--web.console.templates=/etc/prometheus/consoles \  
--web.console.libraries=/etc/prometheus/console_libraries
```

[Install]

WantedBy=multi-user.target

- Redémarrer le daemon

```
systemctl daemon-reload
```

- Activer le service au démarrage

```
systemctl enable prometheus
```

- Vérifier la bonne exécution de Prometheus

```
systemctl status prometheus
```

```
root@srv-supervision:~/etc/prometheus# systemctl status prometheus
● prometheus.service - Prometheus
   Loaded: loaded (/etc/systemd/system/prometheus.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2022-12-08 17:46:25 CET; 4 days ago
     Main PID: 837 (prometheus)
        Tasks: 8 (limit: 1128)
       Memory: 94.7M
          CPU: 5min 44.159s
   CGroup: /system.slice/prometheus.service
           └─837 /usr/local/bin/prometheus --config.file /etc/prometheus/prometheus.yml --storage.tsdb.pat

déc. 13 11:11:17 srv-supervision prometheus[837]: level=info ts=2022-12-13T10:11:17.350510678Z caller=head.g
déc. 13 11:11:17 srv-supervision prometheus[837]: level=error ts=2022-12-13T10:11:17.350981874Z caller=head.g
déc. 13 11:11:17 srv-supervision prometheus[837]: level=info ts=2022-12-13T10:11:17.36948459Z caller=compact
déc. 13 11:11:17 srv-supervision prometheus[837]: level=info ts=2022-12-13T10:11:17.452217729Z caller=head.g
déc. 13 11:11:17 srv-supervision prometheus[837]: level=error ts=2022-12-13T10:11:17.452656452Z caller=head.g
déc. 13 11:11:17 srv-supervision prometheus[837]: level=info ts=2022-12-13T10:11:17.468179164Z caller=compact
déc. 13 11:11:17 srv-supervision prometheus[837]: level=info ts=2022-12-13T10:11:17.54469682Z caller=head.g
déc. 13 11:11:17 srv-supervision prometheus[837]: level=error ts=2022-12-13T10:11:17.545115227Z caller=head.g
déc. 13 11:11:17 srv-supervision prometheus[837]: level=info ts=2022-12-13T10:11:17.562376631Z caller=compact
déc. 13 11:11:17 srv-supervision prometheus[837]: level=info ts=2022-12-13T10:11:17.6731403Z caller=compact
```

Accès à l'interface web

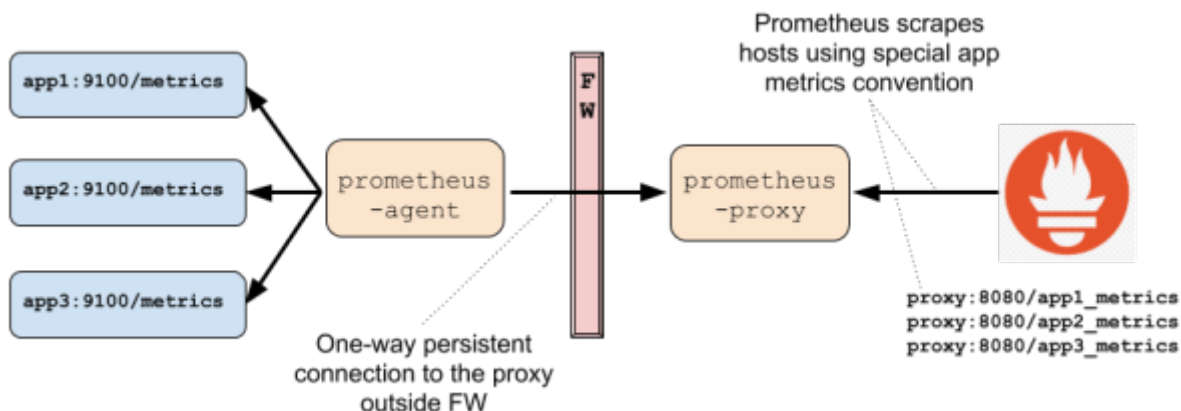
Le serveur web de Prometheus est disponible à l'adresse : <http://ip-serveur:9090>.

Ajouter une règle au Pare-feu local

- Autoriser le port 9090

```
ufw allow 9090
```

Utilisation de Prometheus derrière un Firewall



Pour superviser une infrastructure complexe qui comporte plusieurs niveaux d'isolations, le mode *Pull* devient problématique. Au lieu d'utiliser le mode *Push* qui vient à l'esprit naturellement, Prometheus fourni un *proxy* permettant de conserver le modèle *Pull*, et de superviser tous les systèmes derrière le(s) *Firewall(s)*.

Le *Proxy* est décomposé en deux parties :

- Le proxy qui s'exécute sur la même zone que le serveur ;
- L'agent qui s'exécute derrière le firewall et gère les requêtes en provenance du *Proxy*.

L'agent peut s'exécuter :

- Comme standalone serveur ;
- Embarqué dans un autre serveur ;
- Comme simple Agent Java.

Un *proxy* peut gérer un ou plusieurs agents.

Installation des agents sur les clients

Fonctionnement mode Pull / Push

Par défaut, Prometheus fonctionne en mode *Pull*, c'est à dire que le serveur interroge à intervalle régulier les instances clientes sur lesquelles les *Exporters* sont installés.

Il est possible, quand cela s'avère nécessaire de fonctionner en mode *Push* en utilisant le projet [Prometheus Push Gateway](#). Le seul cas où ce module aurait un intérêt serait pour la supervision de jobs asynchrones. Ces jobs pourraient envoyer des données au Prometheus Server.

Sur Linux : node_exporter

Télécharger node_exporter par les sources

Changer de répertoire temporaire pour le téléchargement, exemple /home/adminlocal.

- Récupérer l'archive avec la commande curl :

<WRAP center round tip 60%> La commande vient directement chercher la dernière version de node_exporter, il n'est pas nécessaire de renseigner la version.

[snippet.bash](#)

```
curl -s  
https://api.github.com/repos/prometheus/node_exporter/releases/latest |  
grep browser_download_url | grep linux-amd64 | cut -d '"' -f 4 | wget -qi -
```

- Extraire l'archive

```
tar -xvf node_exporter-*.linux-amd64.tar.gz
```

- Copier les fichiers dans le répertoire bin

Utiliser la commande ls pour lister les fichiers.

```
mv node_exporter-*.linux-amd64/node_exporter /usr/local/bin/
```

- Création du compte de service

```
useradd -rs /bin/false node_exporter
```

Ajouter le service node_exporter

- Création du fichier de service

```
nano /etc/systemd/system/node_exporter.service
```

- Ajouter la configuration dans le fichier de service

[snippet.bash](#)

```
[Unit]  
Description=Node Exporter Node02  
After=network.target
```



```
[Service]
User=node_exporter
Group=node_exporter
Type=simple
ExecStart=/usr/local/bin/node_exporter

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

- Redémarrer le daemon

```
systemctl daemon-reload
```

- Activer le service au démarrage

```
systemctl enable node_exporter
```

- Vérifier la bonne exécution de node_exporter

```
systemctl status node_exporter
```

```
root@node01:/home/adminlocal# systemctl status node_exporter
● node_exporter.service - Node Exporter Node02
   Loaded: loaded (/etc/systemd/system/node_exporter.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2022-12-13 13:38:35 CET; 2s ago
     Main PID: 40287 (node_exporter)
        Tasks: 4 (limit: 1128)
       Memory: 4.8M
          CPU: 11ms
       CGroup: /system.slice/node_exporter.service
              └─40287 /usr/local/bin/node_exporter

déc. 13 13:38:35 node01 node_exporter[40287]: ts=2022-12-13T12:38:35.069Z caller=node_exporter.go:117 level=
déc. 13 13:38:35 node01 node_exporter[40287]: ts=2022-12-13T12:38:35.069Z caller=node_exporter.go:117 level=
déc. 13 13:38:35 node01 node_exporter[40287]: ts=2022-12-13T12:38:35.069Z caller=node_exporter.go:117 level=
déc. 13 13:38:35 node01 node_exporter[40287]: ts=2022-12-13T12:38:35.069Z caller=node_exporter.go:117 level=
déc. 13 13:38:35 node01 node_exporter[40287]: ts=2022-12-13T12:38:35.069Z caller=node_exporter.go:117 level=
déc. 13 13:38:35 node01 node_exporter[40287]: ts=2022-12-13T12:38:35.069Z caller=node_exporter.go:117 level=
déc. 13 13:38:35 node01 node_exporter[40287]: ts=2022-12-13T12:38:35.069Z caller=node_exporter.go:117 level=
déc. 13 13:38:35 node01 node_exporter[40287]: ts=2022-12-13T12:38:35.070Z caller=node_exporter.go:232 level=inf
déc. 13 13:38:35 node01 node_exporter[40287]: ts=2022-12-13T12:38:35.070Z caller=tsdb/config.go:235 level=inf
```

Sur Windows : windows_exporter

Téléchargement et installation à partir la source

- Télécharger la source

Choisir la version **x64**.

```
https://github.com/prometheus-community/windows_exporter/releases/tag/v0.21.0
```

- Lancer l'installation

<WRAP center round info 60%> Plusieurs arguments sont disponibles sur la documentation de [windows_exporter](#).

[snippet.powershell](#)

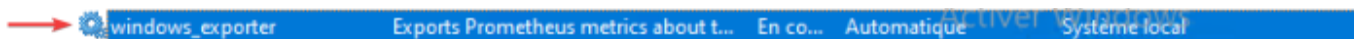
```
msiexec /i <path-to-msi-file> ENABLED_COLLECTORS=os,iis  
LISTEN_PORT=9182 LISTEN_ADDR=0.0.0.0
```

L'installation se déroule en silent, aucune information ne vous est demandée.

Vérification si l'application s'exécute bien

- Dans les services Windows

Démarrer l'utilitaire de gestion des services Windows avec `services.msc`.



On retrouve bien le service En cours d'exécution.

- Via la page web de l'API

Accéder à la page : <http://localhost:9182/>



Ajout des agents dans Prometheus

Modification du fichier de configuration

Ici, vous ajouterez les différents clients dans la configuration de Prometheus.

```
nano /etc/prometheus/prometheus.yml
```

A la fin du fichier ajouter toutes les clients, vous pouvez trier par type d'OS.

```
GNU nano 5.4 /etc/prometheus/prometheus.yml *
# 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:
      - targets:
        # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# 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'
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'node_exporter'
    static_configs:
      - targets: ['10.192.43.10:9100']
```

snippet.yml

```
- job_name: 'node_exporter'
  static_configs:
    - targets: ['10.192.43.10:9100']
```

Vérification du fichier de configuration

Utiliser l'utilitaire `promtool` pour vérifier si le fichier `.yml` ne contient aucune erreur.

```
promtool check config /etc/prometheus/prometheus.yml
```

```
root@srv-supervision:/etc/prometheus# promtool check config /etc/prometheus/prometheus.yml
Checking /etc/prometheus/prometheus.yml
SUCCESS: 0 rule files found
```

<WRAP center round info 60%> Aucune erreur n'a été détectée. Pensez à utiliser cette commande dès qu'une modification est faite dans le fichier config.

Appliquer la configuration

- Redémarrer le service Prometheus

```
systemctl restart prometheus
```

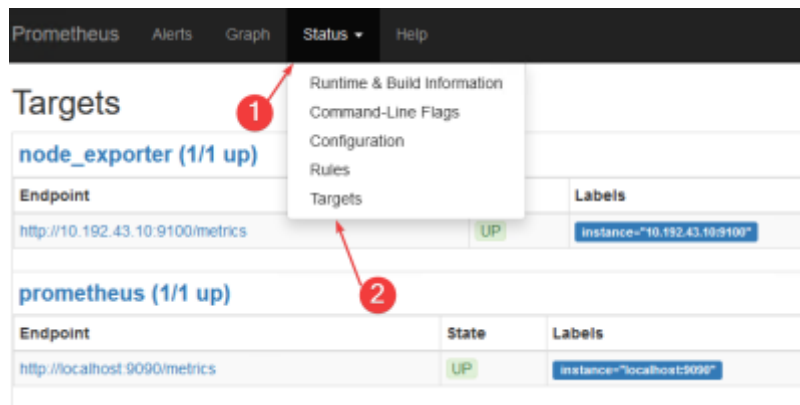
- Vérifier l'état du service

```
systemctl status prometheus
```

Vérification dans l'application web Prometheus

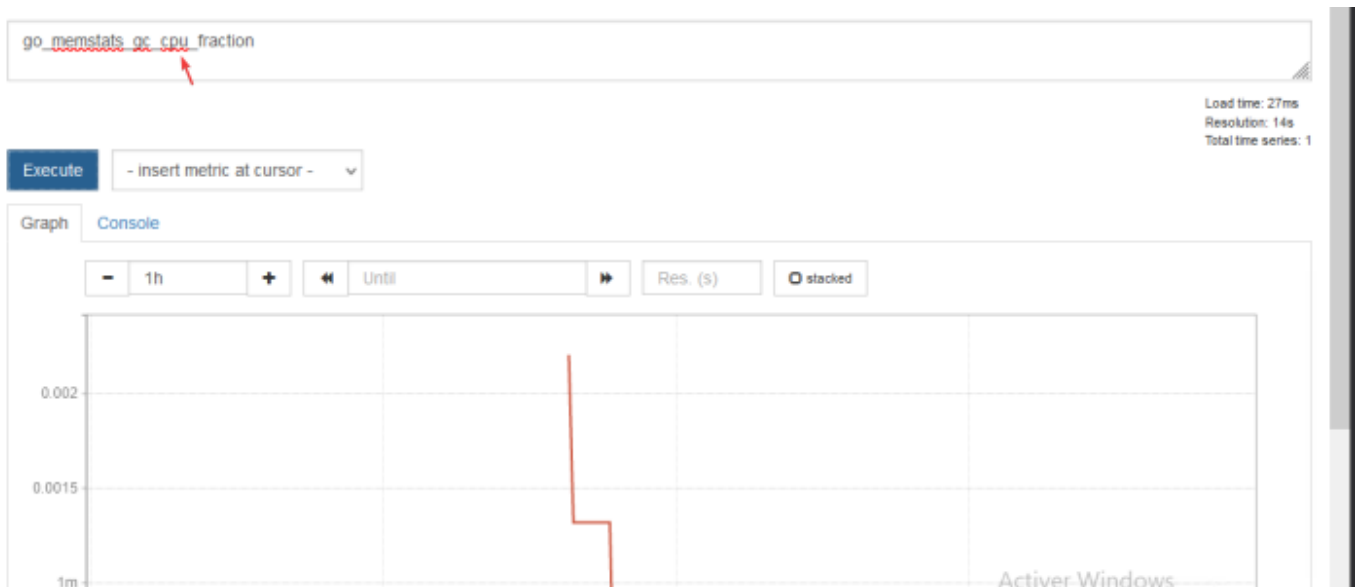
Sur l'interface web de Prometheus, accessible sur <http://serveur-ip:9090/>.

Dans le menu Status → Targets, vous trouverez tous les clients qui ont été ajoutés dans la configuration de Prometheus.



Dans l'onglet d'accueil de Prometheus, vous pouvez faire des requêtes.

<WRAP center round tip 60%> Utilisez l'autocomplétion pour construire vos requêtes.



<WRAP center round info 60%> Toutes les machines sont désormais dans Prometheus. La configuration de Prometheus reste assez simple, le traitement des données est effectué par Grafana.

