Monitoring: Prometheus et Grafana

UP ASI
Bureau E204

Plan du cours

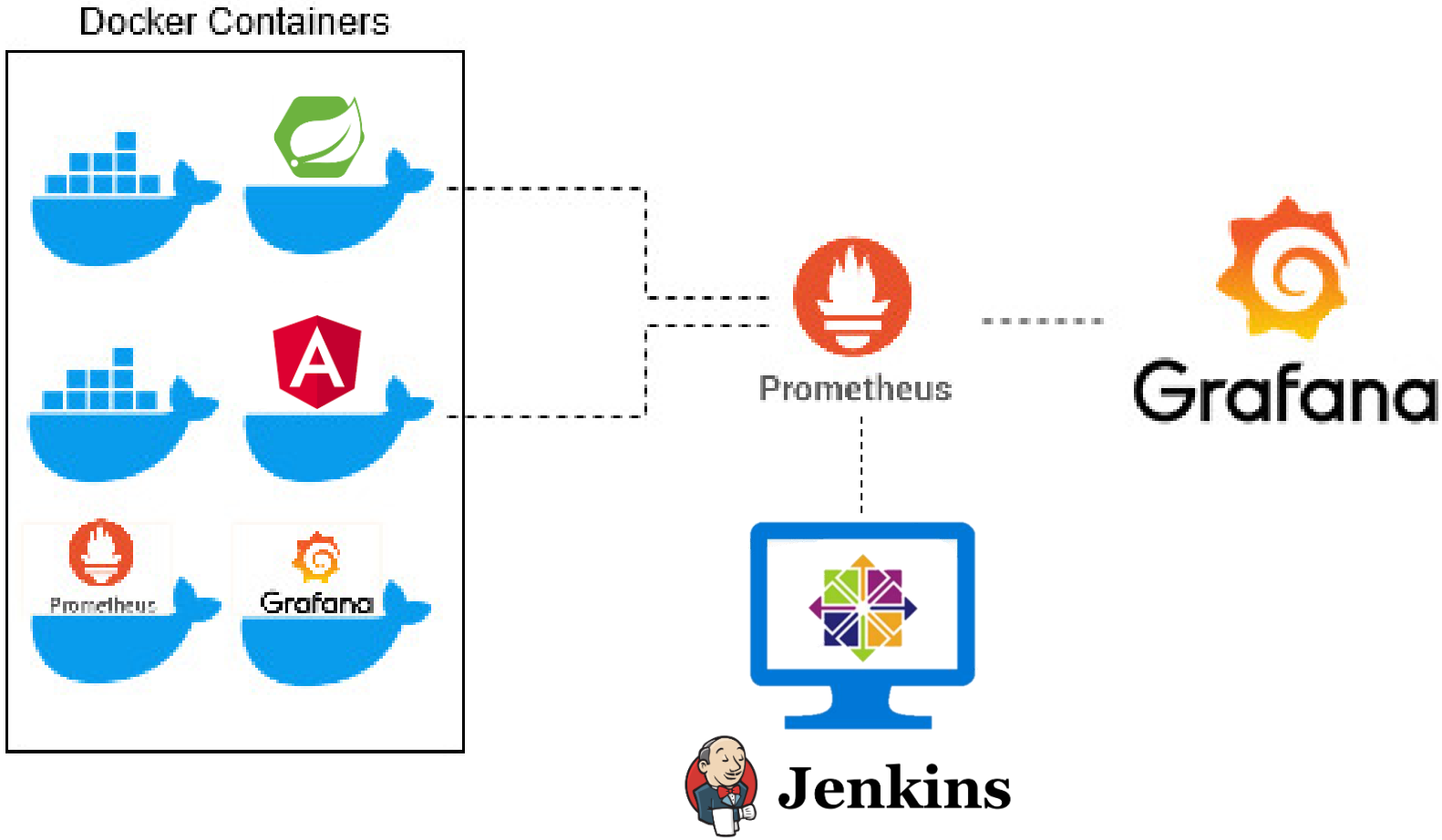
- Introduction: Superviser une application
- Prometheus
  - Définition
  - Installation
  - Configuration
- Grafana
  - Définition
  - Architecture
  - Installation
  - Configuration
- Travail à faire

Introduction

- La supervision des applications (Monitoring) est le processus de **collecte des données** de log afin d'aider l'équipe à suivre la disponibilité, les bugs, l'utilisation des ressources et les changements de performance des applications qui affectent l'expérience de l'utilisateur final (UX).
- Les outils de surveillance des applications fournissent des alertes en cas d'anomalies en direct et, grâce à un traçage distribué, permettent de suivre les événements qui forment une chaîne de causalité (impact d'un évènement sur divers outils).

Introduction

→ Pour notre cas, nous allons utiliser Prometheus et Grafana pour superviser l'environnement sur lequel nous travaillons.



Prometheus - Définition

- Prometheus est un système de monitoring offrant une base de données de séries chronologiques.
- Il est basé sur le langage Go (langage de programmation compilé et concurrent inspiré de C et Pascal développé par Google).
- Il est conçu pour surveiller des cibles (Serveurs, Bases de données, Machines virtuelles) → A peu près tout peut être surveillé avec Prometheus.

Prometheus - Définition

- Il travaille en **double delta**: Il calcule l'écart de la valeur par rapport à sa valeur précédente pour le suivi de performances.
- Ce n'est pas un outil de journalisation.
- Ce n'est pas un outil de traçage.
- C'est l'outil de surveillance le plus répandu dans les environnements natifs du cloud
- Très facile à mettre en place
- Très performant
- Scrapping continu

Prometheus - Définition

- Cet outil est composé de :
  - Une base de données (TimeSeries: BD pour les données collectées après des intervalles de temps successifs ou fixes)
  - Un serveur web
  - Un moteur de base de données

Prometheus - Définition

- **Fonctionnement:** Il cherche sur une route donnée (@IP, URL, …) les informations nécessaires **(Metrics)** et les stocke dans sa base de données interne (TimeSeries) avec un format standardisé (Clé, timestamp, valeur).
- Il fournit un langage de requêtage appelé **PromQL (Prometheus Query Language)** qui permet à l'utilisateur de sélectionner et d'agréger des données de séries chronologiques en temps réel. Le résultat d'une requête peut être affiché sous forme de graphique, de données tabulaires dans l'interface de Prometheus ou consommé par des systèmes externes via l'API HTTP (Grafana, …).

Prometheus - Installation

- Vous allez utiliser une image Docker.

docker pull prom/Prometheus

```
[root@localhost vagrant]# docker pull prom/prometheus
```

- Création d'un conteneur Docker Prometheus.

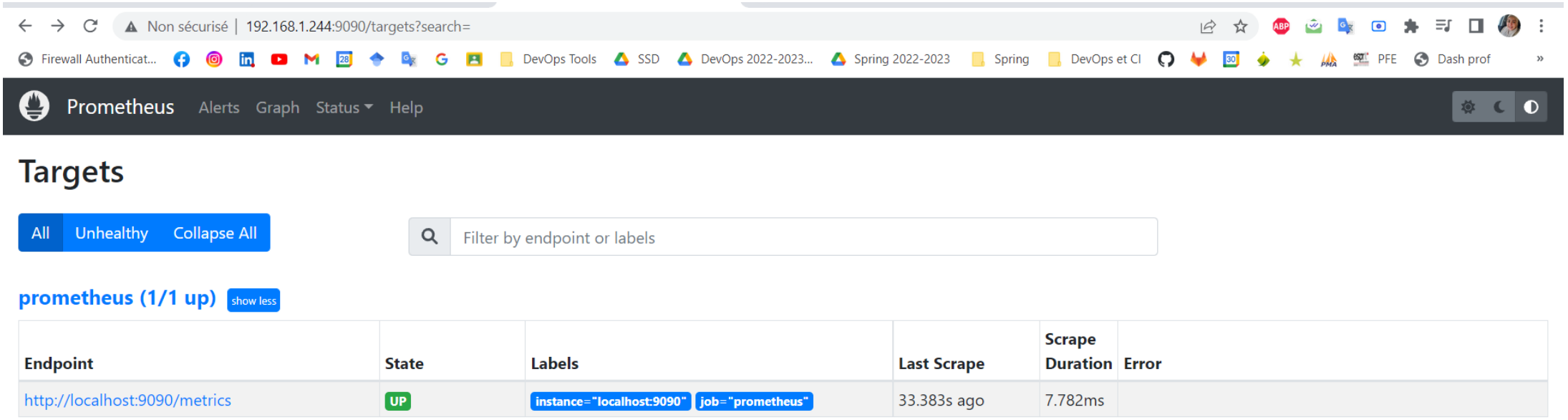
docker run -d --name prometheus -p 9090:9090 prom/prometheus

```
[root@localhost vagrant]# docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS          PORTS                    NAMES
9c45242eec8e   prom/prometheus      "/bin/prometheus -..."   11 days ago    Up 4 seconds    0.0.0.0:9090->9090/tcp   prometheu
s
```

Prometheus - Configuration

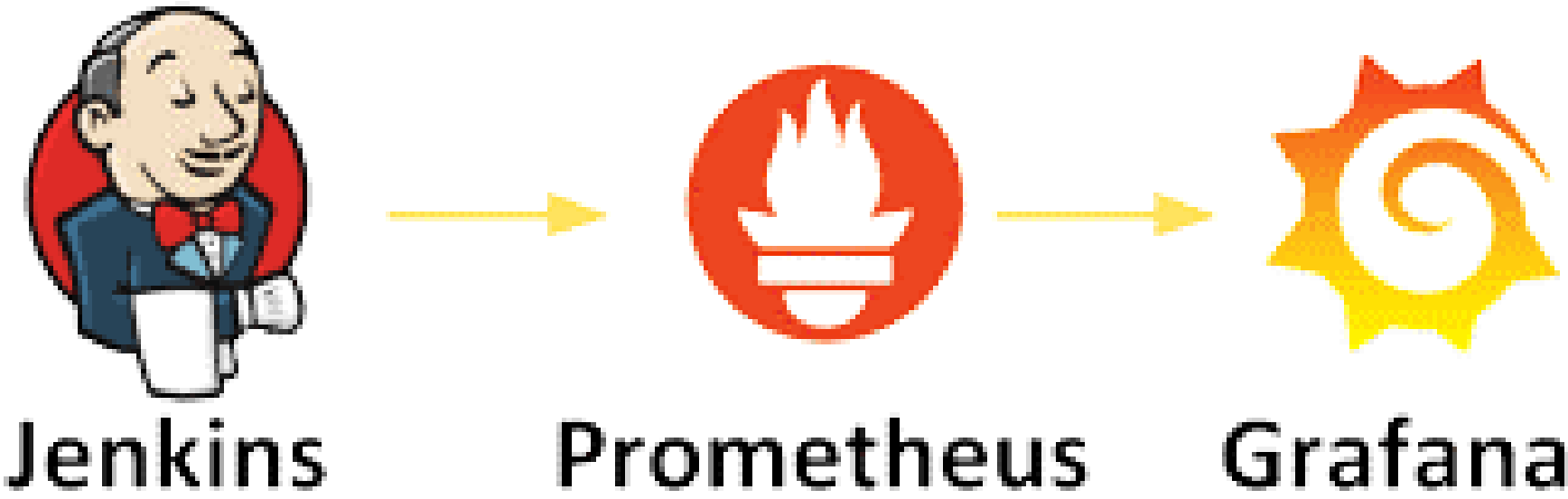
- Accéder à l'interface web de prometheus :

http://@IP:9090



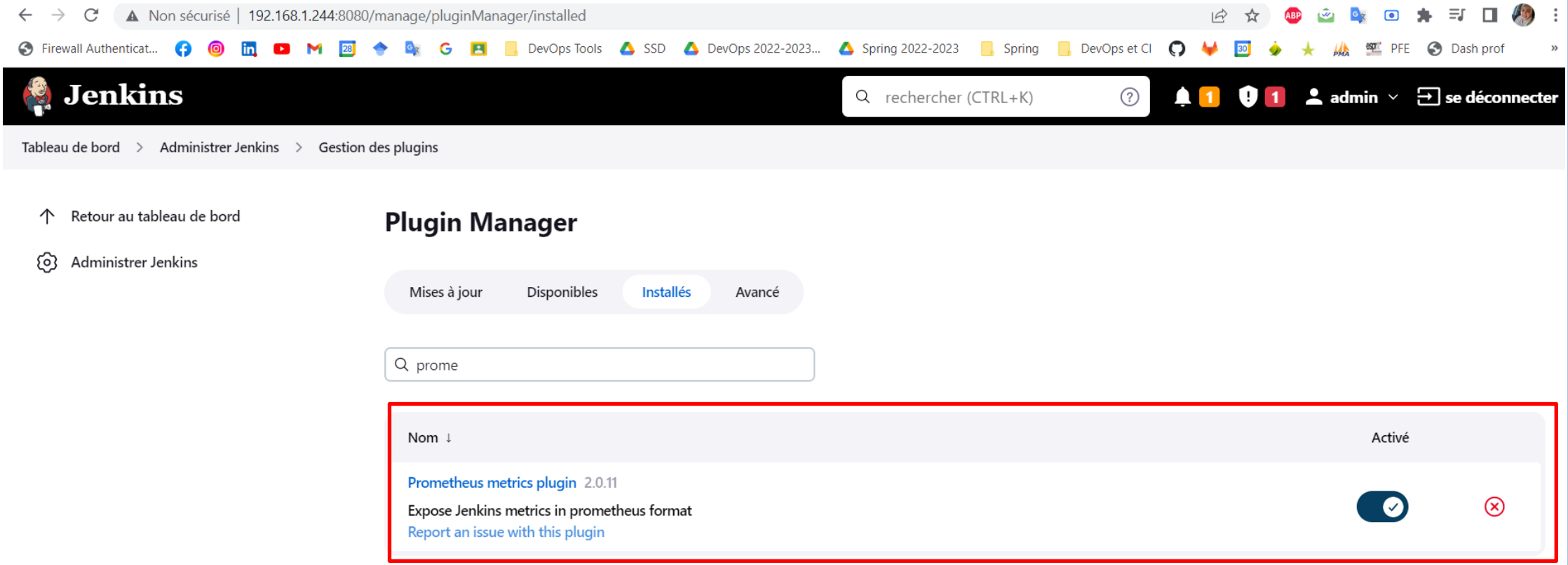
Prometheus - Configuration

- Pour utiliser « Prometheus et Grafana », nous allons configurer la supervision du serveur « Jenkins ».



Prometheus - Configuration

- Installer le plugin « Prometheus metrics » dans Jenkins:



→ Le plugin « Prometheus metrics » ne supporte pas le chargement dynamique. Jenkins doit être redémarré pour que la mise à jour soit effective (systemctl restart jenkins).

Prometheus - Configuration

- Après l'installation du plugin, vous pouvez accéder aux métriques exposés par le serveur jenkins: http://@IP:8080/prometheus

```
# HELP default_jenkins_builds_duration_milliseconds_summary Summary of Jenkins build times in milliseconds by Job
# TYPE default_jenkins_builds_duration_milliseconds_summary summary
default_jenkins_builds_duration_milliseconds_summary_count{jenkins_job="projet",repo="NA",buildable="true",} 31.0
default_jenkins_builds_duration_milliseconds_summary_sum{jenkins_job="projet",repo="NA",buildable="true",} 4.00863599E8
# HELP default_jenkins_builds_success_build_count_total Successful build count
# TYPE default_jenkins_builds_success_build_count_total counter
default_jenkins_builds_success_build_count_total{jenkins_job="projet",repo="NA",buildable="true",} 4.0
# HELP default_jenkins_builds_failed_build_count_total Failed build count
# TYPE default_jenkins_builds_failed_build_count_total counter
default_jenkins_builds_failed_build_count_total{jenkins_job="projet",repo="NA",buildable="true",} 27.0
# HELP default_jenkins_builds_health_score Health score of a job
# TYPE default_jenkins_builds_health_score gauge
default_jenkins_builds_health_score{jenkins_job="projet",repo="NA",buildable="true",} 60.0
# HELP default_jenkins_builds_available_builds_count Number of builds available for this job
# TYPE default_jenkins_builds_available_builds_count gauge
default_jenkins_builds_available_builds_count{jenkins_job="projet",repo="NA",buildable="true",} 31.0
# HELP default_jenkins_builds_discard_active Indicates if the build discarder is active for the given job
# TYPE default_jenkins_builds_discard_active gauge
default_jenkins_builds_discard_active{jenkins_job="projet",repo="NA",buildable="true",} 0.0
# HELP default_jenkins_builds_last_build_result_ordinal Build status of a job.
# TYPE default_jenkins_builds_last_build_result_ordinal gauge
default_jenkins_builds_last_build_result_ordinal{jenkins_job="projet",repo="NA",buildable="true",} 4.0
# HELP default_jenkins_builds_last_build_result Build status of a job as a boolean (0 or 1)
# TYPE default_jenkins_builds_last_build_result gauge
default_jenkins_builds_last_build_result{jenkins_job="projet",repo="NA",buildable="true",} 0.0
# HELP default_jenkins_builds_last_build_duration_milliseconds Build times in milliseconds of last build
# TYPE default_jenkins_builds_last_build_duration_milliseconds gauge
default_jenkins_builds_last_build_duration_milliseconds{jenkins_job="projet",repo="NA",buildable="true",} 1437251.0
# HELP default_jenkins_builds_last_build_start_time_milliseconds Last build start timestamp in milliseconds
# TYPE default_jenkins_builds_last_build_start_time_milliseconds gauge
default_jenkins_builds_last_build_start_time_milliseconds{jenkins_job="projet",repo="NA",buildable="true",} 1.698537143919E12
# HELP default_jenkins_builds_last_stage_duration_milliseconds_summary Summary of Jenkins build times by Job and Stage in the last build
# TYPE default_jenkins_builds_last_stage_duration_milliseconds_summary summary
default_jenkins_builds_last_stage_duration_milliseconds_summary_count{jenkins_job="projet",repo="NA",buildable="true",stage="Declarative: Checkout SCM",} 1.0
default_jenkins_builds_last_stage_duration_milliseconds_summary_sum{jenkins_job="projet",repo="NA",buildable="true",stage="Declarative: Checkout SCM",} 4201.0
# HELP default_jenkins_version_info Jenkins Application Version
# TYPE default_jenkins_version_info gauge
default_jenkins_version_info{version="2.414.2",} 1.0
# HELP default_jenkins_up Is Jenkins ready to receive requests
```

Prometheus - Configuration

- Pour que « Prometheus » puisse récupérer les métriques exposés par « Jenkins », vous devez ajouter un « Job » dans le fichier « prometheus.yml »

```
- job_name: jenkins
    metrics_path: /prometheus
    static_configs:
      - targets: ['@IP:8080']
```

Prometheus - Configuration

1- Créer le fichier « prometheus.yml »:

```
root@vagrant:~# mkdir prometheus
root@vagrant:~# cd prometheus/
root@vagrant:~/prometheus# vim prometheus.yml
```

2- Ajouter cette configuration:

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
rule_files:
scrape_configs:
    - job_name: "prometheus
      static_configs:
              - targets: ["localhost:9090"]
    - job_name: "jenkins"
      metrics_path: "/prometheus"
      static_configs:
              - targets: [“@IP:8080"]
```

Prometheus - Configuration

- Copier le fichier dans le container « Prometheus »:

  docker cp prometheus.yml prometheus:/etc/prometheus/prometheus.yml

```
root@vagrant:~/prometheus# docker cp prometheus.yml prometheus:/etc/prometheus/prometheus.yml
Successfully copied 3.07kB to prometheus:/etc/prometheus/prometheus.yml
```

- Vérifier la configuration:

  docker exec prometheus cat /etc/prometheus/prometheus.yml