Installation de Grafana



Téléchargement et installation à partir du repo

- Installer le gestionnaire de dépôt (permet de rajouter un dépôt facilement) :

```
apt-get install -y apt-transport-https
```

```
apt-get install -y software-properties-common wget
```

- Ajouter la clé gpg :

```
wget -q -O /usr/share/keyrings/grafana.key https://apt.grafana.com/gpg.key
```

- Ajouter le dépôt dans les sources

snippet.bash

```
echo "deb [signed-by=/usr/share/keyrings/grafana.key]
https://apt.grafana.com stable main" | tee -a
/etc/apt/sources.list.d/grafana.list
```

- Mettez à jour vos sources

```
apt update
```

- Démarrer l'installation avec un simple `apt install`

```
apt -y install grafana
```

Ajout au démarrage automatique

```
systemctl daemon-reload
```

```
systemctl enable grafana-server
```

Démarrer le serveur web

```
systemctl start grafana-server
```

- Vérifier le bon fonctionnement

```
systemctl status grafana-server
```

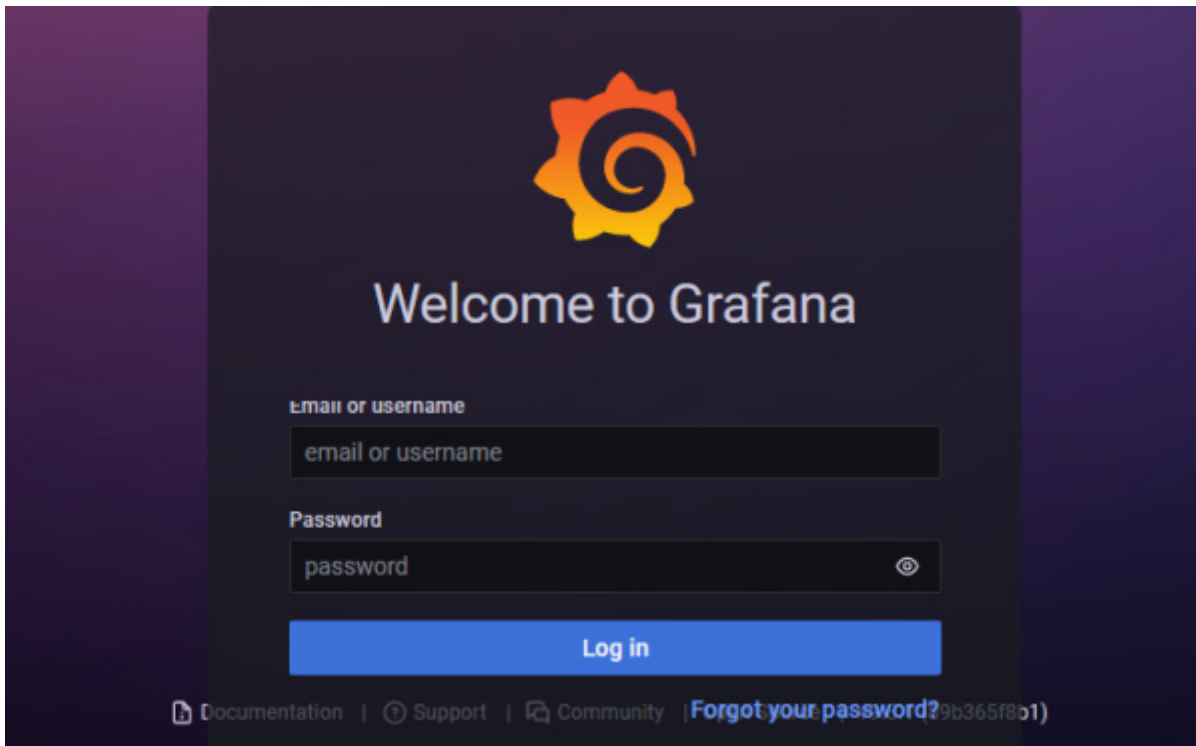
```
root@srv-supervision:~# systemctl status grafana-server
● grafana-server.service - Grafana instance
   Loaded: loaded (/lib/systemd/system/grafana-server.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2022-12-13 19:53:34 CET; 13min ago
     Docs: http://docs.grafana.org
   Main PID: 830 (grafana-server)
    Tasks: 13 (limit: 1128)
   Memory: 129.7M
      CPU: 2.693s
   CGroup: /system.slice/grafana-server.service
           └─830 /usr/sbin/grafana-server --config=/etc/grafana/grafana.ini --pidfile=/run/grafana/grafana-server.pid --packaging=deb cfg:default

déc. 13 20:00:28 srv-supervision grafana-server[830]: logger=context userId=1 orgId=1 uname=admin t=2022-12-13T20:00:28.159007479+01:00 level=error
déc. 13 20:00:28 srv-supervision grafana-server[830]: logger=context userId=1 orgId=1 uname=admin t=2022-12-13T20:00:28.159471303+01:00 level=error
déc. 13 20:00:29 srv-supervision grafana-server[830]: logger=context userId=1 orgId=1 uname=admin t=2022-12-13T20:00:29.161526828+01:00 level=info
déc. 13 20:00:29 srv-supervision grafana-server[830]: logger=context userId=1 orgId=1 uname=admin t=2022-12-13T20:00:29.164678049+01:00 level=info
déc. 13 20:00:29 srv-supervision grafana-server[830]: logger=context userId=1 orgId=1 uname=admin t=2022-12-13T20:00:29.167493015+01:00 level=info
déc. 13 20:00:29 srv-supervision grafana-server[830]: logger=context userId=1 orgId=1 uname=admin t=2022-12-13T20:00:29.168713339+01:00 level=info
déc. 13 20:00:29 srv-supervision grafana-server[830]: logger=context userId=1 orgId=1 uname=admin t=2022-12-13T20:00:29.498921051+01:00 level=info
déc. 13 20:00:29 srv-supervision grafana-server[830]: logger=context userId=1 orgId=1 uname=admin t=2022-12-13T20:00:29.500211861+01:00 level=info
déc. 13 20:00:43 srv-supervision grafana-server[830]: logger=context userId=1 orgId=1 uname=admin t=2022-12-13T20:00:43.434523106+01:00 level=info
déc. 13 20:03:40 srv-supervision grafana-server[830]: logger=cleanup t=2022-12-13T20:03:40.104929845+01:00 level=info msg="Completed cleanup jobs"
```

<WRAP center round info 60%>

- Le nom du service pour le serveur Grafana est grafana - server.
- Les fichiers de configurations sont dans /etc/grafana/grafana.ini
- Les logs sont disponibles dans /var/log/grafana/grafana.log
- La BDD qui contient les configurations est dans /var/lib/grafana/grafana.db
- Les autres fichiers (html/css...) sont dans /usr/share/grafana

Accéder au serveur web



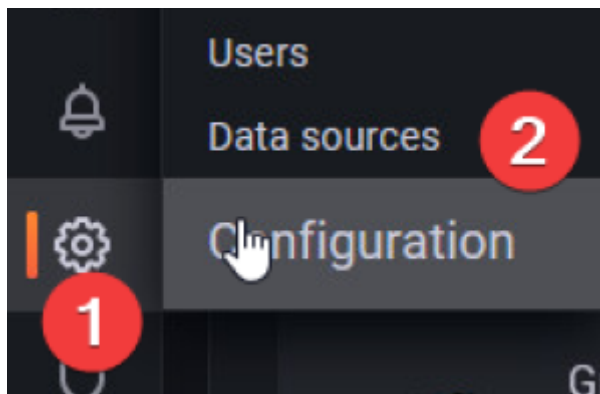
Le serveur web écoute sur le port 3000.

Les identifiants par défaut sont : admin/admin. Le mot de passe sera à changer.

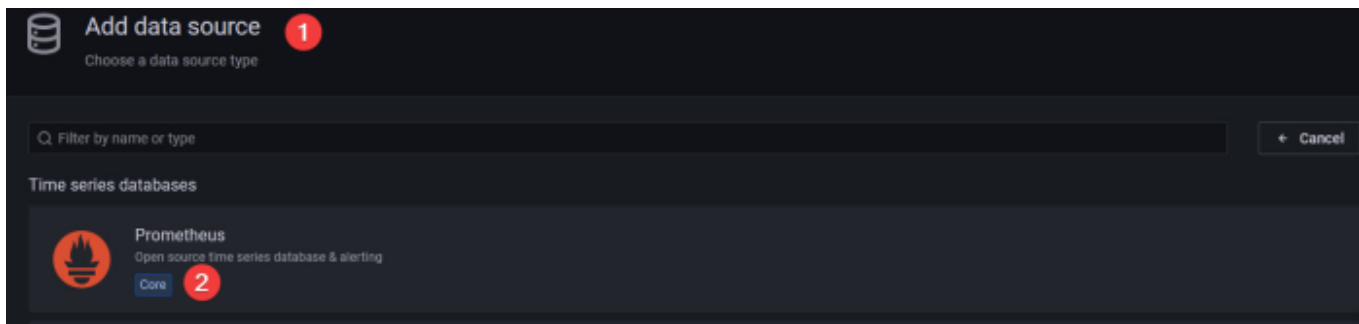
Configuration de Grafana

Ajout de la data-source Prometheus

- Ajouter une data-source depuis le menu latéral



- Sélectionner Prometheus



- Renseigner l'IP du serveur Prometheus (en local)



<WRAP center round important 60%> Si votre installation de Prometheus est en local, c'est à dire sur la même machine qui exécute Grafana vous devez renseigner localhost:9090.

Terminer en validant la configuration, Prometheus est désormais lié au serveur Grafana.

Ajout de tableaux

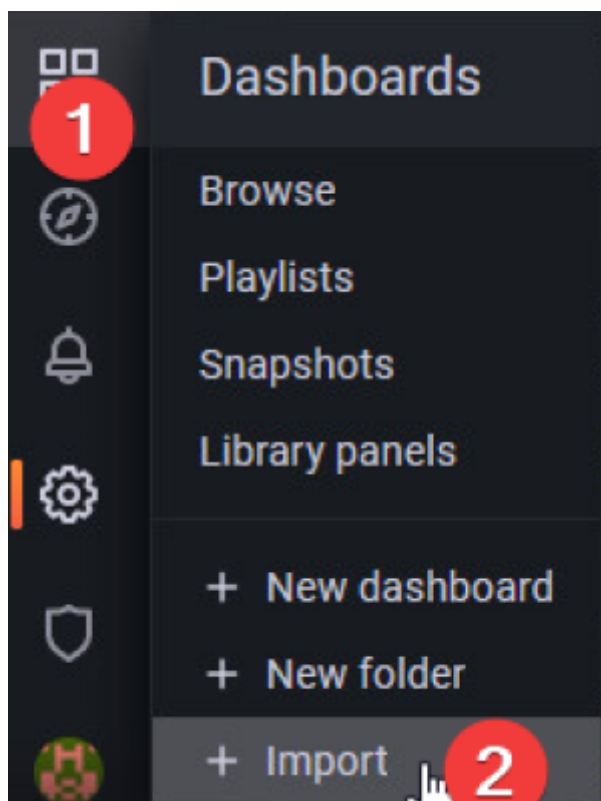
Par importation avec un code d'identification ou par JSON

Voici donc deux tableaux que je recommande à l'utilisation avec node_exporter et windows_exporter :

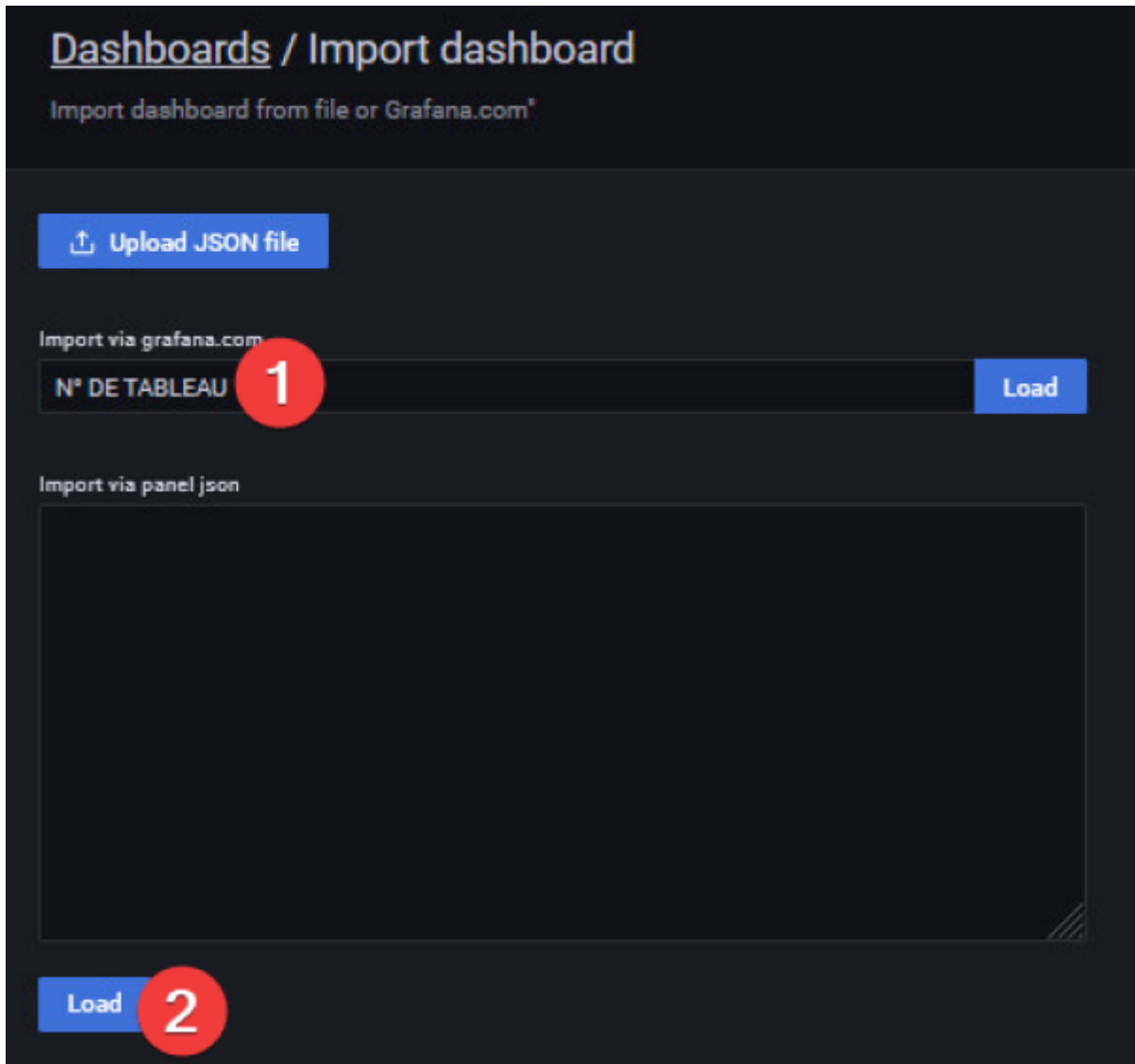
- Le tableau n°14451 pour les données windows_exporter.
- Le tableau n°11074 pour les données node_exporter.

Ils sont très bien construit et permettent une utilisation "Out of the box".

- Ouvrir le gestionnaire d'importation de tableau



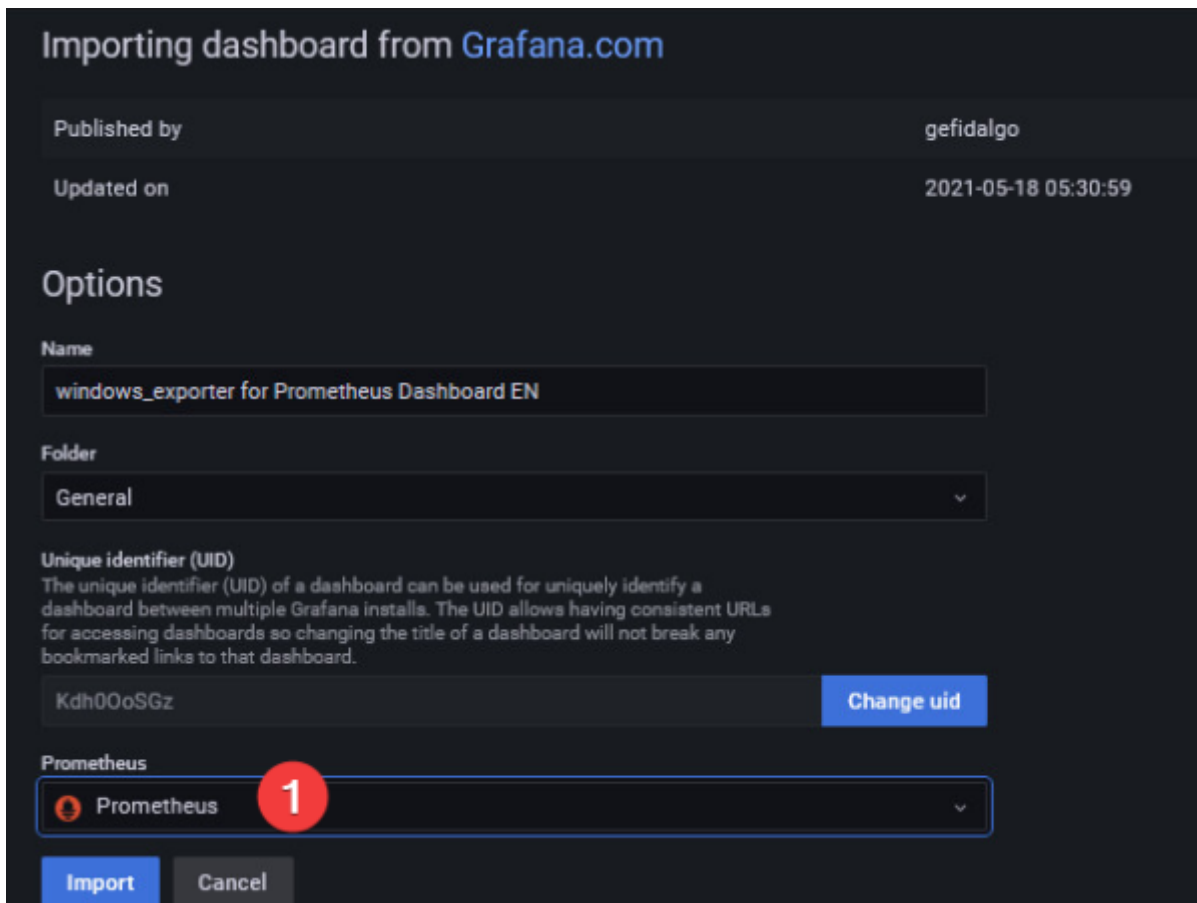
- Indiquer le n° de tableau puis importer.



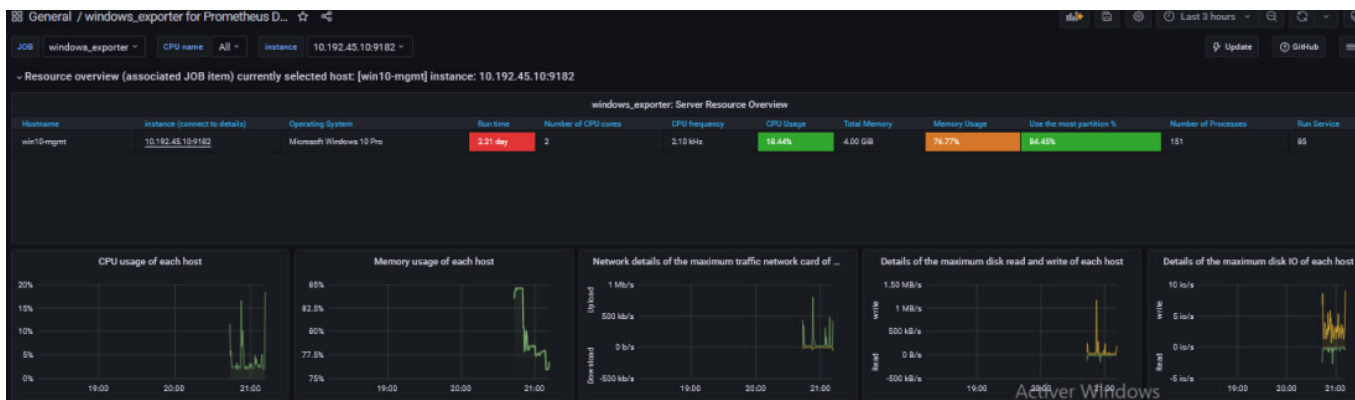
<WRAP center round tip 60%> Il est aussi possible de coller un code j son d'un tableau.