```
root@vagrant:~/prometheus# docker exec prometheus  cat /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"
    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.
    static_configs:
      - targets: ["localhost:9090"]
```

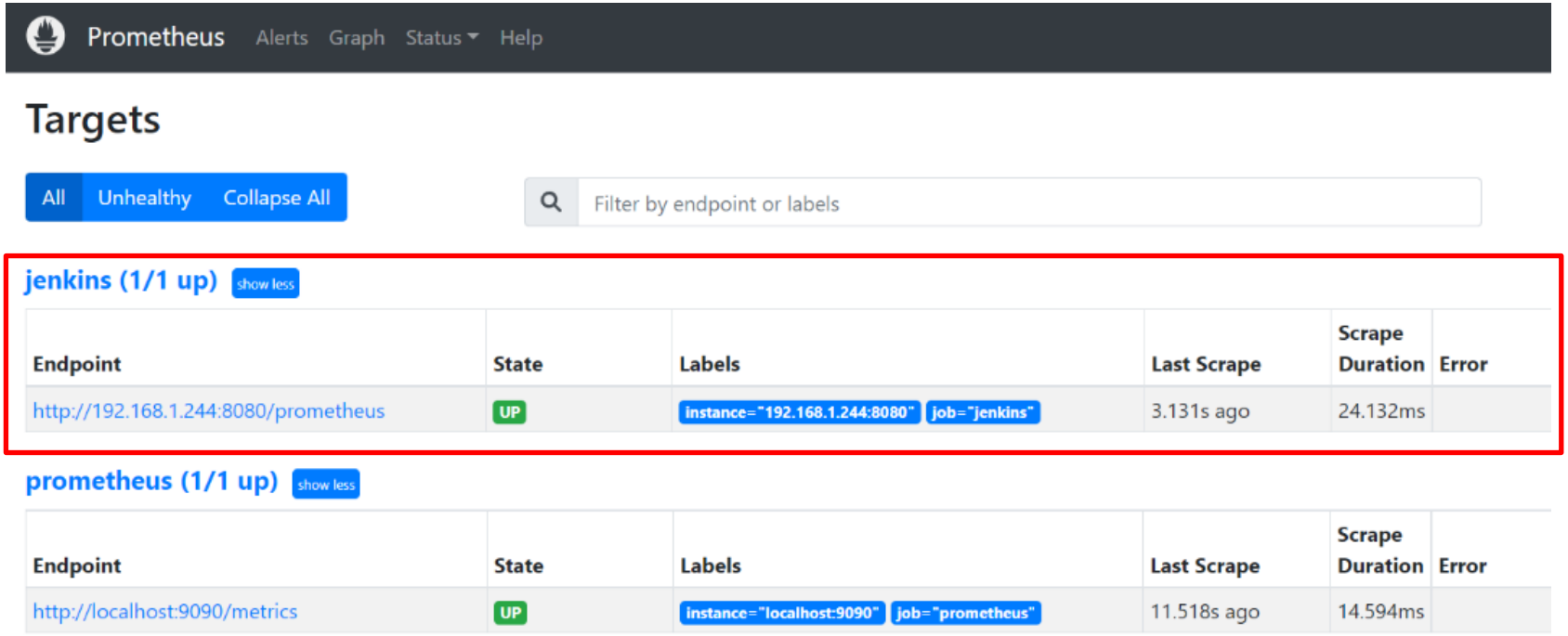
Prometheus - Configuration

- Après la configuration, nous devons redémarrer le conteneur:

docker restart prometheus

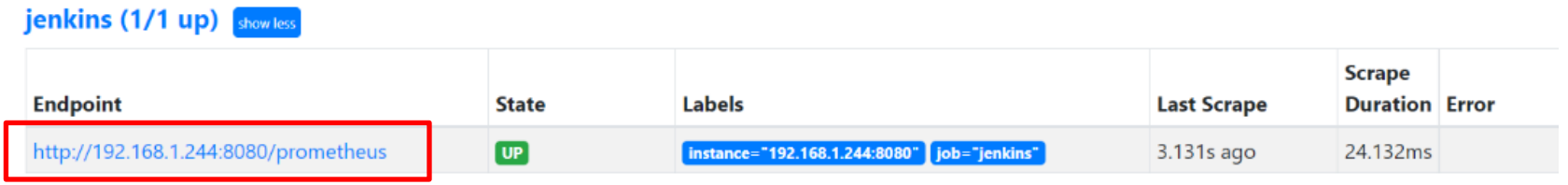
```
[root@localhost vagrant]# docker restart prometheus
prometheus
```

- Après le redémarrage:



Prometheus - Configuration

- Pour visualiser les informations récupérées:



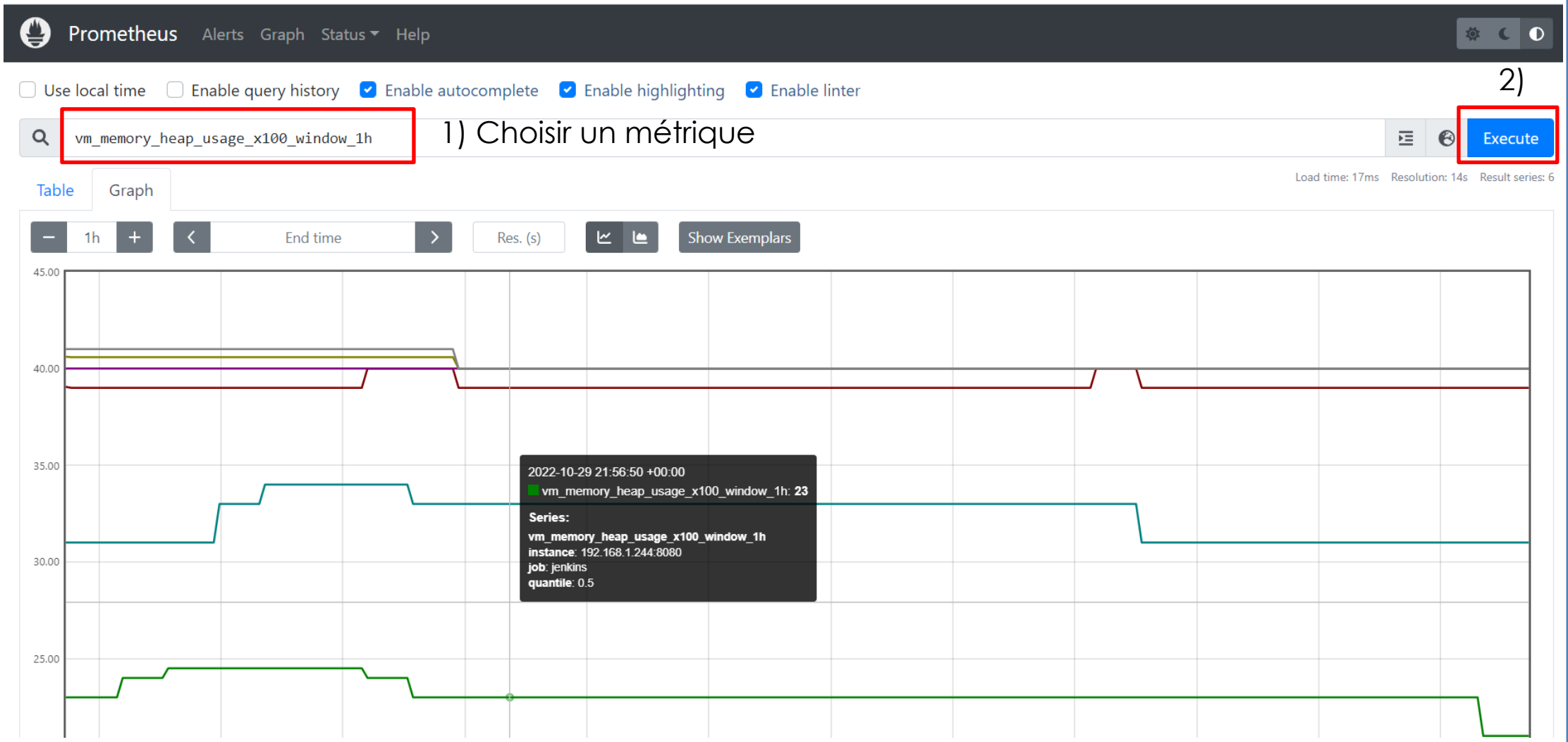
Non sécurisé | 192.168.1.244:8080/prometheus/

Firewall Authenticat... DevOps Tools SSD DevOps 2022-2023... Spring 2022-2023 Spring DevOps et CI

```
# HELP vm_memory_total_used_window_1h Generated from Dropwizard metric import (metric=vm.memory.total.used.window.1h, type=jenkins.metrics.util.AutoSamplingHistogram)
# TYPE vm_memory_total_used_window_1h summary
vm_memory_total_used_window_1h{quantile="0.5",} 7.24088384E8
vm_memory_total_used_window_1h{quantile="0.75",} 7.40475432E8
vm_memory_total_used_window_1h{quantile="0.95",} 7.569601536E8
vm_memory_total_used_window_1h{quantile="0.98",} 7.5711888E8
vm_memory_total_used_window_1h{quantile="0.99",} 7.5711888E8
vm_memory_total_used_window_1h{quantile="0.999",} 7.5711888E8
vm_memory_total_used_window_1h_count 22.0
# HELP jenkins_queue_blocked_history Generated from Dropwizard metric import (metric=jenkins.queue.blocked.history, type=jenkins.metrics.util.AutoSamplingHistogram)
# TYPE jenkins_queue_blocked_history summary
jenkins_queue_blocked_history{quantile="0.5",} 0.0
jenkins_queue_blocked_history{quantile="0.75",} 0.0
jenkins_queue_blocked_history{quantile="0.95",} 0.0
jenkins_queue_blocked_history{quantile="0.98",} 0.0
jenkins_queue_blocked_history{quantile="0.99",} 0.0
jenkins_queue_blocked_history{quantile="0.999",} 0.0
jenkins_queue_blocked_history_count 22.0
# HELP vm_memory_pools_Metaspace_used_window_1h Generated from Dropwizard metric import (metric=vm.memory.pools.Metaspace.used.window.1h, type=jenkins.metrics.util.AutoSampl
```

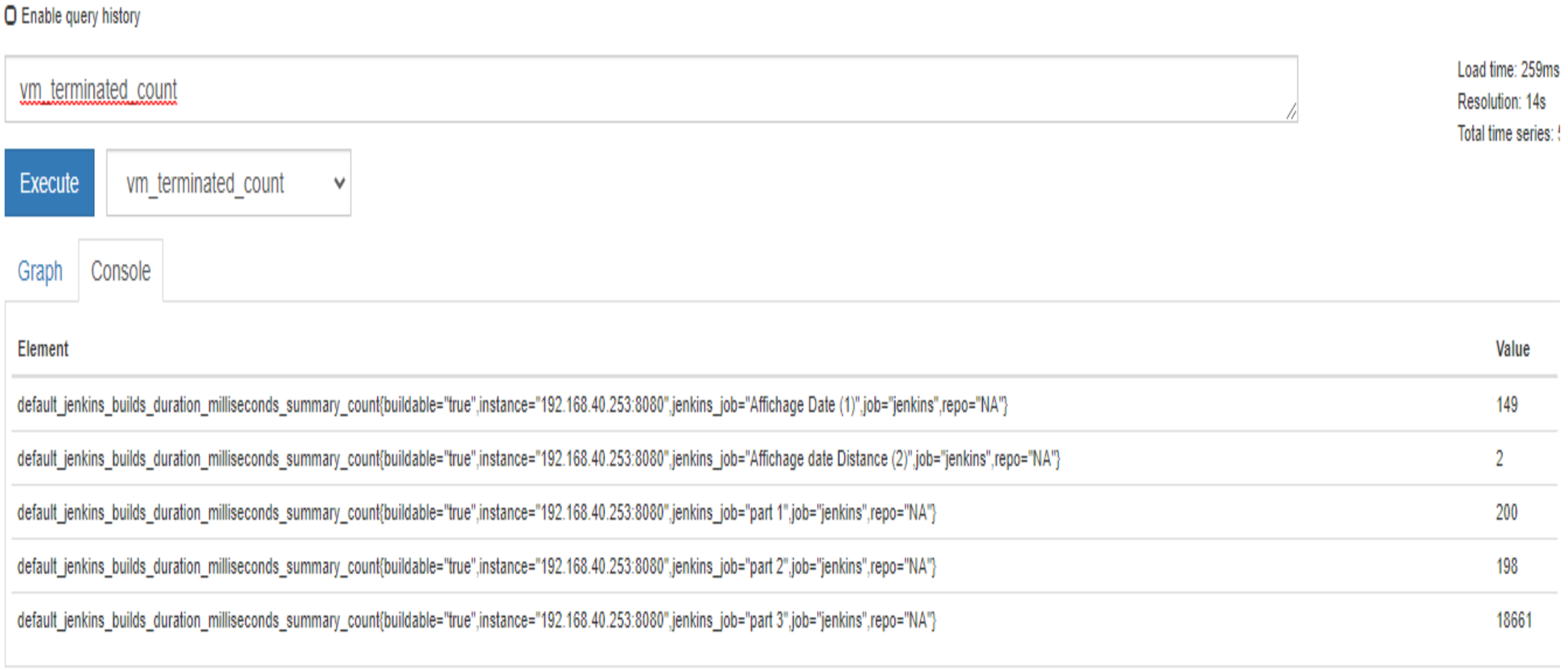
Prometheus - Configuration

- Pour visualiser les valeurs récupérées d'une métrique:



Prometheus - Configuration

- Pour visualiser les valeurs récupérées d'une métrique:



Prometheus - Configuration

- Pour visualiser le contenu du fichier de configuration:

Status → Configuration

La configuration globale

```
global:
  scrape_interval: 15s
  scrape_timeout: 10s
  evaluation_interval: 15s
alerting:
  alertmanagers:
  - follow_redirects: true
    enable_http2: true
    scheme: http
    timeout: 10s
    api_version: v2
    static_configs:
    - targets: []
```

Intervalle de récupération des données

Timeout lors du scrapping

La configuration du scrapping pour un serveur (Exemple: Jenkins)