- Renseigner la source de données

Pensez à le renommer avec de l'importer.



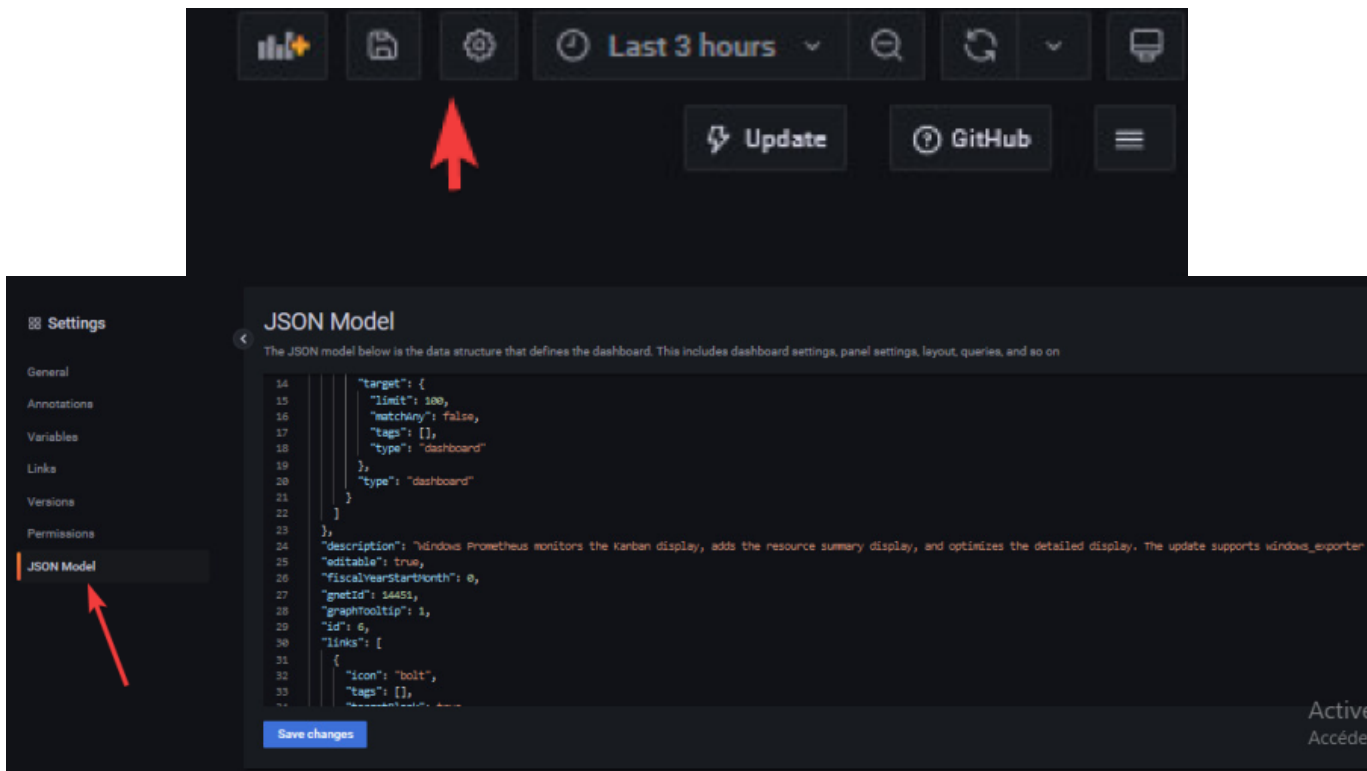
Votre tableau est désormais disponible et commence déjà à traiter les données.



Personnalisation des tableaux

Modification par le JSON

Si vous souhaitez modifier un tableau importé ou un tableau que vous avez créé, vous pouvez utiliser l'éditeur json accessible ici :



Ajout d'un AlertManager

Dans Grafana, les alertes sont directement liées au graphiques, c'est la raison pour laquelle nous créons un tableau de bord par serveur. Vous devez ensuite définir le seuil critique. Par exemple 75 % pour l'utilisation disque. Une fois votre alerte en place, vous devriez avoir un cœur s'affichant à côté du titre de votre graphique, affiché en vert quand tout va bien et en rouge en cas d'alerte. Prometheus gère aussi les alertes avec un plugin à installer.

Fonctionnement d'AlertManager

Prenons un exemple pour mieux expliquer le cycle de vie d'une alerte. Nous avons une alerte simple qui surveille la charge 1m d'un noeud, et qui se déclenche lorsqu'elle est supérieure à 20 pendant au moins 1 minute.

[snippet.yaml](#)

```
ALERT NODE_LOAD_1M
  IF node_load1 > 20
  FOR 1m
```

Prometheus est configuré pour récupérer les métriques toutes les 20 secondes, et l'intervalle d'évaluation est de 1 minute.

[snippet.yaml](#)

```

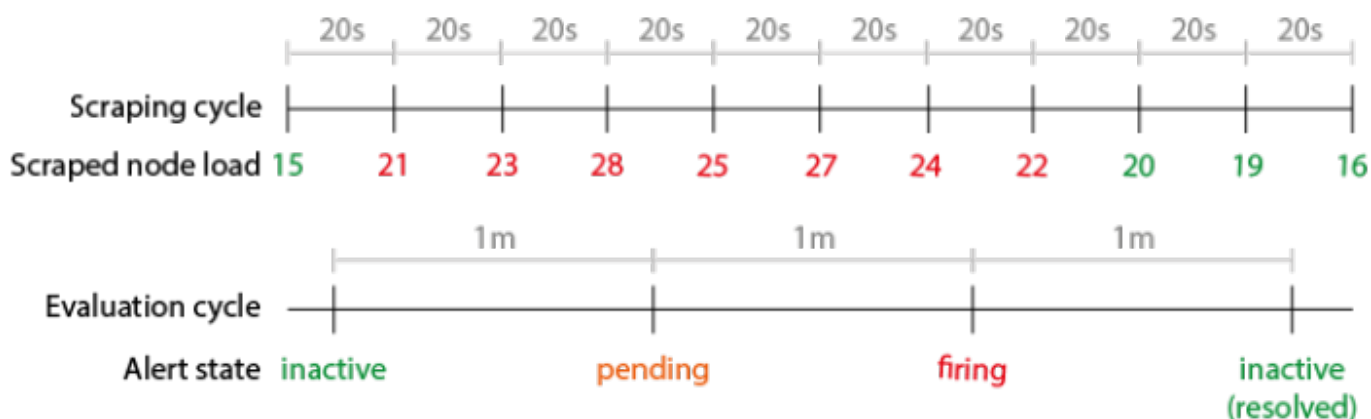
global :
  scrape_interval : 20s
  evaluation_interval : 1m

```

Question : combien de temps faut-il pour lancer NODE_LOAD_1M, une fois que la charge moyenne sur la machine est supérieure à 20 ?

Réponse : il faut un temps compris entre 1m et 20s + 1m + 1m. La limite supérieure est probablement plus élevée que ce à quoi vous vous attendez lorsque vous fixez FOR 1m, mais elle est tout à fait logique dans l'architecture Prometheus.

Le cycle de vie d'une alerte explique la raison d'un tel délai dans le pire des cas. Le diagramme suivant montre la séquence des événements sur une ligne de temps :



La charge d'un nœud change constamment, mais elle est analysée par Prometheus tous les `scrape_interval` (c'est-à-dire 20 secondes). Les règles d'alerte sont ensuite évaluées par rapport aux métriques scrappées tous les `evaluation_interval` (c'est-à-dire 1 minute). Lorsqu'une expression de règle d'alerte est TRUE (c'est-à-dire `node_load1 > 20`), l'alerte passe en pending, afin d'honorer la clause FOR. Lors des cycles d'évaluation suivants, si l'expression de l'alerte est toujours vraie, une fois que la clause FOR est honorée, l'alerte passe finalement au firing et une notification est envoyée au gestionnaire d'alertes.

Ajout des règles pour Prometheus

Toujours dans le répertoire `/etc/prometheus/`.

```

touch prometheus_rules.yml
nano prometheus_rules.yml

```

```
GNU nano 5.4 prometheus.rules.yml
groups:
- name: custom_rules
  rules:
  - record: node_memory_MemFree_percent
    expr: 100 - (100 * node_memory_MemFree_bytes / node_memory_MemTotal_bytes)

  - record: node_filesystem_free_percent
    expr: 100 * node_filesystem_free_bytes(mountpoint="/") / node_filesystem_size_bytes(mountpoint="/")

- name: alert_rules
  rules:
  - alert: InstanceDown
    expr: up == 0
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: "Instance {{{ $labels.instance }}} down"
      description: "[{{{ $labels.instance }}} of job [{{{ $labels.job }}}] has been down for more than 1 minute."
  - alert: DiskSpaceFree10Percent
    expr: node_filesystem_free_percent <= 10
    labels:
```

- Vérification avec promtool

```
promtool check rules /etc/prometheus/prometheus.rules.yml
```

Résultat :

```
Checking prometheus.rules.yml
SUCCESS: 4 rules found
```

Installation d'Alertmanager

- Création du compte de service

```
useradd -M -r -s /bin/false alertmanager
```

- Téléchargement des sources avec la commande wget

<WRAP center round tip 60%> Vérifier la dernière version sur :
<https://github.com/prometheus/alertmanager/releases/>.

- Définir une variable provisoire pour le wget

```
VER=0.25.0
```

- Téléchargement de la source

```
wget
https://github.com/prometheus/alertmanager/releases/download/v$VER/alertmanager-$VER.linux-amd64.tar.gz
```

- Extraire la source

```
tar xzf alertmanager-$VER.linux-amd64.tar.gz
```

- Copie des fichiers sources dans /usr/local/bin

```
cp alertmanager-$VER.linux-amd64/{alertmanager,amtool} /usr/local/bin/
```

- Copie des fichiers de configuration dans /etc/alertmanager

```
mkdir /etc/alertmanager
mkdir /etc/alertmanager/data
cp alertmanager-$VER.linux-amd64/alertmanager.yml /etc/alertmanager/
```

- Définition des droits pour l'utilisateur alertmanager

```
chown alertmanager: /etc/alertmanager/alertmanager.yml
/usr/local/bin/{alertmanager,amtool}
```

Création du service d'Alertmanager

- Création du service alertmanager

```
nano /etc/systemd/system/alertmanager.service
```

[snippet.bash](#)

```
[Unit]
Description=AlertManager Serveur Service
Wants=network-online.target
After=network-online.target

[Service]
User=alertmanager
Group=
Type=simple
ExecStart=/usr/local/bin/alertmanager --config.file
/etc/alertmanager/alertmanager.yml --
storage.path="/etc/alertmanager/data"

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

- Exécution du service

```
systemctl daemon-reload
```

```
systemctl enable --now alertmanager
```

- Vérifier l'état

```
systemctl status alertmanager
```

```
root@srv-supervision:/etc/prometheus# systemctl daemon-reload
root@srv-supervision:/etc/prometheus# systemctl restart alertmanager
root@srv-supervision:/etc/prometheus# systemctl status alertmanager
● alertmanager.service - AlertManager Serveur Service
   Loaded: loaded (/etc/systemd/system/alertmanager.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2022-12-19 00:37:46 CET; 6s ago
     Main PID: 116694 (alertmanager)
        Tasks: 6 (limit: 1128)
      Memory: 14.0M
         CPU: 108ms
    CGroup: /system.slice/alertmanager.service
            └─116694 /usr/local/bin/alertmanager --config.file /etc/alertmanager/alertmanager.yml --storage.path=/etc/alertmanager/data

déc. 19 00:37:46 srv-supervision systemd[1]: Started AlertManager Serveur Service.
```

Modification de la configuration de Prometheus

Dans le répertoire /etc/prometheus/.

- Ajouter l'alertmanager et le path pour les règles

```
nano /etc/prometheus/prometheus.yml
```

- Créer un dossier pour mettre toutes les règles d'alerting :

```
mkdir /etc/prometheus/alerts
```

[snippet.yaml](#)

```
# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
      - targets:
        - <IP du serveur>:9093 # adresse sur serveur d'alerting

# Load rules once and periodically evaluate them according to the
global evaluation_interval.
rule_files:
  - "alerts/*.yaml" # dossier avec les fichiers de configuration pour
les règles.
```



```
# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
      - targets:
        - localhost:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  - "prometheus_rules.yml"
  # - "first_rules.yml"
  # - "second_rules.yml"
```

Création de la première règle

Dans le répertoire `/etc/prometheus/alerts`.

```
touch general.yml
nano general.yml
```

Pour cet exemple nous allons donner une seule règle, en l'occurrence la vérification des targets. Nous serons alerté dès qu'un service de prometheus (targets) est DOWN.

[snippet.yaml](#)

```
groups:
  - name: GeneralGroup
    rules:
      - alert: InstanceDown
        expr: up == 0
        for: 1m #au bout d'une minute, trigger l'alerte
        labels:
          severity: critical
        annotations:
          summary: "Instance [{{ $labels.instance }}] down"
          description: "[{{ $labels.instance }}] of job [{{ $labels.job }}] has been down for more than 1 minute."
```

Vous pouvez retrouver ici plusieurs modèles de règles pour prometheus :

<https://awesome-prometheus-alerts.grep.to/rules.html>

Il suffit de copier coller les règles dans un fichier `.yaml`.

```
GNU nano 5.4 prometheus.rules.yml
groups:
- name: custom_rules
  rules:
  - record: node_memory_MemFree_percent
    expr: 100 - (100 * node_memory_MemFree_bytes / node_memory_MemTotal_bytes)

  - record: node_filesystem_free_percent
    expr: 100 * node_filesystem_free_bytes(mountpoint="/") / node_filesystem_size_bytes(mountpoint="/")

- name: alert_rules
  rules:
  - alert: InstanceDown
    expr: up == 0
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: "Instance {{{ $labels.instance }}} down"
      description: "[{{{ $labels.instance }}} of job [{{{ $labels.job }}}] has been down for more than 1 minute."
  - alert: DiskSpaceFree10Percent
    expr: node_filesystem_free_percent <= 10
    labels:
```

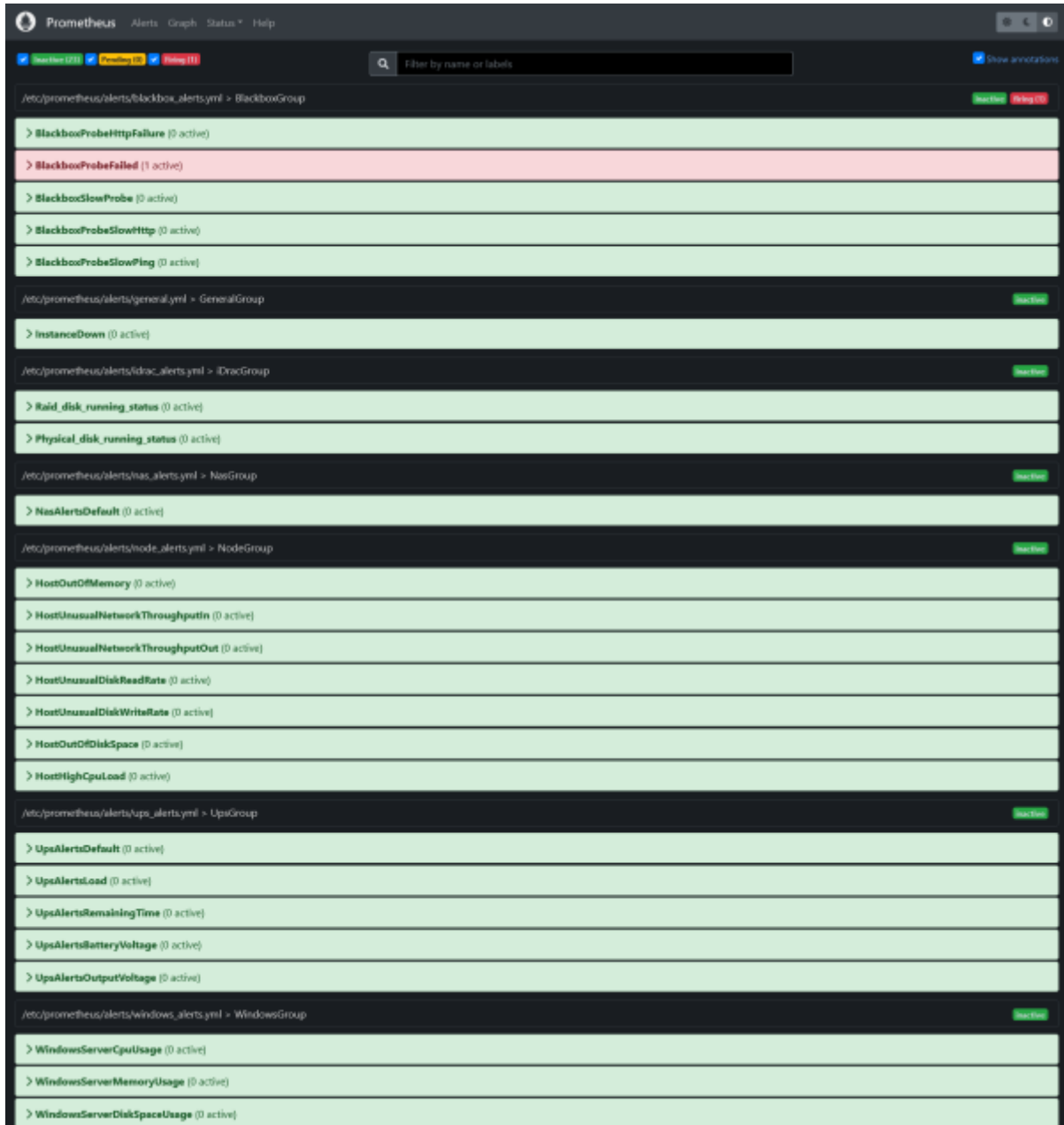
- Vérification du fichier d'alerte avec promtool