```
- job_name: jenkins
  honor_timestamps: true
  scrape_interval: 15s
  scrape_timeout: 10s
  metrics_path: /prometheus
  scheme: http
  follow_redirects: true
  enable_http2: true
  static_configs:
  - targets:
    - 192.168.1.244:8080
```

Nom du job

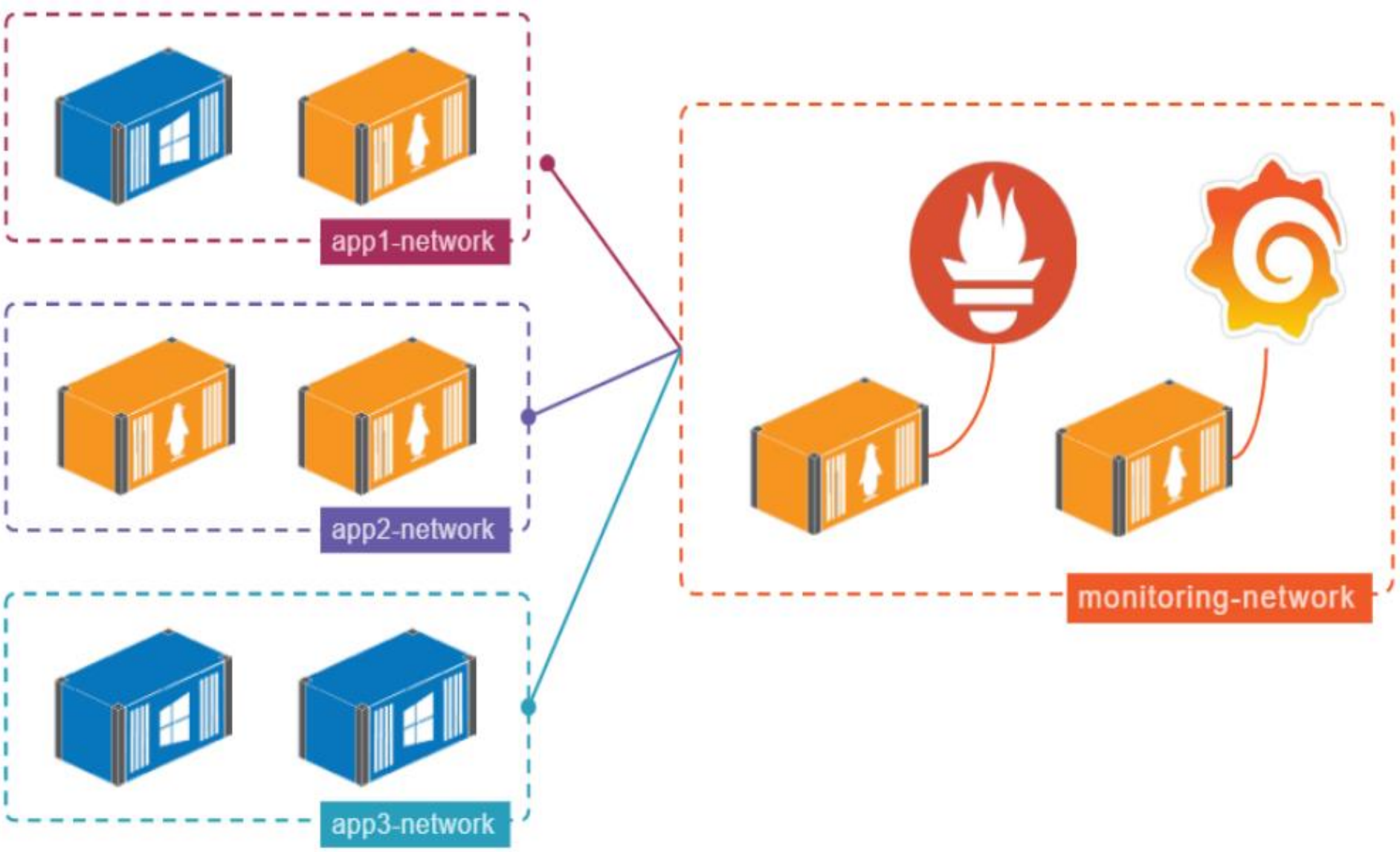
Route de scrapping

L'@ IP du serveur

Grafana - Définition

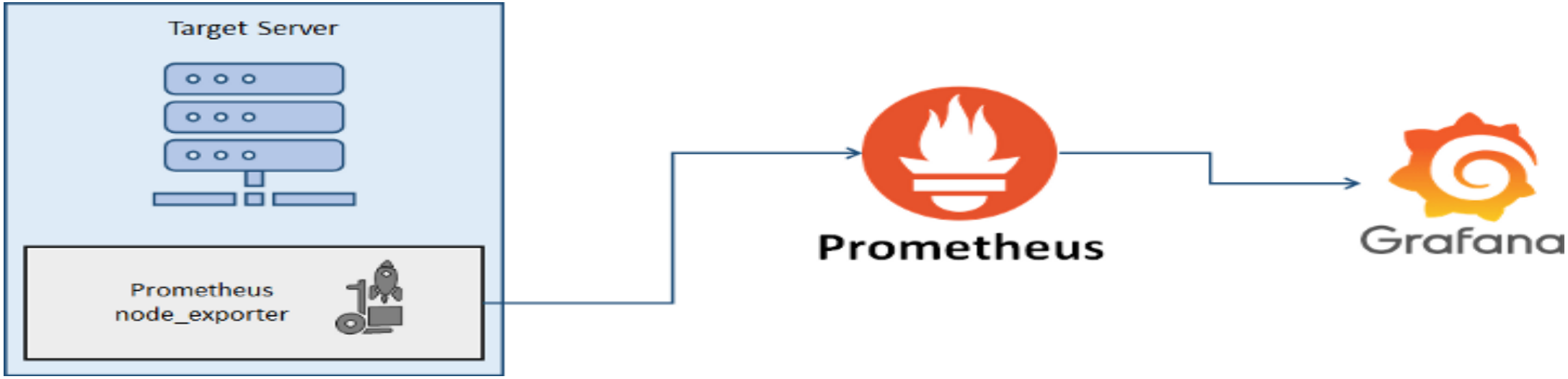
- Grafana est un logiciel **open source** de visualisation et d'analyse.
- Il permet de faire la visualisation à travers:
  - Les graphiques
  - Les tableaux
  - Les gauges
  - Les histogrammes
  - Les points
  - …
- Il permet d'interroger, de visualiser, d'alerter et d'explorer des métriques, quel que soit l'endroit où elles sont stockées (Prometheus, influxdb, postgres, mysql, elastic search, …)
- Il stocke ses données dans une base de données interne (SQL Lite)
- Cet outil est utilisé par des millions d'utilisateurs (Plus de 750k installation et 42k Github stars)

Grafana - Architecture



Grafana - Architecture

- Node exporter est un exportateur Prometheus utilisé pour exposer les métriques des serveurs Linux.
- Grace au Node exporter, nous pouvons exposer diverses ressources du système telles que la RAM, l'utilisation du processeur, l'utilisation de la mémoire, l'espace disque, etc.
- Node exporter fonctionne comme un système qui rassemble les métriques de votre système et avec l'aide de Grafana, nous pouvons visualiser le métrique.



Grafana - Installation

- Vous allez utiliser une image Docker.

docker pull grafana/grafana

```
[root@localhost vagrant]# docker pull grafana/grafana
```

- Création d'un conteneur Docker Grafana.

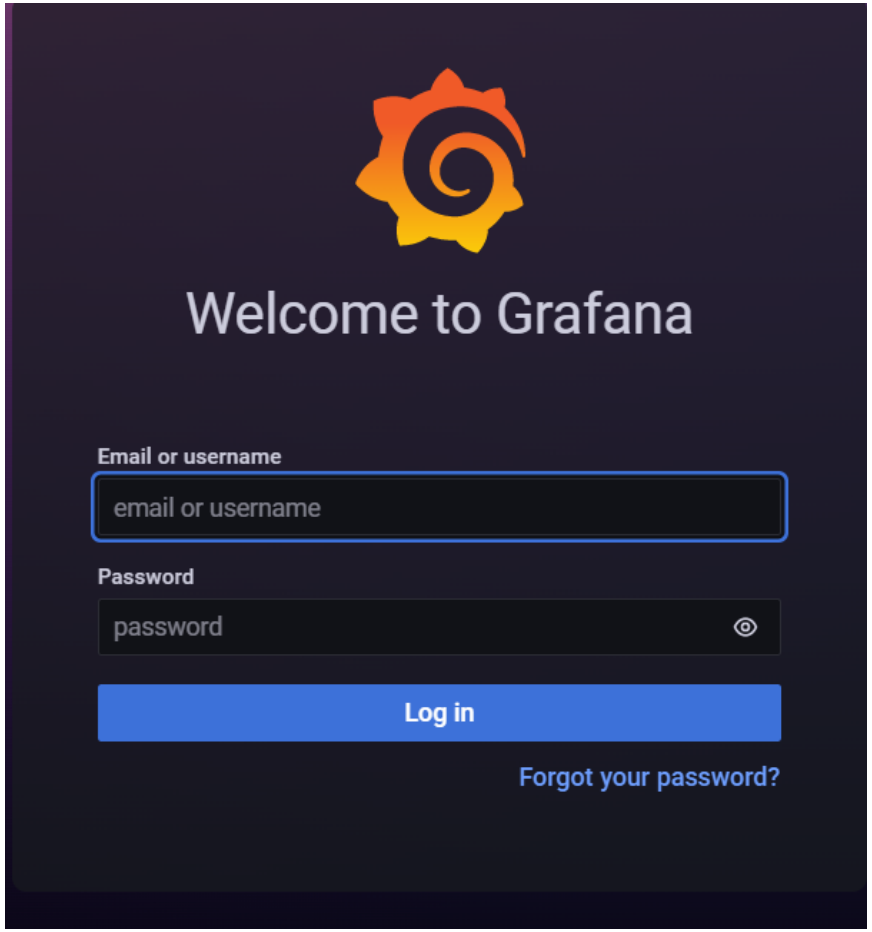
docker run -d --name grafana -p 3000:3000 Grafana/grafana