```
promtool check rules /etc/prometheus/alerts/general.yml
```

Résultat :

```
Checking prometheus_rules.yml
SUCCESS: 1 rule found
```

Exemple avec plusieurs règles :



Gestion des notifications

<WRAP center round info 60%> Aide pour la création du fichier de configuration : prometheus.io/docs/

La manière la plus connue de prévenir d'un événement est via l'email. Il existe d'autres solutions tels que :

- Via SMS (dans le cas de graves alertes)
- Via Slack (beaucoup utilisé dans les entreprises)
- Via Discord (dans un channel)

Trigger une notification par mail

- Ouvrir le fichier de configuration

```
nano /etc/alertmanager/alertmanager.yml
```

Fichier de configuration avec la fonctionnalité d'**emailing**:

[snippet.yaml](#)

```
global:
  resolve_timeout: 5m

route:
  group_by: ['alertname']
  group_wait: 10s
  group_interval: 30s
  repeat_interval: 1h #envoyer un email toute les heures
  routes:
    - receiver: 'email'
      match_re:
        severity: critical|warning #match le label critical ou warning
        continue: true #continuer d'executer les autres trigger

receivers:
- name: 'email'
  email_configs:
    - to: 'admin-alert@dom.megaprod.lan'
      from: 'admin-alert@dom.megaprod.lan'
      smarthost: 10.192.44.11:25 #ip du serveur relay,, choisir entre
      IPV4 ou un FQDN
      auth_username: 'relais'
      auth_identity: 'relais'
      auth_password: 'Not24get'
      require_tls: false #desactiver l'obligation d'une connexion TLS
  headers:
    From: admin-alert@dom.megaprod.lan #modifier le nom du mail
    affich  sur le client
    send_resolved: true #envoyer un mail lorsque l'alerte est termin e
```

Trigger un webhook Discord

Depuis la derni re version d'Alertmanager (0.25.0), il est possible de trigger un webhook-discord pour g n rer des alertes.

Commencer par ajouter la route dans la configuration d'AlertManager :

[snippet.yaml](#)

```
- receiver: 'discord'
  match_re:
    severity: critical|warning
```

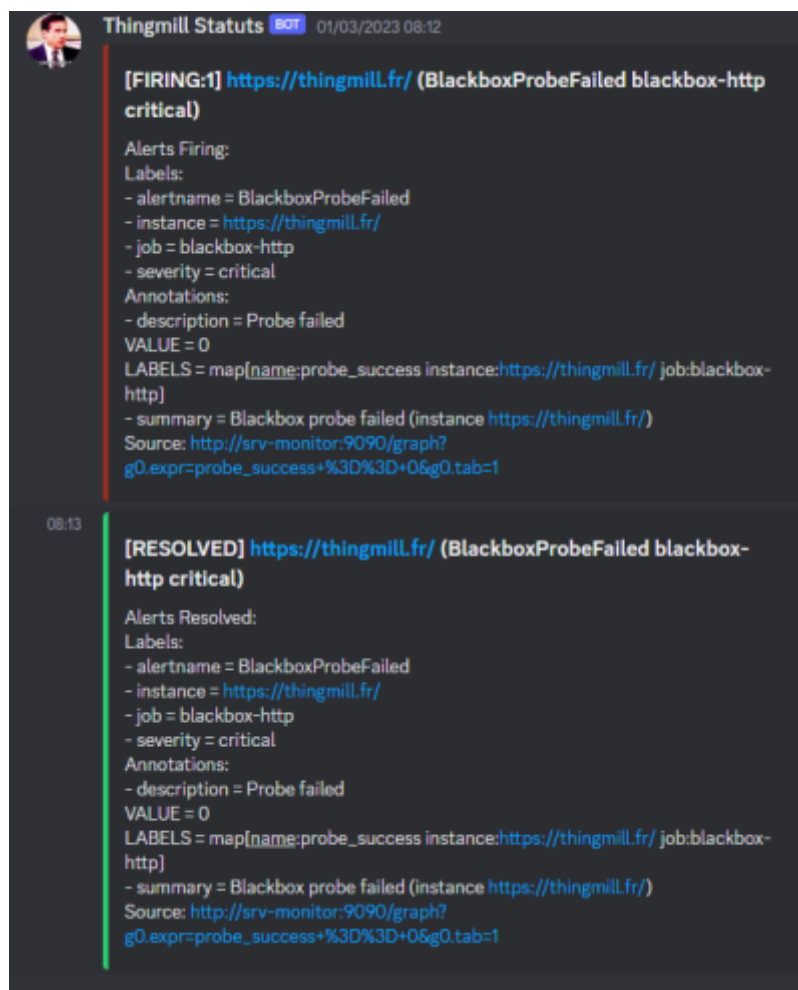
```
continue: true
```

Puis dans les receivers ajouter le webhook discord :

snippet.yaml

```
- name: 'discord'  
  discord_configs:  
  - webhook_url: 'https://discord.com/api/webhooks/XXX/XXX'  
    send_resolved: true
```

Exemple de trigger :



Vérifier le fichier de configuration

```
amtool check-config /etc/alertmanager/alertmanager.yml
```

- Redémarrer le service

```
systemctl restart alertmanager
```

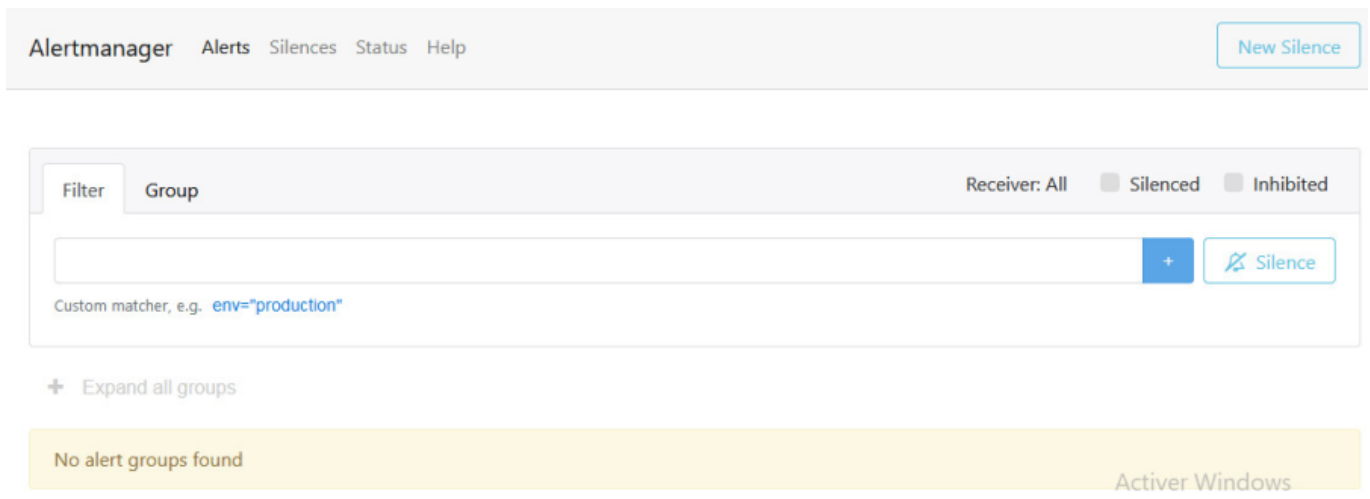
```
GNU nano 5.4
global:
  resolve_timeout: 5m

route:
  group_by: ['alertname']
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 24h
  receiver: 'email'

receivers:
- name: 'email'
  email_configs:
  - to: 'admin-alert@dom.megaprod.lan'
    from: 'admin-alert@dom.megaprod.lan'
    smarthost: 10.192.44.11:25
    auth_username: 'relais'
    auth_identity: 'relais'
    auth_password: 'Not24get'
```

Vérifier le serveur d'alerte

Un serveur web écoute sur le port 9093, accessible sur <http://localhost:9093/#/alerts>



Surveiller les machines avec les agents node et windows exporter

Pour les machines Linux

En se basant sur les modèles d'alertes du site awesome-prometheus-alerts.grep.to, ajouter les règles nécessaires.

- Créer un fichier dans /etc/prometheus/alerts

```
touch /etc/prometheus/alerts/node_exporter_alerts.yml
```

```
nano /etc/prometheus/alerts/node_exporter_alerts.yml
```

- Ajouter les règles

Elles viendront surveiller :

- Si une machine à moins de 10% de ram disponible pendant plus de 2 minutes
- Si les interfaces réseau de l'hôte reçoivent trop de données (> 100 Mo/s)
- Si les interfaces réseau de l'hôte envoient trop de données (> 100 Mo/s)
- Si le disque lit trop de données (> 50 MB/s) pendant 5 minutes
- Si le disque écrit trop de données (> 50 MB/s) pendant 2 minutes
- Si il reste moins de 10% d'espace disque
- Si le processeur de l'hôte est utilisé à plus de 80%

snippet.yaml

```
groups:
  - name: NodeExporterGroup
    rules:
      - alert: HostOutOfMemory
        expr: node_memory_MemAvailable_bytes /
node_memory_MemTotal_bytes * 100 < 10
        for: 2m
        labels:
          severity: warning
        annotations:
          summary: Host out of memory (instance {{ $labels.instance
}})
          description: "Node memory is filling up (< 10% left)\n
VALUE = {{ $value }}\n LABELS = {{ $labels }}"
      - alert: HostUnusualNetworkThroughputIn
        expr: sum by (instance)
(rate(node_network_receive_bytes_total[2m])) / 1024 / 1024 > 100
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: Host unusual network throughput in (instance {{
$labels.instance }})
          description: "Host network interfaces are probably
receiving too much data (> 100 MB/s)\n VALUE = {{ $value }}\n LABELS
= {{ $labels }}"
      - alert: HostUnusualNetworkThroughputOut
        expr: sum by (instance)
(rate(node_network_transmit_bytes_total[2m])) / 1024 / 1024 > 100
        for: 5m
```

```
    labels:
      severity: warning
    annotations:
      summary: Host unusual network throughput out (instance {{
$labels.instance }})
      description: "Host network interfaces are probably sending
too much data (> 100 MB/s)\n VALUE = {{ $value }}\n LABELS = {{
$labels }}"
  - alert: HostUnusualDiskReadRate
    expr: sum by (instance)
(rate(node_disk_read_bytes_total[2m])) / 1024 / 1024 > 50
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: Host unusual disk read rate (instance {{
$labels.instance }})
      description: "Disk is probably reading too much data (> 50
MB/s)\n VALUE = {{ $value }}\n LABELS = {{ $labels }}"
  - alert: HostUnusualDiskWriteRate
    expr: sum by (instance)
(rate(node_disk_written_bytes_total[2m])) / 1024 / 1024 > 50
    for: 2m
    labels:
      severity: warning
    annotations:
      summary: Host unusual disk write rate (instance {{
$labels.instance }})
      description: "Disk is probably writing too much data (> 50
MB/s)\n VALUE = {{ $value }}\n LABELS = {{ $labels }}"
  - alert: HostOutOfDiskSpace
    expr: (node_filesystem_avail_bytes * 100) /
node_filesystem_size_bytes < 10 and ON (instance, device, mountpoint)
node_filesystem_readonly == 0
    for: 2m
    labels:
      severity: warning
    annotations:
      summary: Host out of disk space (instance {{
$labels.instance }})
      description: "Disk is almost full (< 10% left)\n VALUE =
{{ $value }}\n LABELS = {{ $labels }}"
  - alert: HostHighCpuLoad
    expr: sum by (instance) (avg by (mode, instance)
(rate(node_cpu_seconds_total{mode!="idle"}[2m]))) > 0.8
    for: 0m
    labels:
      severity: warning
    annotations:
      summary: Host high CPU load (instance {{ $labels.instance
```



```

}})
      description: "CPU load is > 80%\n VALUE = {{ $value }}\n
LABELS = {{ $labels }}"

```

On retrouve dans Prometheus toutes les alertes pour les machines Linux.

The screenshot shows the Prometheus alerting interface. At the top, it displays the path `/etc/prometheus/alerts/node_exporter_alerts.yml` and the group name `NodeExporterGroup`, with an `inactive` status indicator. Below this, a list of alerts is shown, each with a chevron icon and the text `(0 active)`. The alerts listed are: `HostOutOfMemory`, `HostUnusualNetworkThroughputIn`, `HostUnusualNetworkThroughputOut`, `HostUnusualDiskReadRate`, `HostUnusualDiskWriteRate`, `HostOutOfDiskSpace`, and `HostHighCpuLoad`. A Windows watermark is visible in the bottom right corner of the screenshot.

Pour les machines Windows

En se basant sur les modèles d'alertes du site awesome-prometheus-alerts.grep.to, ajouter les règles nécessaires.

- Créer un fichier dans `/etc/prometheus/alerts`

```
touch /etc/prometheus/alerts/windows_exporter_alerts.yml
```

```
nano /etc/prometheus/alerts/windows_exporter_alerts.yml
```

- Ajouter les règles

Elles viendront surveiller :

- Si le disque dur de l'hôte est utilisé à plus de 80%
- Si la ram de l'hôte est utilisée à plus de 90% pendant 2 minutes
- Si le processeur de l'hôte est utilisé à plus de 80%

[snippet.yaml](#)

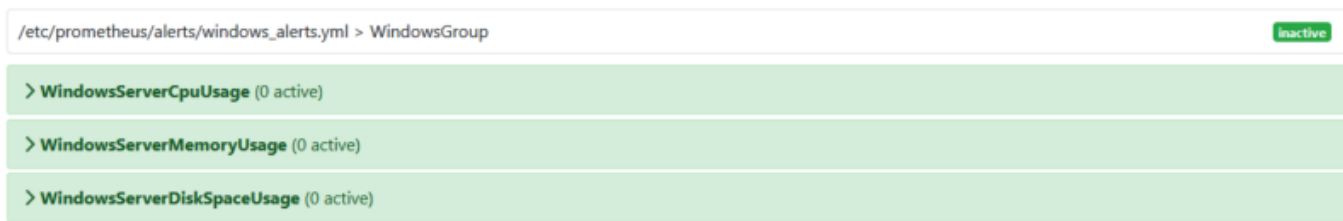
```

groups:
  - name: WindowsExporterGroup
    rules:
      - alert: WindowsServerCpuUsage
        expr: 100 - (avg by (instance)
(rate(windows_cpu_time_total{mode="idle"}[2m])) * 100) > 80
        for: 0m
        labels:

```

```
severity: warning
annotations:
  summary: Windows Server CPU Usage (instance {{
$labels.instance }})
  description: "CPU Usage is more than 80%\n VALUE = {{
$value }}\n LABELS = {{ $labels }}"
- alert: WindowsServerMemoryUsage
  expr: 100 - ((windows_os_physical_memory_free_bytes /
windows_cs_physical_memory_bytes) * 100) > 90
  for: 2m
  labels:
    severity: warning
  annotations:
    summary: Windows Server memory Usage (instance {{
$labels.instance }})
    description: "Memory usage is more than 90%\n VALUE = {{
$value }}\n LABELS = {{ $labels }}"
- alert: WindowsServerDiskSpaceUsage
  expr: 100.0 - 100 * ((windows_logical_disk_free_bytes / 1024
/ 1024 ) / (windows_logical_disk_size_bytes / 1024 / 1024)) > 80
  for: 2m
  labels:
    severity: critical
  annotations:
    summary: Windows Server disk Space Usage (instance {{
$labels.instance }})
    description: "Disk usage is more than 80%\n VALUE = {{
$value }}\n LABELS = {{ $labels }}"
```

On retrouve dans Prometheus toutes les alertes pour les machines Linux.



Agent de monitoring WEB : blackbox_exporter



Qu'est-ce que Blackbox Exporter? Il est utilisé pour sonder les points de terminaison tels que **HTTPS, HTTP, TCP, DNS et ICMP**. Une fois que vous avez défini le point de terminaison, l'exportateur Blackbox génère des centaines de métriques qui peuvent être visualisées à l'aide de Grafana.

Installation de l'agent Blackbox

Changer de répertoire temporaire pour le téléchargement, exemple `/home/adminlocal`.

- Créer le compte de service pour blackbox :

```
useradd --no-create-home --shell /bin/false blackbox_exporter
```

- Télécharger le binary (vérifier la dernière version) :

```
wget  
https://github.com/prometheus/blackbox_exporter/releases/download/v0.23.0/blackbox_exporter-0.23.0.linux-amd64.tar.gz
```

- Extraire et copier le binary

```
tar xvzf blackbox_exporter-0.23.0.linux-amd64.tar.gz
```

```
cp blackbox_exporter-0.23.0.linux-amd64/blackbox_exporter  
/usr/local/bin/blackbox_exporter
```

- Définir les permissions du binary

```
chown blackbox_exporter:blackbox_exporter /usr/local/bin/blackbox_exporter
```

- Créer le dossier pour la configuration :

```
mkdir /etc/blackbox_exporter
```

- Editer ce fichier :

```
/etc/blackbox_exporter/blackbox.yml
```

snippet.yaml

```
modules:
  http_2xx: #vérifier si la réponse est bien 200
    prober: http
    timeout: 5s
  http:
    valid_status_codes: []
    method: GET
    preferred_ip_protocol: ip4 #préferer l'ipv4 pour les requêtes
  icmp_ipv4: #vérifier si une machine répond au ping
    timeout: 5s
    prober: icmp
  icmp:
    preferred_ip_protocol: ip4
#   source_ip_address: "127.0.0.1"
  tcp_connect: #pouvoir ping avec un port
    prober: tcp
```

- Donner la propriété au compte de service

```
chown blackbox_exporter:blackbox_exporter
/etc/blackbox_exporter/blackbox.yml
```

- Créer le service Blackbox

```
/etc/systemd/system/blackbox_exporter.service
```

snippet.bash

```
[Unit]
Description=Blackbox Exporter
Wants=network-online.target
After=network-online.target

[Service]
User=blackbox_exporter
Group=blackbox_exporter
Type=simple
ExecStart=/usr/local/bin/blackbox_exporter --config.file
/etc/blackbox_exporter/blackbox.yml

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

- Démarrer et vérifier le statut du service

```
systemctl daemon-reload
systemctl enable blackbox_exporter
systemctl start blackbox_exporter
systemctl status blackbox_exporter
```

<WRAP center round info 60%> Le service écoute sur le port 9115

Blackbox Exporter

[Probe prometheus.io for http_2xx](#)

[Debug probe prometheus.io for http_2xx](#)

[Metrics](#)

[Configuration](#)

Recent Probes

Module	Target	Result	Debug
tcp_connect	82.127.69.111:25565	Failure	Logs
icmp_ipv4	82.65.195.193	Success	Logs
http_2xx	https://valentinderouet.fr	Success	Logs
tcp_connect	82.127.69.111:25565	Failure	Logs
icmp_ipv4	82.65.195.193	Success	Logs
http_2xx	https://thingmill.fr	Success	Logs
tcp_connect	82.127.69.111:25565	Failure	Logs
icmp_ipv4	82.65.195.193	Success	Logs
http_2xx	https://valentinderouet.fr	Success	Logs

Configuration dans Prometheus

Dans cet exemple nous allons monitorer plusieurs terminaisons :

- ICMP (penser à autoriser la machine pour les requêtes ICMP)
- TCP
- HTTP

Ces trois modules ont été configuré dans le fichier de blackbox.