```
[root@localhost vagrant]# docker ps -a
CONTAINER ID        IMAGE                 COMMAND                  CREATED             STATUS              PORTS                    NAMES
e72dcb971cec        grafana/grafana       "/run.sh"                15 seconds ago      Up 10 seconds       0.0.0.0:3000->3000/tcp   grafana
```

Grafana - Configuration

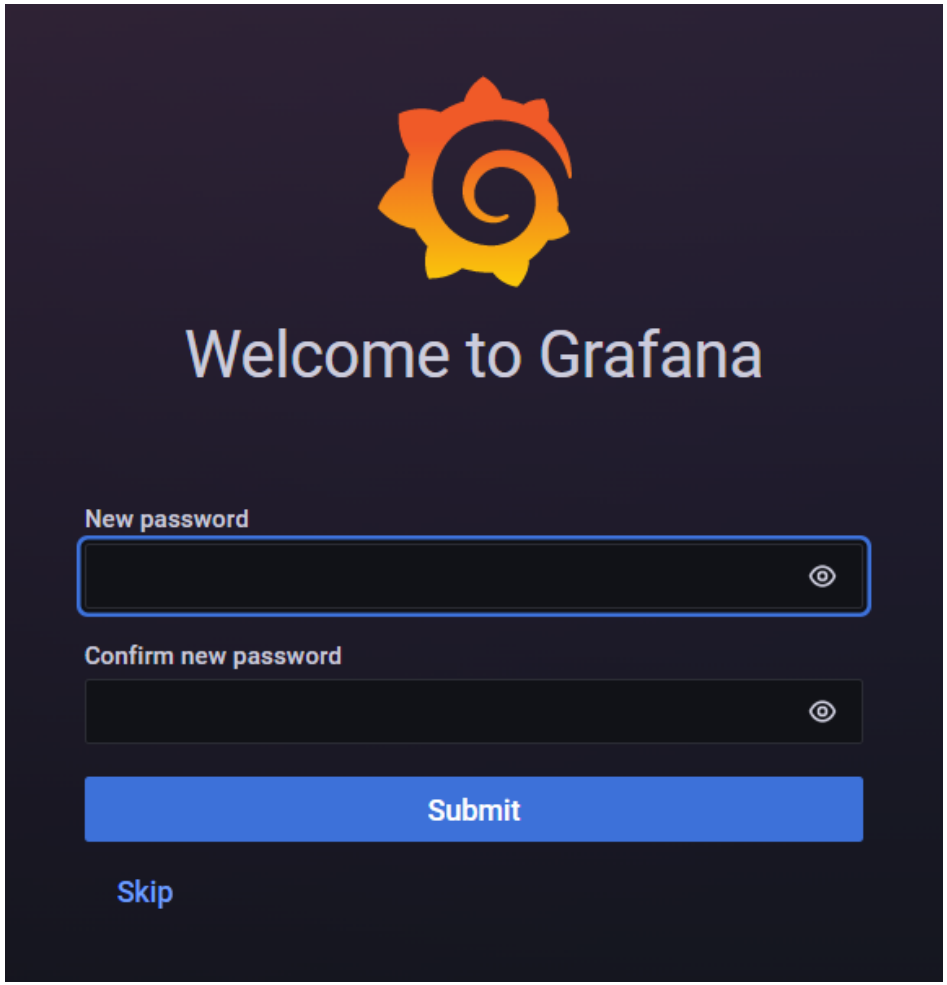
- Accéder à l'interface web de grafana : http://@IP:3000

Les paramètres d'accès par défaut: admin/admin