[snippet.yaml](#)

```
- job_name: "blackbox-http"
  scrape_interval: 10s #vérifier toutes les 10 secondes
  metrics_path: /probe
  params:
    module: [ "http_2xx" ] #fait appel au module blackbox-http
  static_configs:
    - targets:
      - http://intranet.dom.megaprod.lan #monitorer le site intranet
  relabel_configs:
    - source_labels: [__address__]
```

```
target_label: __param_target
- source_labels: [__param_target]
target_label: instance
- target_label: __address__
  replacement: 10.192.43.12:9115 #spécifier l'ip de la machine de supervision
- job_name: 'blackbox-icmp'
  metrics_path: /probe
  params:
    module: [icmp_ipV4]
  scrape_interval: 5s #tester toutes les 5 secondes
  static_configs:
    - targets:
      - 10.192.45.2 #monitorer une IPV4 publique ou privée
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: 10.192.43.12:9115
- job_name: 'blackbox-tcp' #monitorer n'importe quelle IP avec un port
  metrics_path: /probe
  params:
    module: [tcp_connect]
  scrape_interval: 5s #vérifier toutes les 5 secondes
  static_configs:
    - targets:
      - 82.127.69.111:80 #vérifier si le port 80 répond toujours
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: 10.192.43.12:9115
```

Autoriser le ping par blackbox

Vous avez l'erreur :

```
ts=2023-03-09T17:54:36.672432326Z level=error msg="Error listening to socket: listen ip4:icmp 0.0.0.0: socket: operation not permitted" file=icmp.go line=32
```

<WRAP center round important 60%> Pour que blackbox_exporter puisse exécuter des ping, il faut

changer les capabilités du binary pour qu'il accède à la fonction.

- Se positionner dans `/usr/local/bin`

```
cd /usr/local/bin
```

- Changer les capabilités du binary de blackbox :

[snippet.bash](#)

```
setcap cap_net_raw+ep blackbox_exporter
```

Avec ce paramètre, l'exportateur fonctionnera correctement sans `suid/uid=0`.

```
ts=2023-03-09T17:54:36.675846592Z caller=handler.go:117 module=icmp_ipv4 target=10.192.45.2 level=info msg="Creating ICMP packet" seq=30185 id=1835
ts=2023-03-09T17:54:36.675883277Z caller=handler.go:117 module=icmp_ipv4 target=10.192.45.2 level=info msg="Writing out packet"
ts=2023-03-09T17:54:36.675894829Z caller=handler.go:117 module=icmp_ipv4 target=10.192.45.2 level=debug msg="Setting TTL (IPv4 unprivileged)" ttl=64
ts=2023-03-09T17:54:36.676054984Z caller=handler.go:117 module=icmp_ipv4 target=10.192.45.2 level=info msg="Waiting for reply packets"
ts=2023-03-09T17:54:36.676673357Z caller=handler.go:117 module=icmp_ipv4 target=10.192.45.2 level=info msg="Found matching reply packet"
ts=2023-03-09T17:54:36.676704347Z caller=main.go:181 module=icmp_ipv4 target=10.192.45.2 level=info msg="Probe succeeded" duration_seconds=0.004215907
```

Une fois la configuration validée, les services sont en ligne :

blackbox-http (1/1 up) [show less](#)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.192.43.12:9115/probe <code>module="http_2xx"</code> <code>target="http://intranet.dom.megaprod.lan"</code>	UP	<code>instance="http://intranet.dom.megaprod.lan"</code> <code>job="blackbox-http"</code>	4.59s ago	5.412ms	

blackbox-icmp (1/1 up) [show less](#)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.192.43.12:9115/probe <code>module="icmp_ipv4"</code> <code>target="82.127.69.111"</code>	UP	<code>instance="82.127.69.111"</code> <code>job="blackbox-icmp"</code>	3.127s ago	3.308ms	

blackbox-tcp (1/1 up) [show less](#)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.192.43.12:9115/probe <code>module="tcp_connect"</code> <code>target="82.127.69.111:80"</code>	UP	<code>instance="82.127.69.111:80"</code> <code>job="blackbox-tcp"</code>	171.000ms ago	23.709ms	

Création des alertes

En se basant sur les modèles d'alertes du site awesome-prometheus-alerts.grep.to, ajouter les règles nécessaires.

- Créer un fichier dans `/etc/prometheus/alerts`

```
touch /etc/prometheus/alerts/blackbox_alerts.yml
```

```
nano /etc/prometheus/alerts/blackbox_alerts.yml
```

- Ajouter les règles

Elles viendront surveiller :

- Si une probe blackbox tombe
- Si une probe prend du temps à répondre
- Si une probe ne retourne pas une réponse HTTP entre 200 et 399
- Si un certificat expire dans moins de 3 jours
- Si une requête HTTP prend plus d'1 seconde
- Si le ping prend plus d'1 seconde

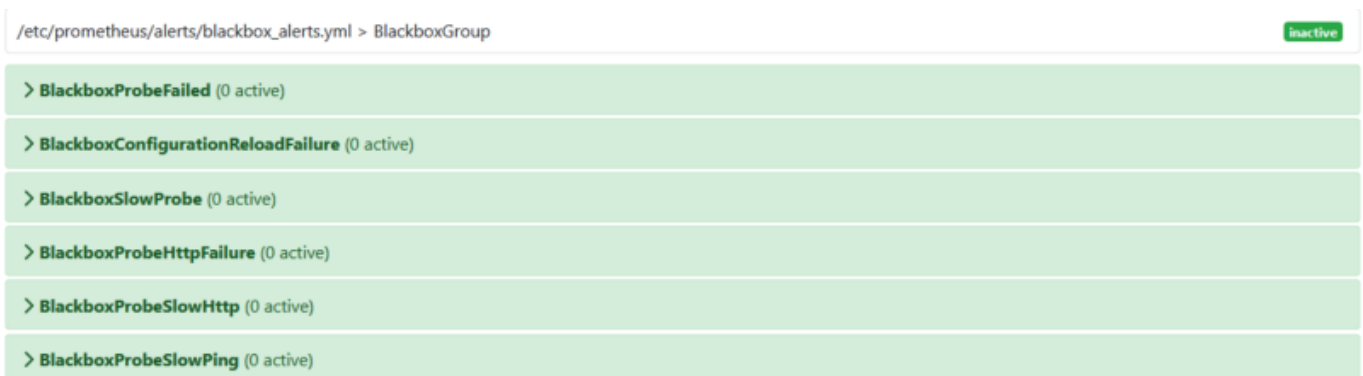
[snippet.yaml](#)

```
groups:
  - name: BlackboxGroup
    rules:
      - alert: BlackboxProbeFailed
        expr: probe_success == 0
        for: 0m
        labels:
          severity: critical
        annotations:
          summary: Blackbox probe failed (instance {{
$labels.instance }})
          description: "Probe failed\n VALUE = {{ $value }}\n
LABELS = {{ $labels }}"
      - alert: BlackboxConfigurationReloadFailure
        expr: blackbox_exporter_config_last_reload_successful != 1
        for: 0m
        labels:
          severity: warning
        annotations:
          summary: Blackbox configuration reload failure (instance {{
$labels.instance }})
          description: "Blackbox configuration reload failure\n
VALUE = {{ $value }}\n LABELS = {{ $labels }}"
      - alert: BlackboxSlowProbe
        expr: avg_over_time(probe_duration_seconds[1m]) > 1
        for: 1m
        labels:
          severity: warning
        annotations:
          summary: Blackbox slow probe (instance {{ $labels.instance
}})
          description: "Blackbox probe took more than 1s to
complete\n VALUE = {{ $value }}\n LABELS = {{ $labels }}"
      - alert: BlackboxProbeHttpFailure
        expr: probe_http_status_code <= 199 OR probe_http_status_code
>= 400
```



```
for: 0m
labels:
  severity: critical
annotations:
  summary: Blackbox probe HTTP failure (instance {{
$labels.instance }})
  description: "HTTP status code is not 200-399\n VALUE = {{
$value }}\n LABELS = {{ $labels }}"
- alert: BlackboxProbeSlowHttp
  expr: avg_over_time(probe_http_duration_seconds[1m]) > 1
  for: 1m
  labels:
    severity: warning
  annotations:
    summary: Blackbox probe slow HTTP (instance {{
$labels.instance }})
    description: "HTTP request took more than 1s\n VALUE = {{
$value }}\n LABELS = {{ $labels }}"
- alert: BlackboxProbeSlowPing
  expr: avg_over_time(probe_icmp_duration_seconds[1m]) > 1
  for: 1m
  labels:
    severity: warning
  annotations:
    summary: Blackbox probe slow ping (instance {{
$labels.instance }})
    description: "Blackbox ping took more than 1s\n VALUE = {{
$value }}\n LABELS = {{ $labels }}"
```

On retrouve dans Prometheus toutes les alertes pour l'agent Blackbox

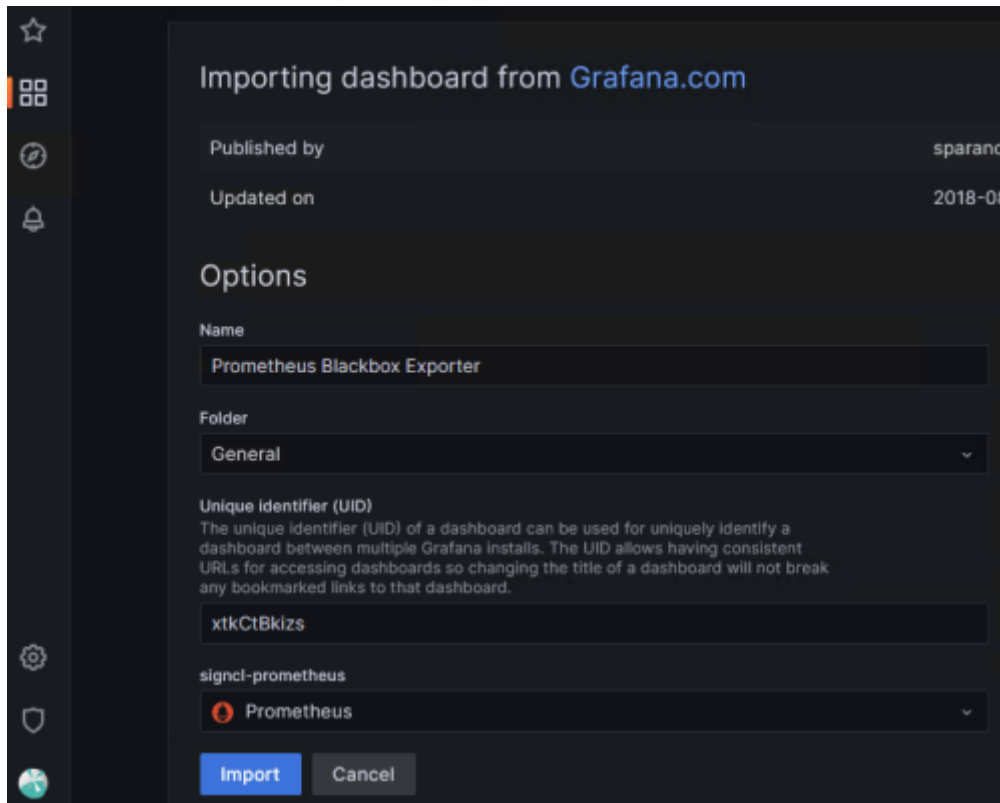


Ajout du tableau dans Grafana

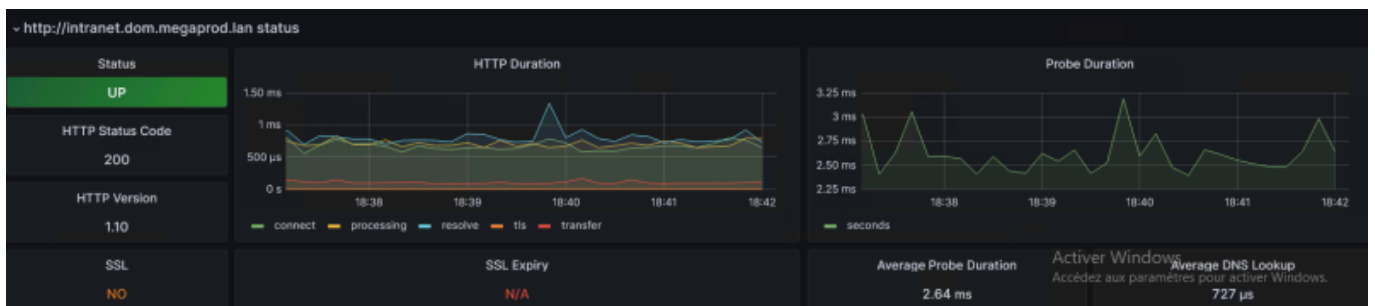
Il existe un tableau pour traiter les données de blackbox_exporter :

- 7587 (<https://grafana.com/grafana/dashboards/7587>)

Il suffit donc de l'importer dans Grafana.



Une fois le tableau importé, les valeurs sont affichées :



<WRAP center round important 60%> Penser à autoriser les réponses aux ping dans le pare-feu windows !

```
netsh firewall set icmpsetting 8
```

Agent de monitoring SQL : mysqld_exporter



Il est possible de superviser des bases de données SQL avec le service Prometheus, l'installation repose sur le même principe que les autres agents.

Installation de l'agent

Sur la machine dont le serveur SQL est installé

- Création du groupe et de l'utilisateur

```
groupadd --system prometheus
useradd -s /sbin/nologin --system -g prometheus prometheus
```

- Télécharger l'archive et créer le service

Cette opération doit être effectuée sur les serveurs MySQL / MariaDB, qu'ils soient esclaves ou maîtres. Il se peut que vous deviez consulter la page Prometheus MySQL exporter releases pour connaître la dernière version, puis exporter la dernière version vers la variable VER comme indiqué ci-dessous :

- Télécharger la dernière version de l'exportateur MySQL :

```
curl -s
https://api.github.com/repos/prometheus/mysqld_exporter/releases/latest |
grep browser_download_url | grep linux-amd64 | cut -d '"' -f 4 | wget -
qi -
```

- Extraire et définir les permissions sur `mysqld_exporter`

(spécifier la bonne version à la place de *)

```
tar xvf mysqld_exporter*.tar.gz
mv mysqld_exporter-*.linux-amd64/mysqld_exporter /usr/local/bin/
chmod +x /usr/local/bin/mysqld_exporter
```

Créer les accès sur la base SQL

- Se connecter sur la BDD

```
mysql -h localhost -u root -p
```

- Créer l'utilisateur et ajouter les droits

L'utilisateur doit disposer des autorisations PROCESS, SELECT, REPLICATION et CLIENT :

[snippet.sql](#)

```
CREATE USER 'mysqld_exporter'@'localhost' IDENTIFIED BY 'Not24get' WITH  
MAX_USER_CONNECTIONS 2;  
GRANT PROCESS, REPLICATION CLIENT, SELECT ON *.* TO  
'mysqld_exporter'@'localhost';  
FLUSH PRIVILEGES;  
EXIT
```

Configurer la base de données

- Éditer le fichier `.mysqld_exporter.cnf`

```
/etc/.mysqld_exporter.cnf
```

```
[client]  
user=mysqld_exporter  
password=Not24get
```

- Changer le propriétaire du fichier de configuration

```
chown root:prometheus /etc/.mysqld_exporter.cnf
```

Création du service

- Créer le fichier dans systemd

```
nano /etc/systemd/system/mysqld_exporter.service
```

[snippet.bash](#)

```
[Unit]
```

```
Description=Prometheus MySQL Exporter
After=network.target
User=prometheus
Group=prometheus

[Service]
Type=simple
Restart=always
ExecStart=/usr/local/bin/mysqld_exporter \
--config.my-cnf /etc/.mysqld_exporter.cnf \
--collect.global_status \
--collect.info_schema.innodb_metrics \
--collect.auto_increment.columns \
--collect.info_schema.processlist \
--collect.binlog_size \
--collect.info_schema.tablestats \
--collect.global_variables \
--collect.info_schema.query_response_time \
--collect.info_schema.userstats \
--collect.info_schema.tables \
--collect.perf_schema.tablelocks \
--collect.perf_schema.file_events \
--collect.perf_schema.eventswaits \
--collect.perf_schema.indexiowaits \
--collect.perf_schema.tableiowaits \
--collect.slave_status \
--web.listen-address=10.192.43.11:9104

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

- Activer le service et vérifier son statut