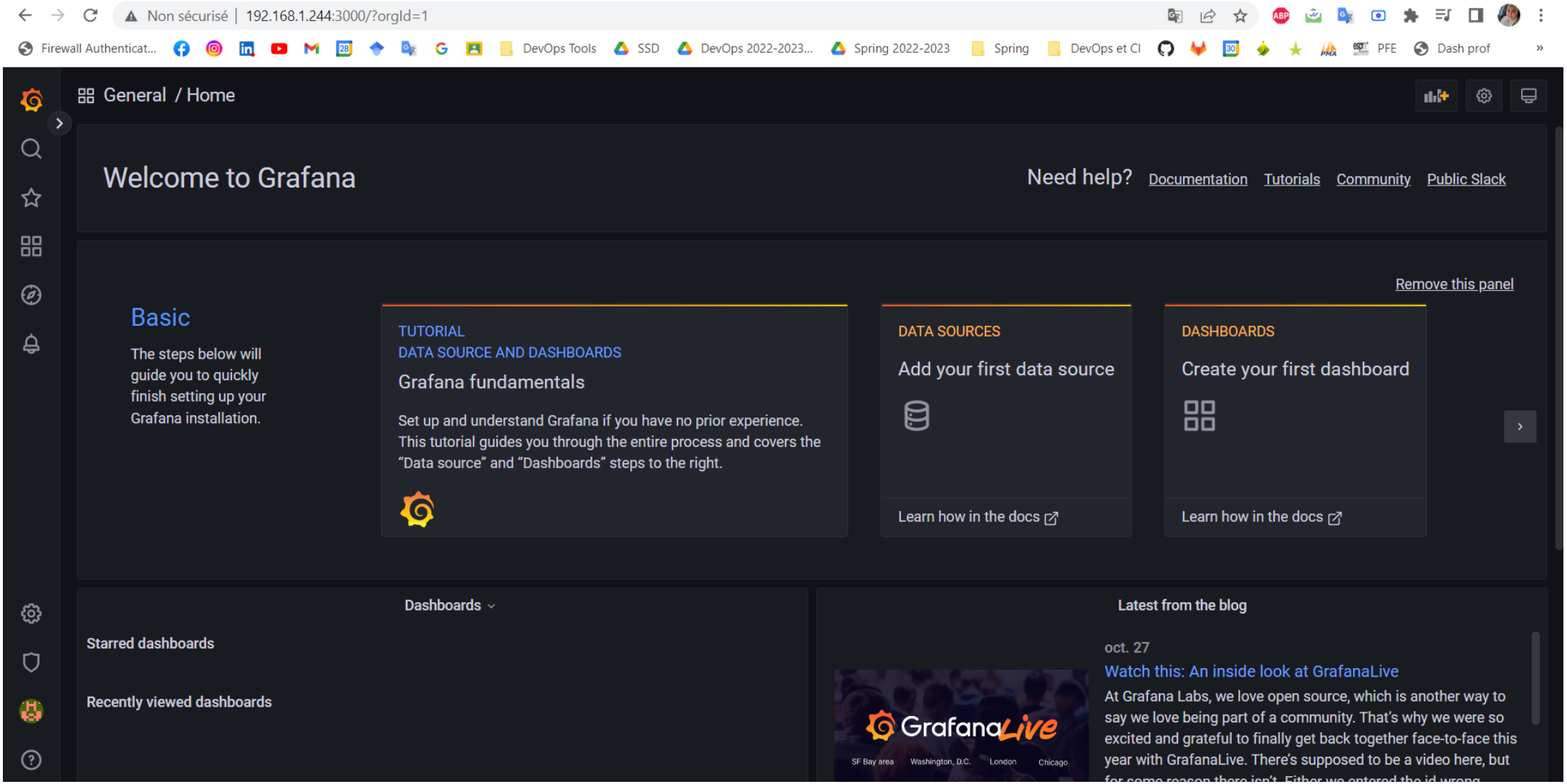


Grafana - Configuration

- Changer le mot de passe:

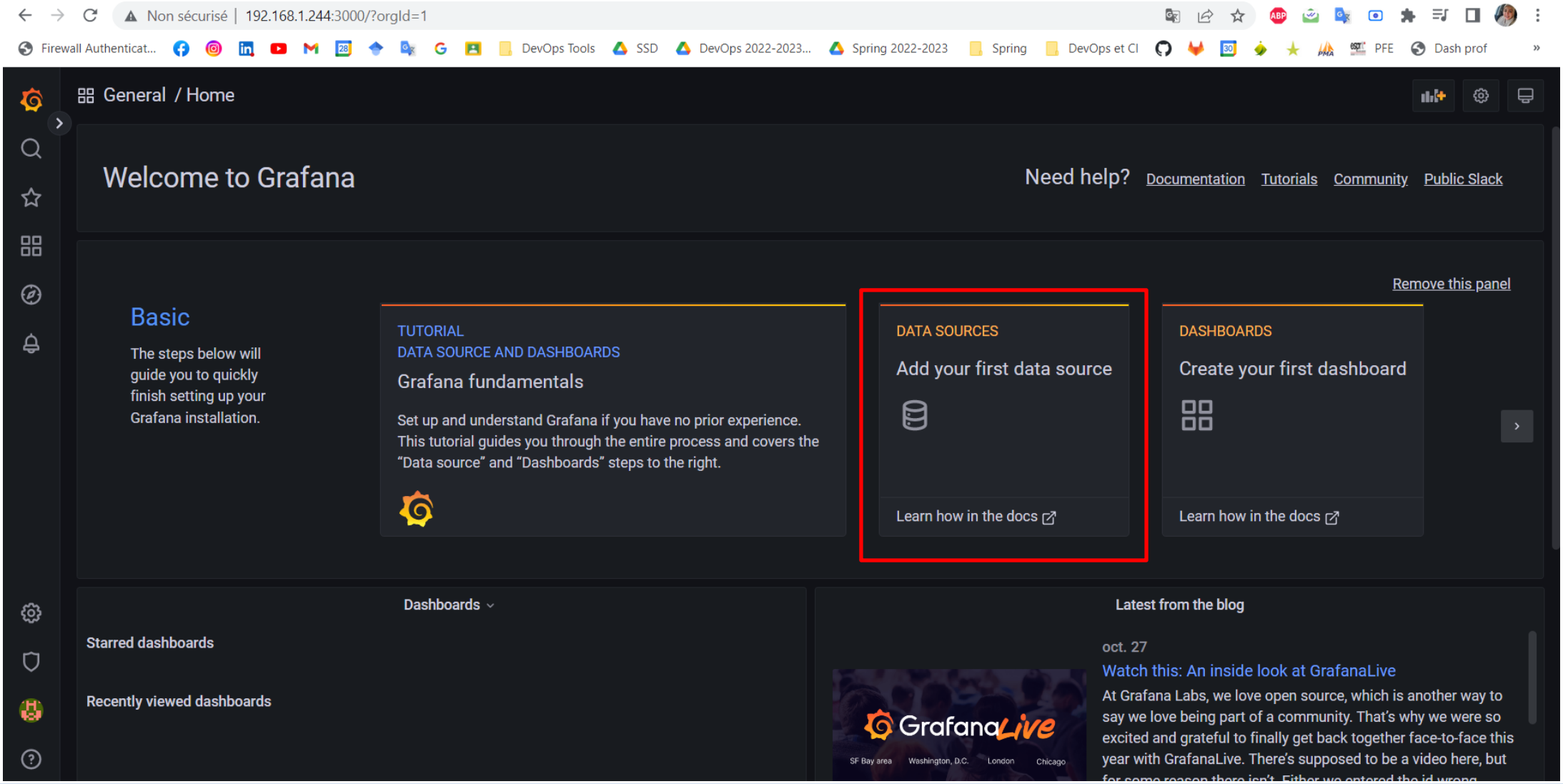


Grafana - Configuration

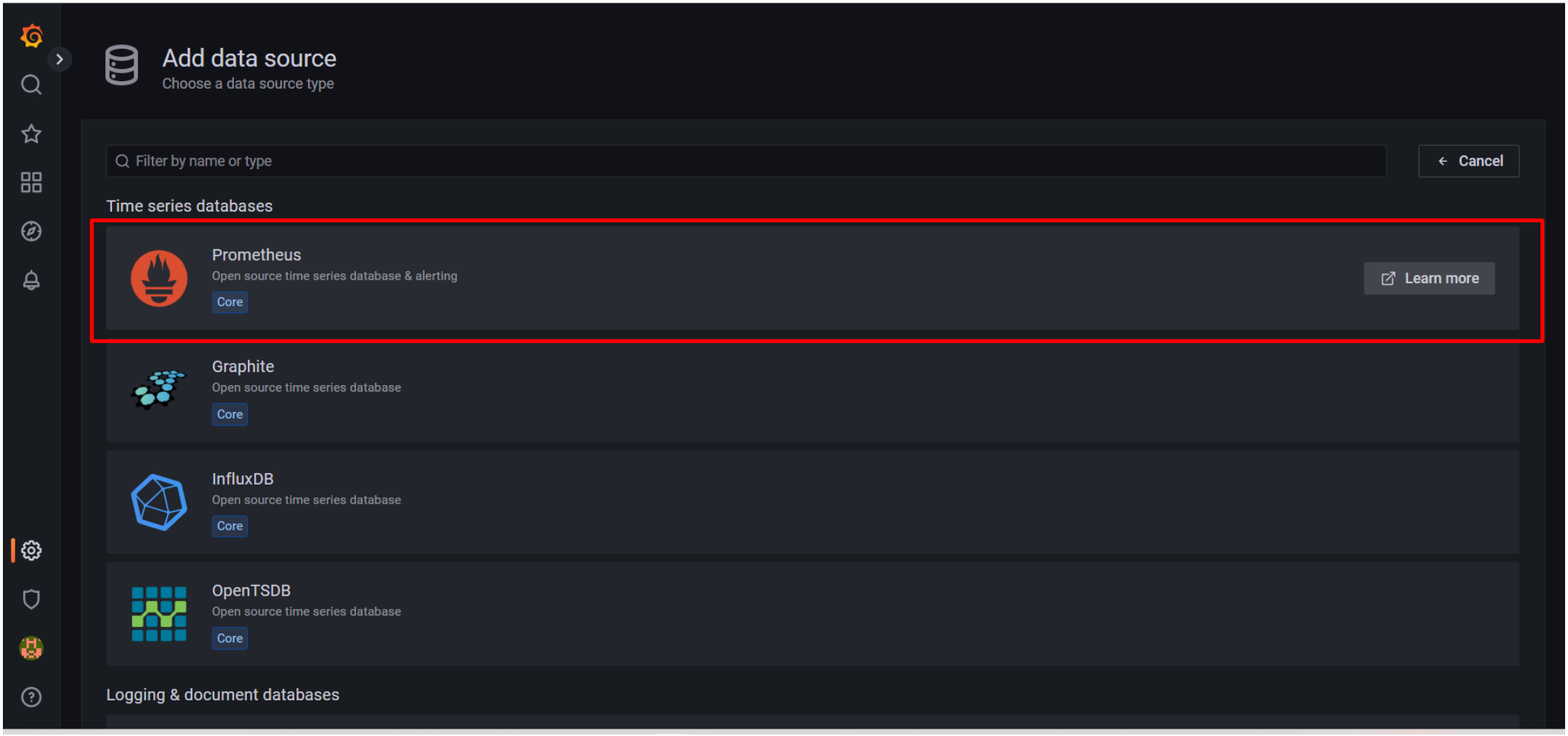


Grafana - Configuration

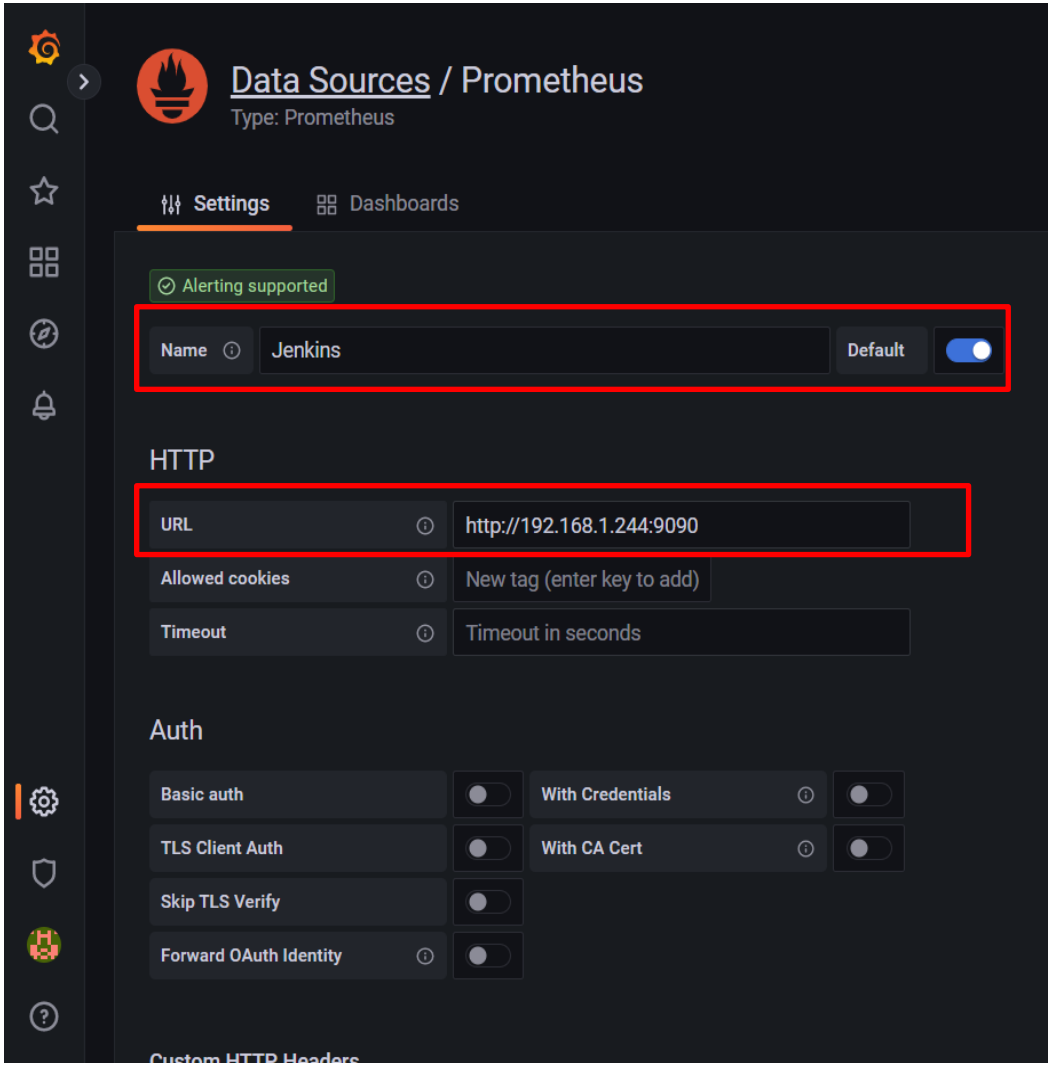
- **Première chose à faire:** Ajouter une source de données

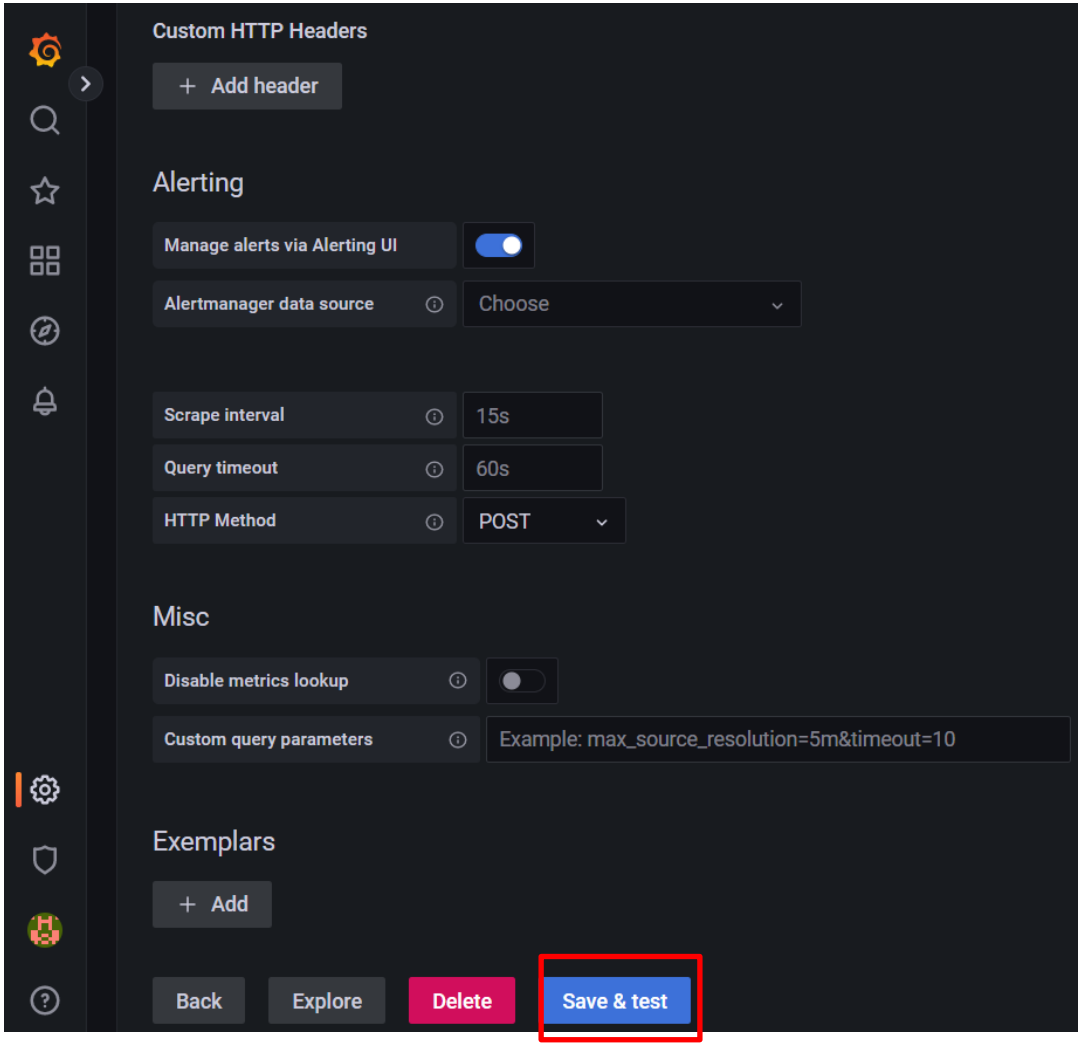


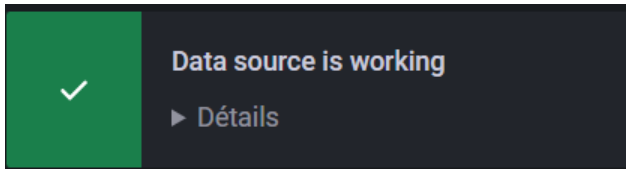
Grafana - Configuration



Grafana - Configuration

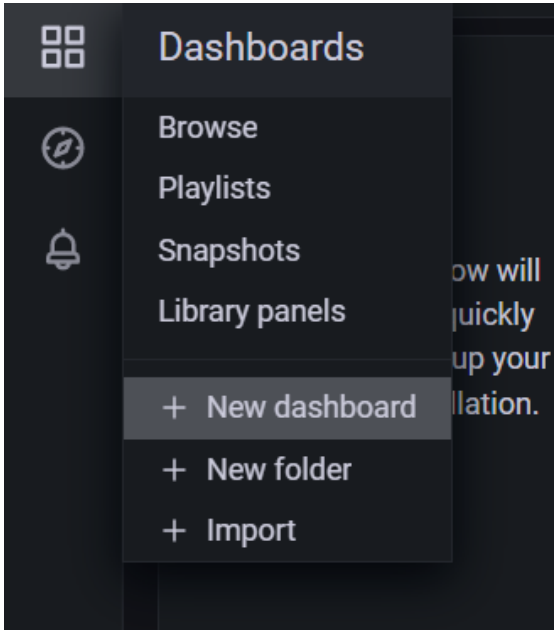


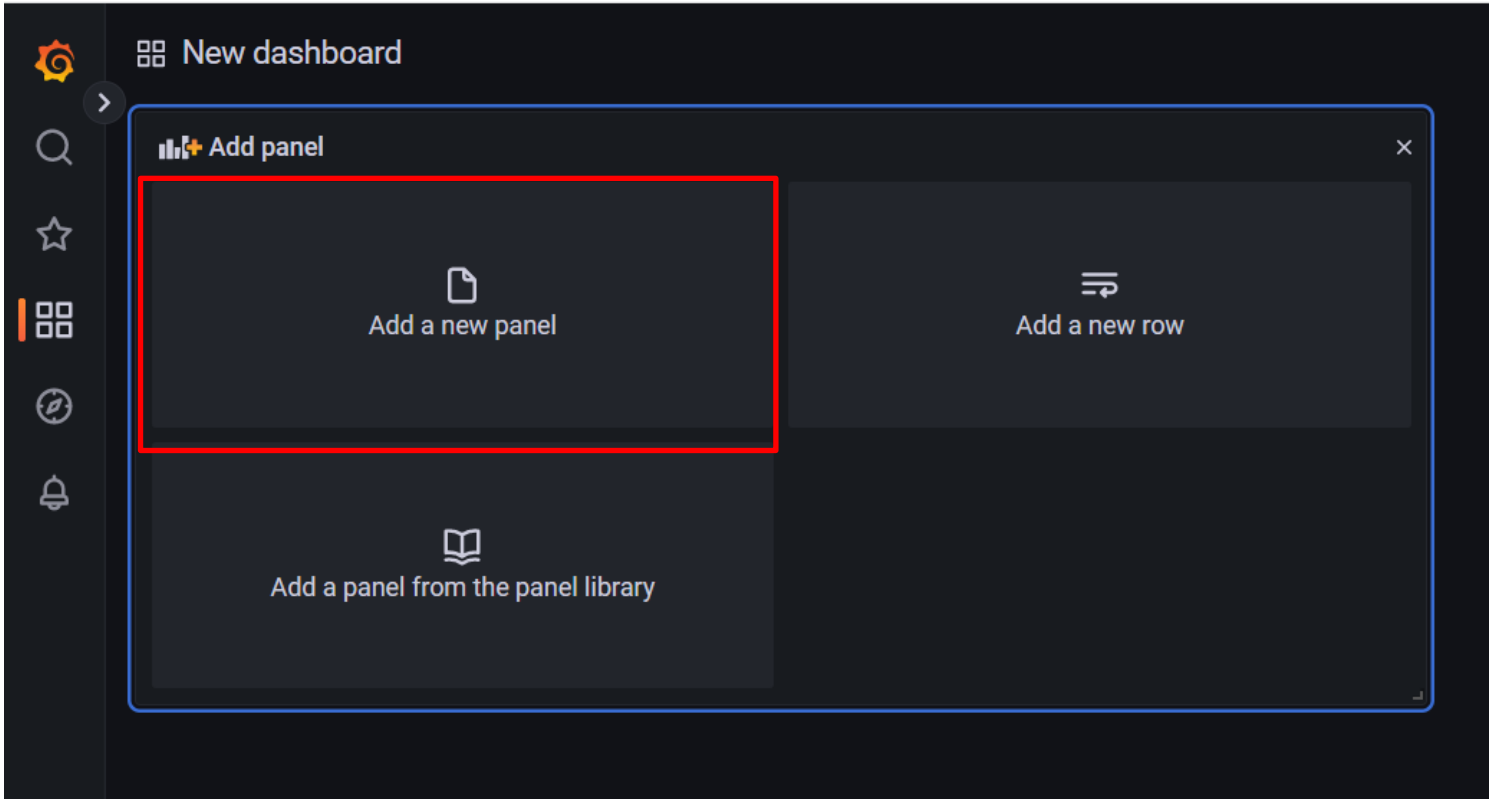




Grafana - Configuration

- **Deuxième chose à faire:** Représenter les données
  - ✓ 1ère manière: Créer un dashboard

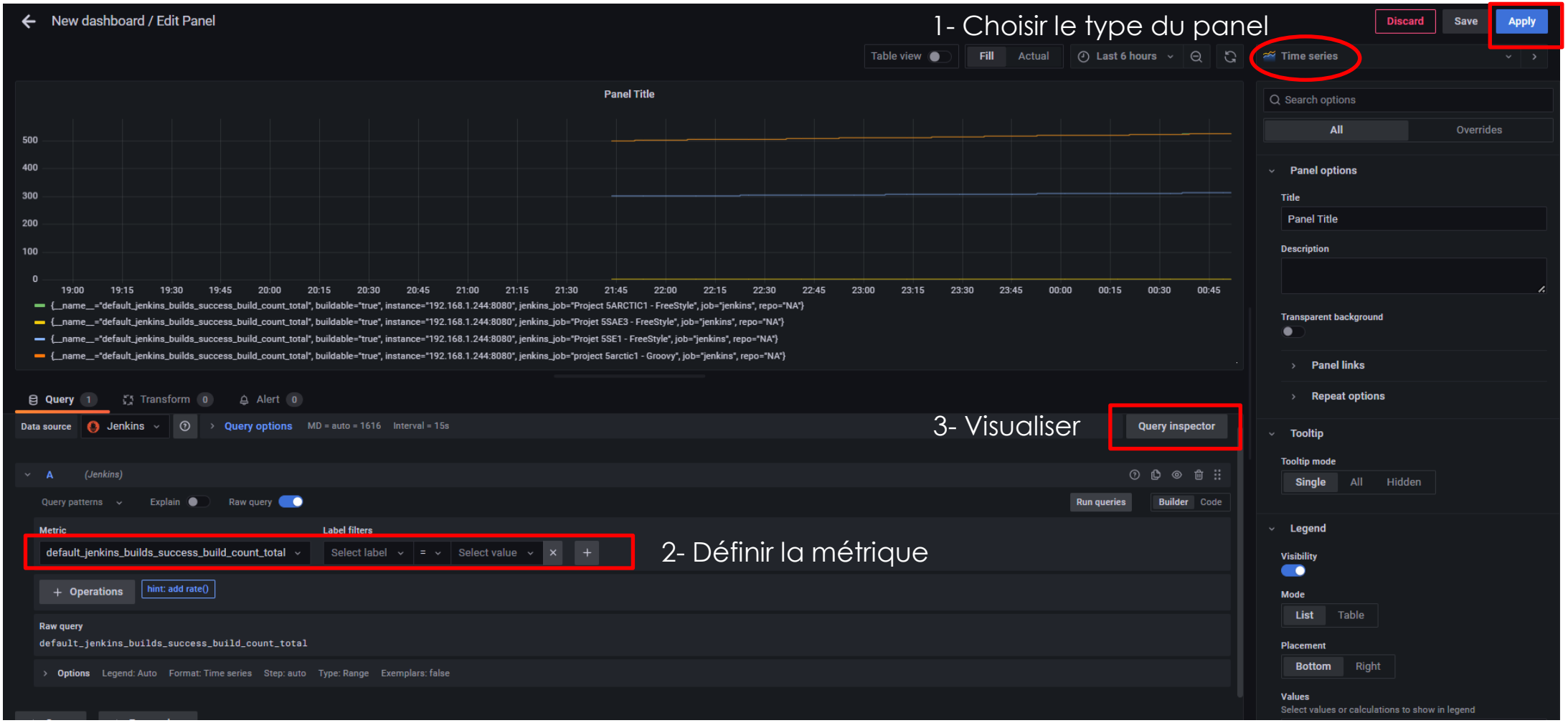




Grafana - Configuration

✓ 1ère manière: Créer un dashboard

4- Sauvgarder

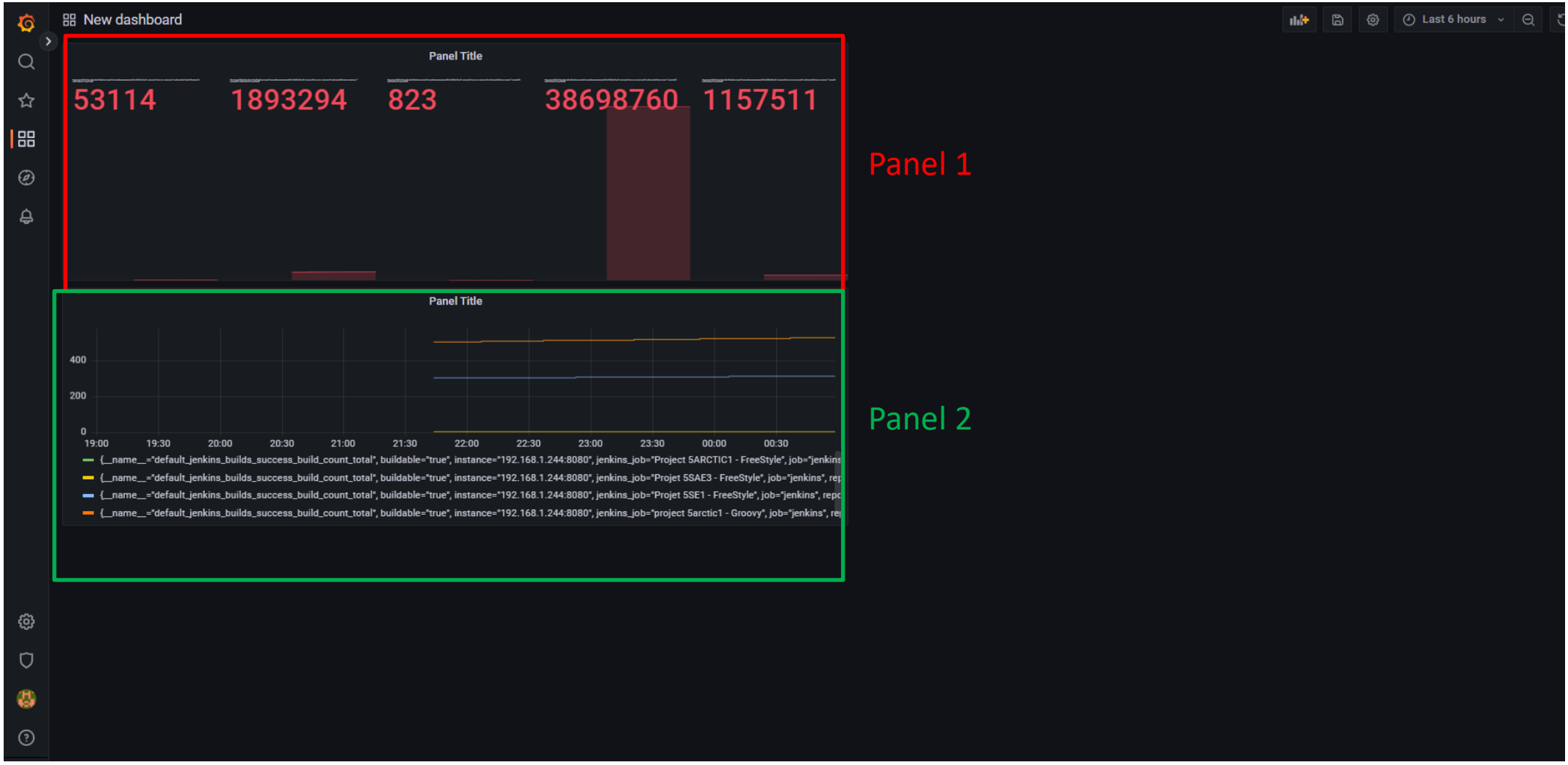


Grafana - Configuration

✓ 1ère manière: Créer un dashboard

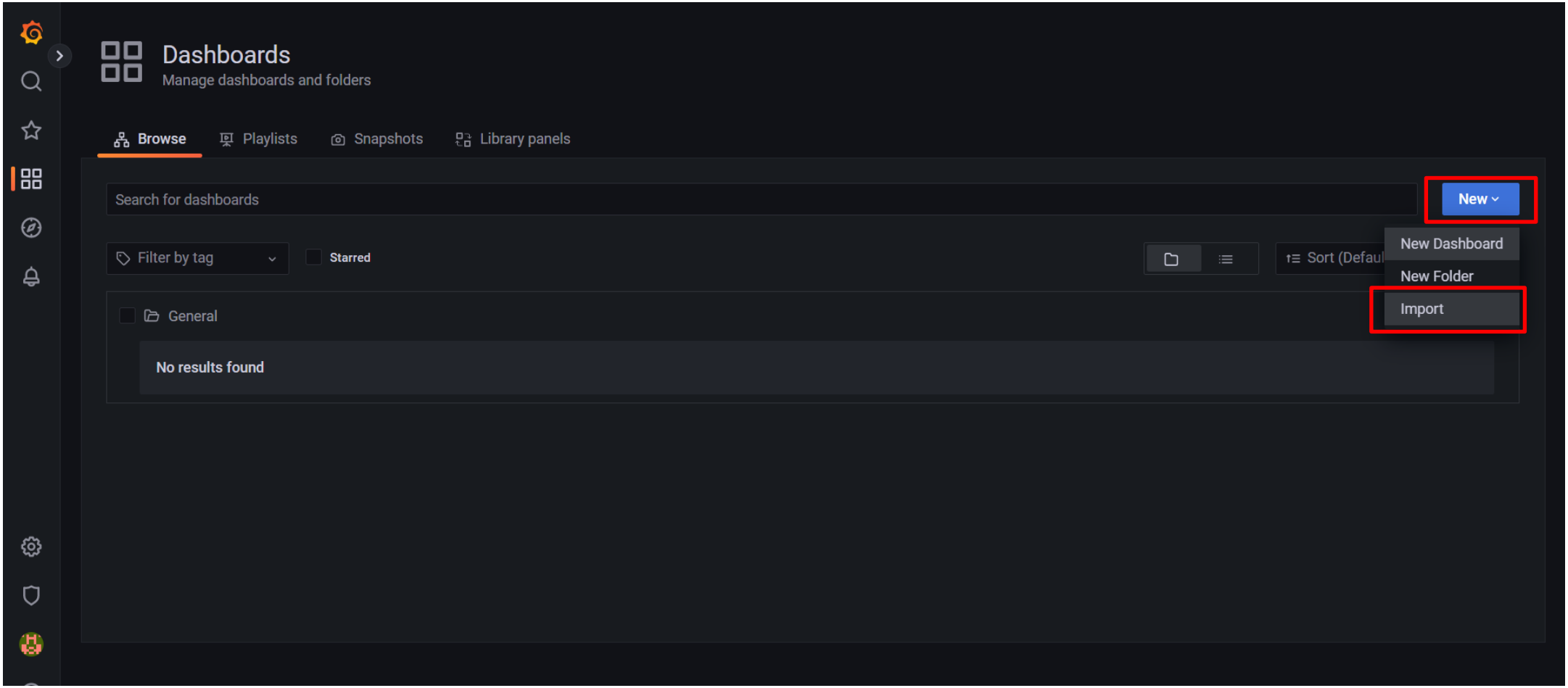