```
systemctl daemon-reload
systemctl enable mysql_exporter
systemctl start mysql_exporter
```

```
systemctl status mysql_exporter
```

Ajout de l'agent dans Prometheus

Dans le fichier `/etc/prometheus/prometheus.yml`

[snippet.yaml](#)

```
scrape_configs:
```

```
- job_name: 'server1_db'  
  static_configs:  
    - targets:  
      - 10.192.43.11:9104  
      labels:  
        alias: db1
```

<WRAP center round info 60%> Le serveur Prometheus doit être en mesure d'atteindre les cibles sur le réseau. Veillez à ce que la configuration du réseau et du pare-feu soit correcte.

Redémarrer le service prometheus

Création des alertes

En se basant sur les modèles d'alertes du site awesome-prometheus-alerts.grep.to, ajouter les règles nécessaires.

- Créer un fichier dans /etc/prometheus/alerts

```
touch /etc/prometheus/alerts/mysql_alerts.yml
```

```
nano /etc/prometheus/alerts/mysql_alerts.yml
```

- Ajouter les règles

Elles viendront surveiller :

- Si le serveur SQL a redémarré
- Si il prend du temps à répondre
- Si une requête prend trop de temps
- Si il y a trop de connexions
- Si le serveur SQL est DOWN

[snippet.yaml](#)

```
groups:  
  - name: MySQLGroup  
    rules:  
      - alert: MysqlDown  
        expr: mysql_up == 0  
        for: 0m  
        labels:  
          severity: critical  
        annotations:  
          summary: MySQL down (instance {{ $labels.instance }})  
          description: "MySQL instance is down on {{ $labels.instance }}\n VALUE = {{ $value }}\n LABELS = {{ $labels }}"
```

```
- alert: MysqlTooManyConnections(>80%)
  expr: max_over_time(mysql_global_status_threads_connected[1m])
/ mysql_global_variables_max_connections * 100 > 80
  for: 2m
  labels:
    severity: warning
  annotations:
    summary: MySQL too many connections (> 80%) (instance {{
$labels.instance }})
    description: "More than 80% of MySQL connections are in use
on {{ $labels.instance }}\n VALUE = {{ $value }}\n LABELS = {{
$labels }}"
- alert: MysqlHighThreadsRunning
  expr: max_over_time(mysql_global_status_threads_running[1m]) /
mysql_global_variables_max_connections * 100 > 60
  for: 2m
  labels:
    severity: warning
  annotations:
    summary: MySQL high threads running (instance {{
$labels.instance }})
    description: "More than 60% of MySQL connections are in
running state on {{ $labels.instance }}\n VALUE = {{ $value }}\n
LABELS = {{ $labels }}"
- alert: MysqlSlowQueries
  expr: increase(mysql_global_status_slow_queries[1m]) > 0
  for: 2m
  labels:
    severity: warning
  annotations:
    summary: MySQL slow queries (instance {{ $labels.instance }})
    description: "MySQL server mysql has some new slow query.\n
VALUE = {{ $value }}\n LABELS = {{ $labels }}"
- alert: MysqlRestarted
  expr: mysql_global_status_uptime < 60
  for: 0m
  labels:
    severity: info
  annotations:
    summary: MySQL restarted (instance {{ $labels.instance }})
    description: "MySQL has just been restarted, less than one
minute ago on {{ $labels.instance }}.\n VALUE = {{ $value }}\n
LABELS = {{ $labels }}"
```

On retrouve dans Prometheus toutes les alertes pour le SQL

/etc/prometheus/alerts/mysql_alerts.yml > MySQLGroup inactive

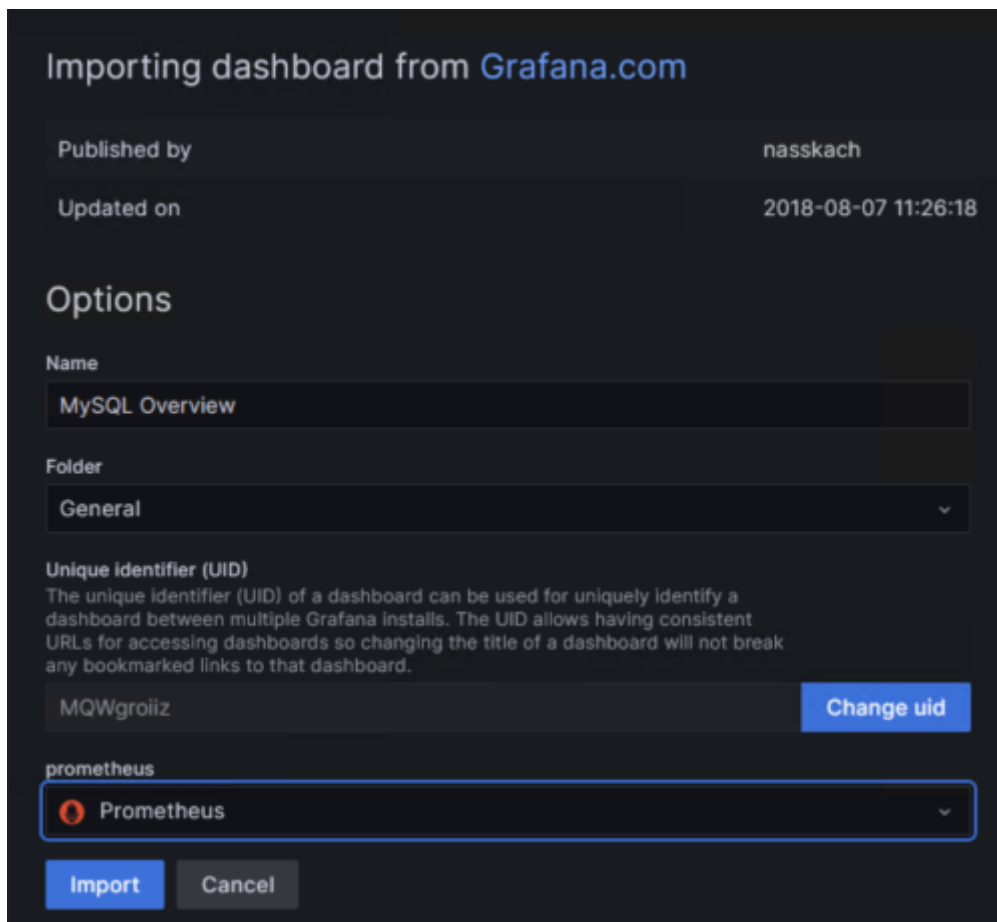
- > **MySQLDown** (0 active)
- > **MySQLTooManyConnections(> 80%)** (0 active)
- > **MySQLHighThreadsRunning** (0 active)
- > **MySQLSlowQueries** (0 active)
- > **MySQLRestarted** (0 active)

Ajout du tableau dans Grafana

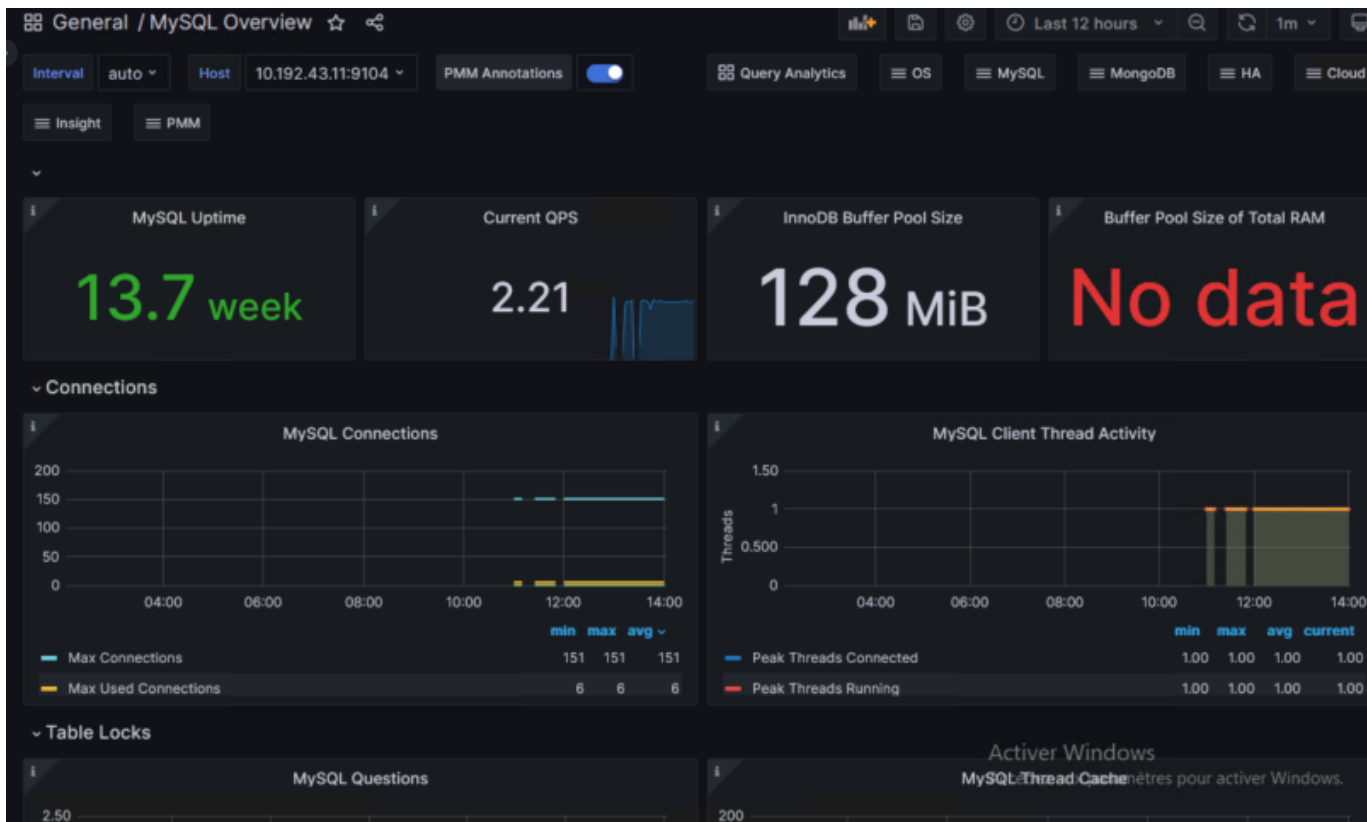
Il existe un tableau pour traiter les données de mysqld_exporter:

- 7362 <https://grafana.com/grafana/dashboards/7362-mysql-overview/>

Il suffit donc de l'importer dans Grafana.



Une fois le tableau importé, les valeurs sont affichées :



Agent de monitoring SNMP : snmp_exporter



Le protocole SNMP **permet à une application de gestion de demander des informations provenant d'une unité gérée**. L'unité gérée contient un logiciel qui envoie et reçoit des informations SNMP. Ce module logiciel est généralement appelé agent SNMP. Cet exportateur est le moyen recommandé pour exposer les données SNMP dans un format que Prometheus peut intégrer.

Dans notre cas nous souhaitons monitorer un UPS de la marque Eaton en SNMP.

Installation de l'agent

Téléchargement de la dernière version

- Télécharger la dernière version de snmp_exporter :

```
wget  
https://github.com/prometheus/snmp_exporter/releases/download/v0.21.0/snmp_exporter-0.21.0.linux-amd64.tar.gz
```

- Extraire et définir les permissions sur snmp_exporter

```
tar xzf snmp_exporter-0.21.0.linux-amd64.tar.gz  
cd snmp_exporter-0.21.0.linux-amd64  
cp ./snmp_exporter /usr/local/bin/snmp_exporter  
cp ./snmp.yml /usr/local/bin/snmp.yml  
cd /usr/local/bin/  
chmod +x /usr/local/bin/snmp_exporter
```

Création du service

- Créer le fichier dans systemd

```
nano /etc/systemd/system/snmp-exporter.service
```

snippet.bash

```
[Unit]  
Description=Prometheus SNMP Exporter Service  
After=network.target  
  
[Service]  
Type=simple  
User=prometheus  
ExecStart=/usr/local/bin/snmp_exporter --  
config.file="/usr/local/bin/snmp.yml"  
  
[Install]  
WantedBy=multi-user.target
```

- Activer le service et vérifier son statut

```
systemctl daemon-reload  
systemctl enable snmp-exporter  
systemctl start snmp-exporter
```

```
systemctl status snmp-exporter
```

<WRAP center round info 60%> Le serveur web de snmp_exporter écoute sur le port 9116. Le

protocole SNMP utilise les ports 160 et 161 en UDP pour communiquer, penser à les autoriser dans votre pare-feu.

Configuration de l'agent

- Editer le fichier de configuration de prometheus pour ajouter l'UPS en target

```
nano /etc/prometheus/prometheus.yml
```

snippet.yaml

```
- job_name: 'ups'
  scrape_interval: 120s #récupérer les informations toutes les 2
minutes
  scrape_timeout: 120s
  # SNMP device.
  metrics_path: /snmp
  params:
    module: [rfc1628_ups] #MIB pour l'UPS ; que l'ont va générer
après
  static_configs:
    - targets:
      - 10.192.20.10 #adresse ip de l'UPS
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: 10.192.43.12:9116 #adresse et port de
snmp_exporter
```

- Vérifier la configuration du fichier Prometheus

```
promtool check config /etc/prometheus/prometheus.yml
```

Génération du fichier de configuration avec des MIBS personnelles

Le plugin snmp-exporter intègre un logiciel pour compiler des MIBS dans un fichier .yaml. En effet, le plugin fonctionne par module, un module correspond à une MIB. Ce générateur de configuration utilise NetSNMP pour analyser les MIB, et génère des configurations pour le snmp_exporter qui les utilise.

- Installation des dépendances

```
apt-get install unzip build-essential libsnmp-dev
```

- Cloner le repo

```
git clone https://github.com/prometheus/snmp_exporter.git
```

- Se placer dans le repo github de snmp_exporter, puis dans generator

```
cd snmp_exporter/generator
```

- Ajout des mibs dans le dossier /mibs

Nous nous basons sur la MIB [RFC1628UPS-MIB](#) qui est compatible avec les cartes Network MS de chez Eaton.



Pack de MIBS : eaton.com.

```
wget https://raw.githubusercontent.com/cliv/rfc1628_ups_prometheus_module/master/RFC1628UPS-MIB
mv RFC1628UPS-MIB mibs
```

- Modification du fichier generator.yml

```
nano generator.yml
```

Ajouter à la toute fin du fichier :

[snippet.yaml](#)

```
rfc1628_ups:
  version: 1
  walk:
    - sysUpTime
    - interfaces
    # Use OIDs to avoid conflict with APCUPS if using with
    # prometheus' default generator.yml
```

```
# Comment out anything you don't want prometheus to query
- 1.3.6.1.2.1.33.1.1 # upsIdent
- 1.3.6.1.2.1.33.1.2 # upsBattery
- 1.3.6.1.2.1.33.1.3 # upsInput
- 1.3.6.1.2.1.33.1.4 # upsOutput
- 1.3.6.1.2.1.33.1.5 # upsBypass
- 1.3.6.1.2.1.33.1.6 # upsAlarm
- 1.3.6.1.2.1.33.1.7 # upsTest
- 1.3.6.1.2.1.33.1.8 # upsControl
- 1.3.6.1.2.1.33.1.9 # upsConfig
lookups:
- source_indexes: [ifIndex]
  # Use OID to avoid conflict with ifDescr in other modules
  lookup: 1.3.6.1.2.1.2.2.1.2
  drop_source_indexes: true
overrides:
  ifType:
    type: EnumAsInfo
  upsBatteryStatus:
    type: EnumAsStateSet
  upsOutputSource:
    type: EnumAsStateSet
  upsTestResultsSummary:
    type: EnumAsStateSet
  upsShutdownType:
    type: EnumAsStateSet
  upsAutoRestart:
    type: EnumAsStateSet
  upsConfigAudibleStatus:
    type: EnumAsStateSet
```

- Compiler le générateur

```
make generator mibs
```

- Générer le fichier avec le générateur de fichier snmp.yml

```
make generate
```

- Copie du fichier généré dans snmp-exporter

```
cp snmp.yml /usr/local/bin/
```

- Redémarrer le service

```
systemctl restart snmp-exporter
```

Essais de récupération des traps SNMP

<WRAP center round important 60%> Pensez à activer le SNMP v1 et de définir la communauté sur public.

Configuration SNMP	
5130 RT 3000	
Version SNMP :	V1&V3
<input type="checkbox"/> Port SNMP :	161
Configuration SNMP V1	
Communauté en lecture seule :	public
Ecriture SNMP :	Désactivé
Communauté en lecture-écriture :	private

Rendez vous sur ipduseurveursupervision:9116 et essayer d'appeler le module rfc1628_ups avec l'IP de l'onduleur.

SNMP Exporter

Target:

Module:

[Config](#)

Exécuter et vous devez avoir des metrics de l'onduleur :

```
# HELP ifAdminStatus The desired state of the interface - 1.3.6.1.2.1.2.2.1.7
# TYPE ifAdminStatus gauge
ifAdminStatus{ifDescr="eth0"} 1
ifAdminStatus{ifDescr="lo"} 1
# HELP ifDescr A textual string containing information about the interface - 1.3.6.1.2.1.2.2.1.2
# TYPE ifDescr gauge
ifDescr{ifDescr="eth0"} 1
ifDescr{ifDescr="lo"} 1
# HELP ifInDiscards The number of inbound packets which were chosen to be discarded even though no errors had been detected to prevent
protocol - 1.3.6.1.2.1.2.2.1.13
# TYPE ifInDiscards counter
ifInDiscards{ifDescr="eth0"} 0
ifInDiscards{ifDescr="lo"} 0
# HELP ifInErrors For packet-oriented interfaces, the number of inbound packets that contained errors preventing them from being del
1.3.6.1.2.1.2.2.1.14
# TYPE ifInErrors counter
ifInErrors{ifDescr="eth0"} 0
ifInErrors{ifDescr="lo"} 0
# HELP ifInMulticastPkts The number of packets, delivered by this sub-layer to a higher (sub-)layer, which were addressed to a multica
1.3.6.1.2.1.2.2.1.12
# TYPE ifInMulticastPkts counter
ifInMulticastPkts{ifDescr="eth0"} 223381
ifInMulticastPkts{ifDescr="lo"} 0
# HELP ifInOctets The total number of octets received on the interface, including framing characters - 1.3.6.1.2.1.2.2.1.10
# TYPE ifInOctets counter
ifInOctets{ifDescr="eth0"} 5.11436033e+08
ifInOctets{ifDescr="lo"} 1.064147741e+09
# HELP ifInUcastPkts The number of packets, delivered by this sub-layer to a higher (sub-)layer, which were not addressed to a mult
1.3.6.1.2.1.2.2.1.11
# TYPE ifInUcastPkts counter
ifInUcastPkts{ifDescr="eth0"} 4.028048e+06
ifInUcastPkts{ifDescr="lo"} 1.714211e+06
# HELP ifInUnknownProtos For packet-oriented interfaces, the number of packets received via the interface which were discarded becau
1.3.6.1.2.1.2.2.1.15
# TYPE ifInUnknownProtos counter
```

Ajout et test de requêtage dans Prometheus

Nous avons précédemment modifier le fichier prometheus .yml sans redémarrer le service.

Vous pouvez désormais le redémarrer :

```
systemctl restart prometheus
```

Il doit désormais apparaître dans les targets l'UPS :

ups (1/1 up) [show test](#)

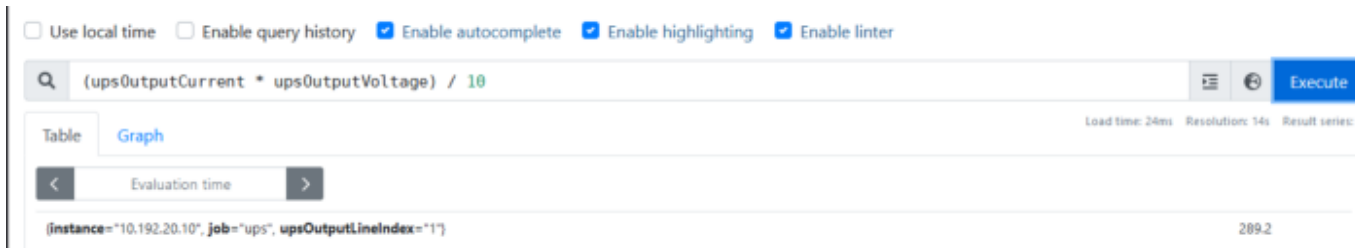
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.192.100.203:9116/snmp module="rfc1028_ups" target="10.192.20.10"	UP	instance="10.192.20.10" job="ups"	1m 31s ago	571.368ms	

Rédaction d'une requête PromQL de test :

Nous souhaitons calculer la puissance en Watts de sorties de l'onduleur :

$$(upsOutputCurrent * upsOutputVoltage) / 10$$

Résultat : la réponse est **289W** en instantané.



Création d'alertes en lien avec les traps SNMP

Nous avons vu précédemment comment faire une requête PromQL pour aller chercher une valeur précise. Nous allons réutiliser ces techniques pour générer des règles.

- Créer un fichier dans `/etc/prometheus/alerts`

```
touch /etc/prometheus/alerts/ups_alerts.yml
```

```
nano /etc/prometheus/alerts/ups_alerts.yml
```

- Ajouter les règles

Elles viendront surveiller :

- Si il y a une alarme sur l'onduleur
- Si la charge de l'onduleur est supérieur à 11% pendant 2 minutes
- Si le temps restant sur batterie est inférieur à 25 minutes
- Si la tension des batteries sont inférieurs à 75V
- Si la tension de sortie est supérieur à 246V

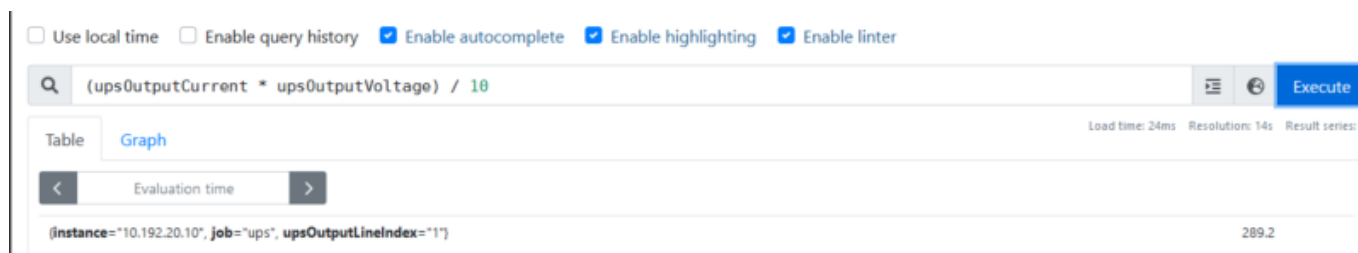
snippet.yaml

```
groups:
  - name: UpsGroup
    rules:
      - alert: UpsAlertsDefault
        expr: upsAlarmsPresent == 2
        for: 0s
        labels:
          severity: critical
        annotations:
          summary: Défaut onduleur
          description: "L'onduleur à une alarme, vérifier au plus vite."
      - alert: UpsAlertsLoad
        expr: upsOutputPercentLoad > 11
        for: 2m
        labels:
          severity: critical
        annotations:
```



```
summary: Défaut puissance onduleur
description: "L'onduleur présente une charge anormale
supérieure à 11%"
- alert: UpsAlertsRemainingTime
  expr: upsEstimatedMinutesRemaining < 25
  for: 0s
  labels:
    severity: critical
  annotations:
    summary: Défaut onduleur temps restant secteur
    description: "L'onduleur dispose moins de 25 minutes de
disponibilité"
- alert: UpsAlertsBatteryVoltage
  expr: upsBatteryVoltage / 10 < 75
  for: 0s
  labels:
    severity: critical
  annotations:
    summary: Défaut onduleur tension batterie
    description: "Les batteries de l'onduleur sont à moins de
75V"
- alert: UpsAlertsOutputVoltage
  expr: upsOutputVoltage > 246
  for: 0s
  labels:
    severity: critical
  annotations:
    summary: Défaut onduleur tension secteur sortie
    description: "L'onduleur délivre une tension supérieur à
246V' "
```

On retrouve dans Prometheus toutes les alertes pour l'onduleur.



Nous verrons par la suite comment créer un tableau sur mesure pour traiter les données.

Création d'un tableau Grafana sur mesure

La documentation de Grafana est disponible ici :

<https://grafana.com/docs/grafana/latest/getting-started/build-first-dashboard/>.

Définir le besoin des données

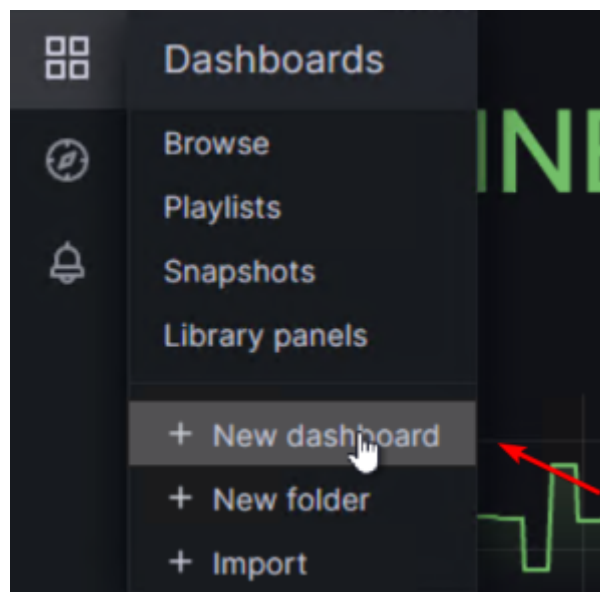
Nous souhaitons récupérer un maximum d'information sur l'onduleur et pouvoir les traiter.

Comme par exemple :

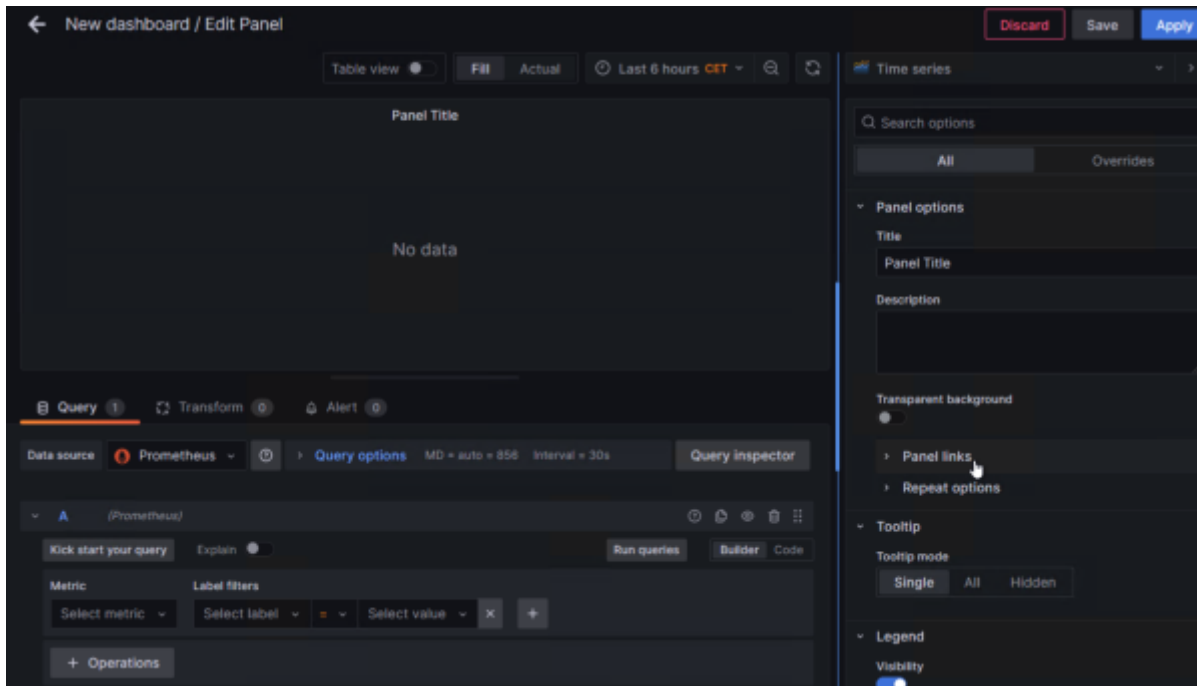
- La tension d'entrée/sortie
- La fréquence d'entrée/sortie
- La charge de l'onduleur
- Calculer le coût moyen de l'infrastructure par mois en direct

Créer un nouveau tableau vierge

- Créer un nouveau tableau vierge depuis Grafana



- Ajouter le premier panel



Rédiger les requêtes PromQL

Pour exemple nous souhaitons avoir :

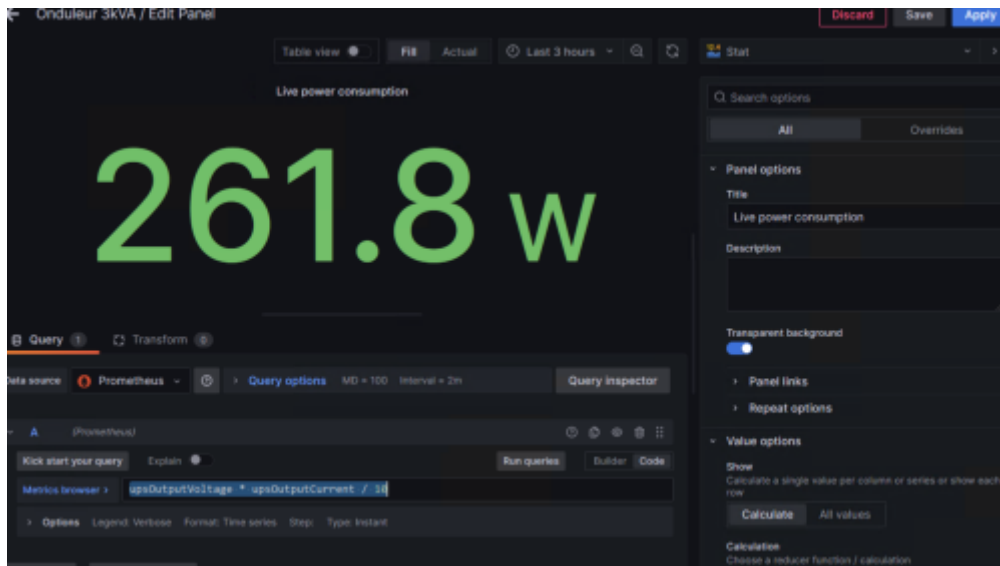
- La consommation en direct de l'infrastructure
- La consommation en kWh
- Le coût moyen par mois

La consommation en direct de l'infrastructure

$$P = U * I$$

```
upsOutputVoltage * upsOutputCurrent / 10
```

Puis choisir l'unité pour W (pour le formatage de la donnée).



La consommation en kWh

$$P = (\text{heures} * \text{jours} * P) / 1000$$

$$(24 * 365 * (\text{upsOutputVoltage} * \text{upsOutputCurrent} / 10)) / 1000$$

Le coût moyen par mois

En prenant compte que le prix moyen du kWh fournit par EDF est de 0,18 centimes.

$$(((24 * 365 * (\text{upsOutputVoltage} * \text{upsOutputCurrent} / 10)) / 1000) * 0.18) / 12$$

Essais et exemple

Après avoir ajouter toutes les requêtes dans les panels, nous avons un tableau exploitable qui permet d'avoir en un coup d'oeil les défauts électriques de l'installation.



Supervision active : prometheus_am_executor

Prometheus-am-executor est un serveur HTTP qui reçoit des alertes du Prometheus Alertmanager et exécute une commande donnée avec les détails de l'alerte définis en tant que variables d'environnement.

Installation de l'agent

Installation des dépendances

- Installation de git :

```
apt install git -y
```

- Installation de GO



Retrouvez la doc de GO ici : <https://go.dev/doc/install>

Version actuelle de GO : **1.20.2**

```
wget https://go.dev/dl/go1.20.2.linux-amd64.tar.gz
```

- **Supprimez toute installation antérieure de Go** en supprimant le dossier `/usr/local/go` (s'il existe), puis extrayez l'archive que vous venez de télécharger dans `/usr/local`, créant ainsi une nouvelle arborescence Go dans `/usr/local/go` :

```
rm -rf /usr/local/go && tar -C /usr/local -xzf go1.20.2.linux-amd64.tar.gz
```

Ne pas décompressez l'archive dans une arborescence `/usr/local/go` existante. Ceci est connu pour produire des installations Go cassées.

- Ajoutez `/usr/local/go/bin` à la variable d'environnement `PATH`.

Vous pouvez le faire en ajoutant la ligne suivante à votre `$HOME/.profile` ou `/etc/profile` (pour une installation sur l'ensemble du système) :

```
export PATH=$PATH:/usr/local/go/bin
```

- Tester votre version de GO

```
go version
```

Installation de Prometheus AM executor

- Cloner le repo de l'agent

```
git clone https://github.com/imgix/prometheus-am-executor.git
```

- Télécharger les dépendances

```
go test -count 1 -v ./...
```

- Compiler le binary avec GO

```
go build
```

- Copier le binary dans `/usr/local/bin/`

```
cp prometheus-am-executor /usr/local/bin/am-executor  
chmod +x /usr/local/bin/am-executor
```

- Créer le répertoire de configuration

```
mkdir /etc/prometheus/am-executor
```

Création du service

- Créer le fichier dans systemd

```
nano /etc/systemd/system/prometheus-am-executor.service
```

snippet.bash

```
[Unit]
Description=Prometheus script executor
Documentation=https://github.com/imgix/prometheus-am-executor

[Service]
Restart=always
ExecStart=/usr/local/bin/am-executor -v -l 10.192.43.12:9118 -f
/etc/prometheus/am-executor/config.yml
ExecReload=/bin/kill -HUP $MAINPID
TimeoutStopSec=20s
SendSIGKILL=no

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

- Activer le service et vérifier son statut

```
systemctl daemon-reload
systemctl enable prometheus-am-executor
systemctl start prometheus-am-executor
```

```
systemctl status prometheus-am-executor
```

Une erreur sera présente car le fichier configuration n'est pas encore créé.

Configuration de l'agent

Créer le fichier de configuration d'AM executor

```
nano /etc/prometheus/am-executor/config.yml
```

snippet.yaml

```
#port d'écoute d'AlertManager
listen_address: ":9093"
# Display more output
verbose: true
commands:
  - cmd: /etc/prometheus/am-executor/am-executor_hook.sh #script à
    exécuter
    match_labels:
      "severity": "critical" #se déclencher seulement lorsque l'alerte
    est de niveau "critical"
    notify_on_failure: false
    resolved_signal: SIGUSR1
    ignore_resolved: true
```

Nous pouvons redémarrer le service prometheus-am-executor

```
systemctl restart prometheus-am-executor
```

Avec la commande `ss -pentu`, nous pouvons voir le service écouter sur le port 9118.

```
tcp        LISTEN      0          4096          10.192.43.12:9118          0.0.0.0:*
users: (("am-executor",pid=18518,fd=3)) ino:704995 sk:7 cgroup:/system.slice/prometheus-am-executor.service <->
```

Ajout dans le service Alertmanager

- Rajouter dans les routes

snippet.yaml

```
- receiver: 'executor'
  match_re:
    severity: critical
  continue: true
```

- Ajouter dans les receivers

snippet.yaml

```
- name: 'executor'
  webhook_configs:
    - url: 'http://10.192.43.12:9118' #port d'écoute de prometheus am
    executor
    send_resolved: true
```


- Redémarrer le service Alertmanager

```
systemctl restart alertmanager
```

Rédaction d'un script d'exemple

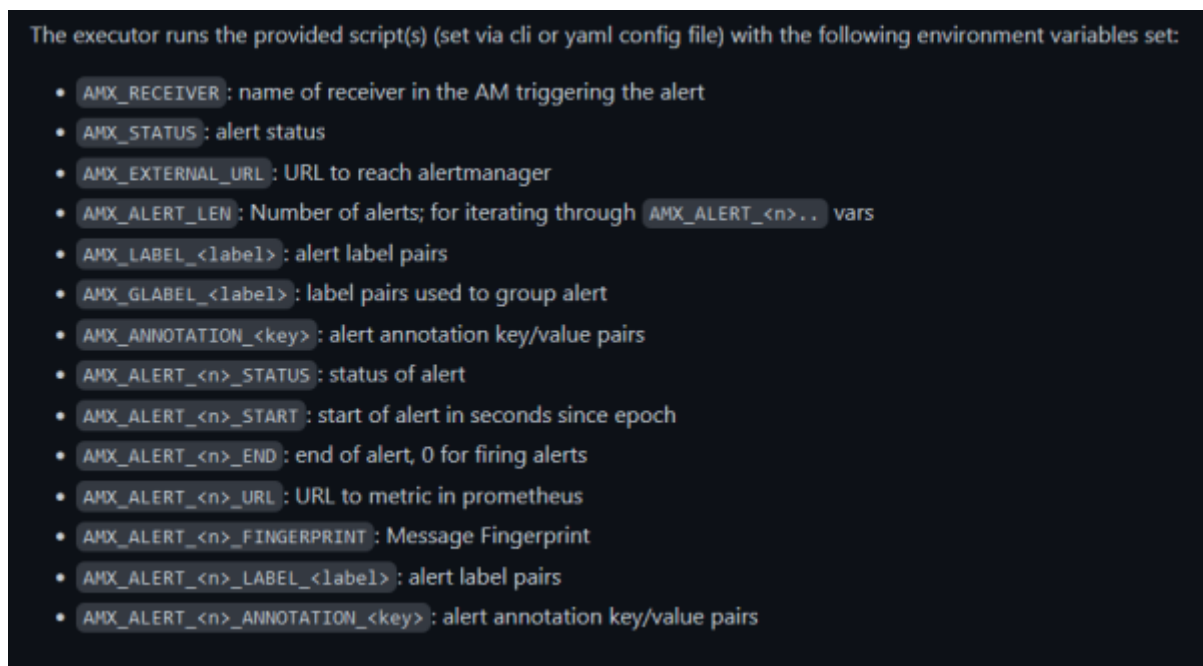
<WRAP center round tip 60%> Rappel : A noter que ce script s'exécute que lorsque la gravité de l'événement est `critical`.

Créer le fichier `am-executor_hook.sh` dans le répertoire `am-executor`

[snippet.bash](#)

```
touch am-executor_hook.sh
chmod a+x am-executor_hook.sh
```

Selon la documentation de Prometheus - am-executor, voici les variables exploitables :



Execution d'une commande sur une machine distante

Dans notre exemple nous souhaitons redémarrer le service `mariaDB` sur serveur `srv-node02` lorsque une alerte est trigger.

Ajout des clés SSH sur les machines

Documentation : <https://www.ssh.com/academy/ssh/copy-id>

Sur le serveur de supervision

- Générer la clé SSH

```
ssh-keygen
```

- Copier la clé sur le serveur distant :

```
ssh-copy-id -i ~/.ssh/id_rsa root@10.192.43.11
```

Sur le serveur BDD

- Générer la clé SSH

```
ssh-keygen
```

- Copier la clé sur le serveur distant :

```
ssh-copy-id -i ~/.ssh/id_rsa root@10.192.43.12
```

- Test de l'accès SSH

```
ssh root@10.192.43.11
```

```
root@node02:~# ssh 'root@10.192.43.12'  
Linux srv-supervision 5.10.0-21-amd64 #1 SMP Debian 5.10.162-1 (2023-01-21) x86_64
```

Modification du script

La commande 'ssh root@10.192.43.11 'systemctl restart mariadb* --all' va redémarrer les services mariaDB.

[snippet.bash](#)

```
#!/bin/bash  
# logger toutes les variables dans un fichier log avec un timecode  
touch executor.log  
echo "$(date)" >> executor.log  
echo $AMX_RECEIVER >> executor.log  
echo $AMX_STATUS >> executor.log  
echo $AMX_EXTERNAL_URL >> executor.log  
echo $AMX_LABEL_alertname >>executor.log  
echo "AMX_LABEL_instance "$AMX_LABEL_instance >> executor.log  
Instance=$(echo $AMX_LABEL_instance | cut -f1 -d":")  
echo ""
```

```
BDD_HOSTNAME="10.192.43.11"

if [[ "$AMX_LABEL_alertname" == "MysqlDown" ]];
then
    #écrire la commande à exécuter ici

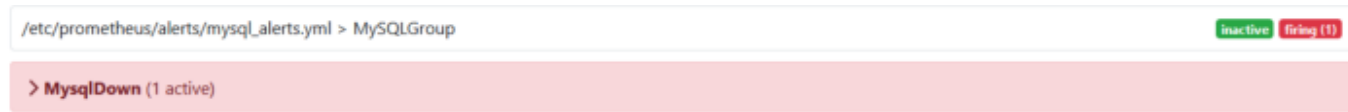
    ssh root@$BDD_HOSTNAME 'systemctl restart mariadb* --all'

else
    echo "Label is different, ${AMX_LABEL_alertname}" >> executor.log
fi
```

Essais de déclenchement

Sur la VM de BDD couper les services mariaDB :

```
systemctl stop mariadb*
```



Vérifier le status de prometheus-am-executor :

```
systemctl status prometheus-am-executor
```

```
19:52:20 Webhook triggered from remote address:port 10.192.43.12:53778
19:52:20 Body: {"receiver":"executor","status":"firing","alerts":[{"status":"firing","labels":{"alertname":"MysqlDown","alias":
19:52:20 Got: &template.Data{Receiver:"executor", Status:"firing", Alerts:template.Alerts(template.Alert{Status:"firing", Labels
19:52:20 Executing: /etc/prometheus/am-executor/am-executor_hook.sh
19:52:20 Command: /etc/prometheus/am-executor/am-executor_hook.sh, result: Ok
```

La commande est bien exécutée et le serveur de BDD est de nouveau en ligne

```
tcp LISTEN 0 80 0.0.0.0:3306 0.0.0.0:* users:((("mariadb",pid=3534114,fd=15)) uid:106 ino:27423991 sk:e
cgroupp:/system.slice/mariadb.service <->
Active Windows
```

Execution d'une requête cURL pour appeler une API

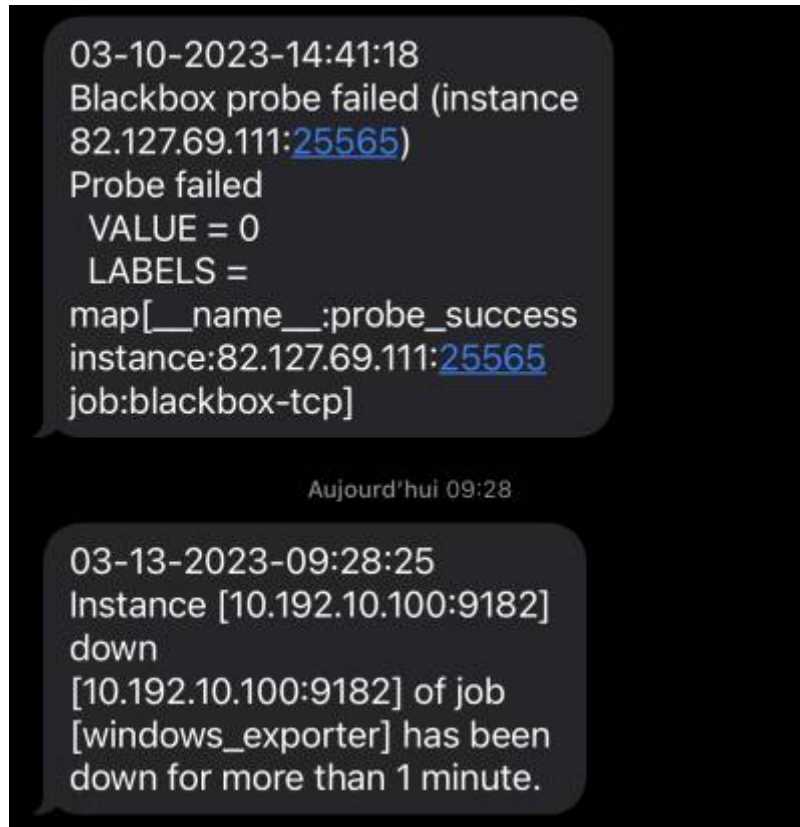
Dans une [autre documentation](#) j'explique comment mettre en place une passerelle SMS pour moins de 20€ afin d'envoyer des alertes SMS.

Suivant la [documentation de RaspiSMS](#) nous pouvons construire cette requête :

[snippet.bash](#)

```
curl -X POST http://10.192.100.204/raspisms/api/scheduled/ -H "X-API-Key: XXXXXXX" -d "text=$NOW%0A$AMX%0A$AMX_ANNOTATION_summary%0A$AMX_ANNOTATION_description" -d contacts["1"]
```

On passe dedans toutes les variables qui nous intéresse afin d'être alerter en cas de soucis grave.

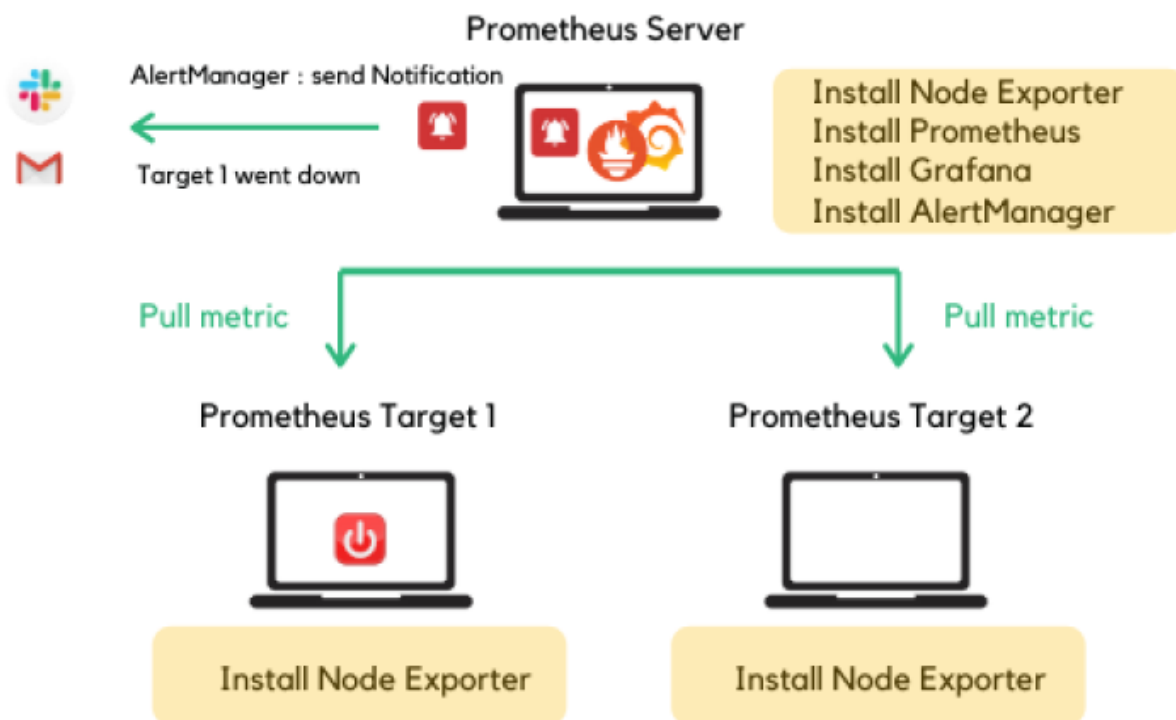


Conclusion

Il existe plein d'autres exporter, la liste est disponible ici :

<https://prometheus.io/docs/instrumenting/exporters/>

Tous les objectifs que nous voulions pour notre solution de monitoring ont été atteints. Grafana et Prometheus sont des outils libres et gratuit, cela les rends beaucoup plus accessible pour les TPE et PME. Il existe une version Enterprise de Grafana qui rajoute des moyens d'authentification, un support et des plugins premium. Ici, il est nullement nécessaire de financer une licence, la version OSS répond parfaitement au besoin.



Infrastructure finale de supervision

Mes sources

1. <https://blog.ippon.fr/2019/03/29/superviser-une-infrastructure-avec-prometheus-part-1-fonctionnement/>
2. <https://blog.zwindler.fr/2019/11/12/tutoriel-installer-prometheus-grafana-sans-docker/>
3. <https://www.how2shout.com/linux/how-to-install-prometheus-in-debian-11-or-ubuntu-20-04/>

Prometheus

1. <https://www.devopsschool.com/blog/how-to-run-prometheus-server-as-a-service/>
2. <https://gist.github.com/eiri/1102e1f3c168684b5a8b0e7a0f5a5a14>
3. https://github.com/prometheus/snmp_exporter
4. <https://techexpert.tips/fr/prometheus-fr/prometheus-surveille-mysql-sur-ubuntu-linux/>

Grafana

1. <https://grafana.com/blog/2022/02/01/an-advanced-guide-to-network-monitoring-with-grafana-and-prometheus/>
2. <https://geekflare.com/prometheus-grafana-setup-for-linux/>
3. <https://grafana.com/docs/grafana/latest/setup-grafana/installation/debian/>
4. <https://www.hostwinds.fr/tutorials/how-to-install-grafana-debian-ubuntu>
5. <http://thibaut.ovh/adminsys/linux/grafana/tuto-installer-grafana-sous-debian-ubuntu>
6. <https://community.grafana.com/t/how-uninstall-all-of-grafana/41732/14>
7. <https://computingforgeeks.com/how-to-install-grafana-on-debian-linux/>

8. <https://techexpert.tips/fr/grafana-fr/grafana-surveillance-des-peripheriques-snmp/>
9. <https://grafana.com/docs/grafana/latest/setup-grafana/installation/debian/>

Tableaux utilisés :

- <https://grafana.com/grafana/dashboards/14451-windows-exporter-for-prometheus-dashboard-en/>
- <https://grafana.com/grafana/dashboards/11074-node-exporter-for-prometheus-dashboard-en-v2-0201010/>
- <https://grafana.com/grafana/dashboards/7587>

node_exporter

1. <https://computingforgeeks.com/how-to-install-prometheus-and-node-exporter-on-debian/>
2. <https://devopscube.com/monitor-linux-servers-prometheus-node-exporter/>
3. <https://gist.github.com/jarek-przygodzki/735e15337a3502fea40beba27e193b04>

windows_exporter

1. https://github.com/prometheus-community/windows_exporter
2. <https://www.devopsschool.com/blog/how-to-install-windows-exporter-for-prometheus/>

pve_exporter

1. <https://blog.zwindler.fr/2020/01/06/proxmox-ve-prometheus/>
2. <https://blog.ataxya.net/supervision-de-proxmox-et-de-mikrotik-via-prometheus-grafana/>

Prometheus Blackbox

1. <https://blog.ruanbekker.com/blog/2019/05/17/install-blackbox-exporter-to-monitor-websites-with-prometheus/>
2. <https://geekflare.com/fr/monitor-website-with-blackbox-prometheus-grafana/>
3. <https://medium.com/techno101/how-to-send-a-mail-using-prometheus-alertmanager-7e880a3676db>
4. https://github.com/prometheus/blackbox_exporter/issues/14

Prometheus SQL

1. <https://computingforgeeks.com/monitoring-mysql-mariadb-with-prometheus-in-five-minutes/>

Prometheus SNMP

1. <https://sbcode.net/prometheus/snmp-exporter/>
2. <https://medium.com/@openmohan/snmp-monitoring-and-easing-it-with-prometheus-b157c0a42>

c0c

3. https://github.com/prometheus/snmp_exporter
4. <https://performance-monitoring-with-prometheus.readthedocs.io/en/latest/switch.html>
5. <https://awesome-prometheus-alerts.grep.to/rules.html>
6. https://github.com/cliv/rfc1628_ups_prometheus_module
7. https://github.com/billykwooten/idrac_prometheus_snmp_module
8. https://github.com/prometheus/snmp_exporter/tree/main/generator
9. <https://sbcode.net/prometheus/snmp-exporter-generator>
10. <https://stackoverflow.com/questions/56009729/prometheus-help-editing-configuring-snmp-exporters-generator-yml-file-for-cisc/>
11. <https://grumpysysadmin.medium.com/configuring-prometheus-am-executor-for-automation-87d8f5514056>

Pour aller plus loin...

- agent prometheus pour mesurer la débit du lien Internet
 1. <https://github.com/MiguelNdeCarvalho/speedtest-exporter>
- extinction automatique via surveillance d'onduleur
 1. <https://asokolsky.github.io/proxmox/nut.html>
 2. https://www.linkedin.com/pulse/installer-et-surveiller-votre-onduleur-sous-debian-olivier-henry/?trk=pulse-article_more-articles_related-content-card

From:
<https://wiki.stoneset.fr/> - **StoneSet - Documentations**

Permanent link:
https://wiki.stoneset.fr/doku.php?id=wiki:linux:grafana_prometheus&rev=1682517975

Last update: **2023/04/26 16:06**