Grafana - Configuration

✓ 1ère manière: Créer un dashboard



Grafana - Configuration

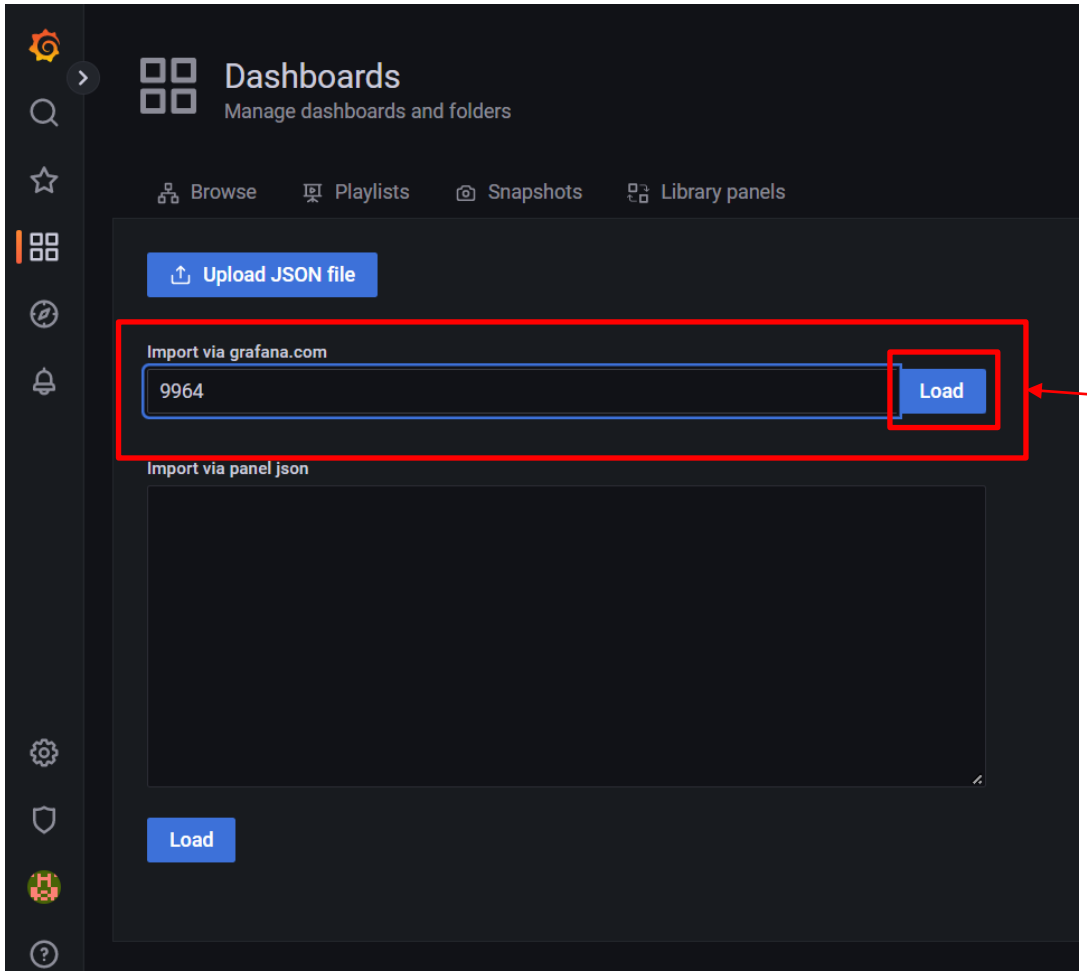
- **Deuxième chose à faire:** Représenter les données
  - ✓ 2ème manière: Importer un dashboard



Grafana - Configuration

✓ 2ème manière: Importer un dashboard

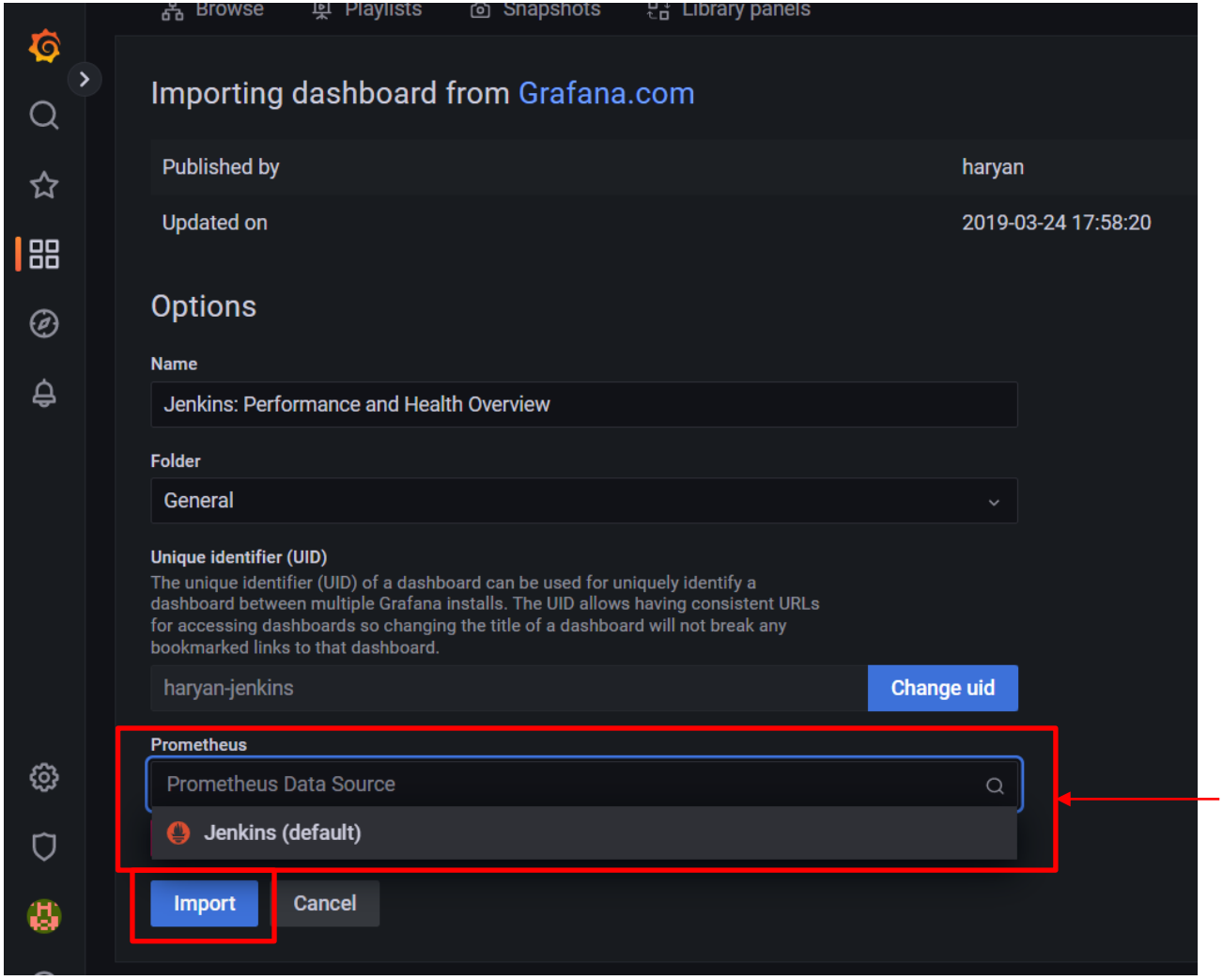
Choisir un template: https://grafana.com/grafana/dashboards/



Saisir l'identifiant du template

Grafana - Configuration

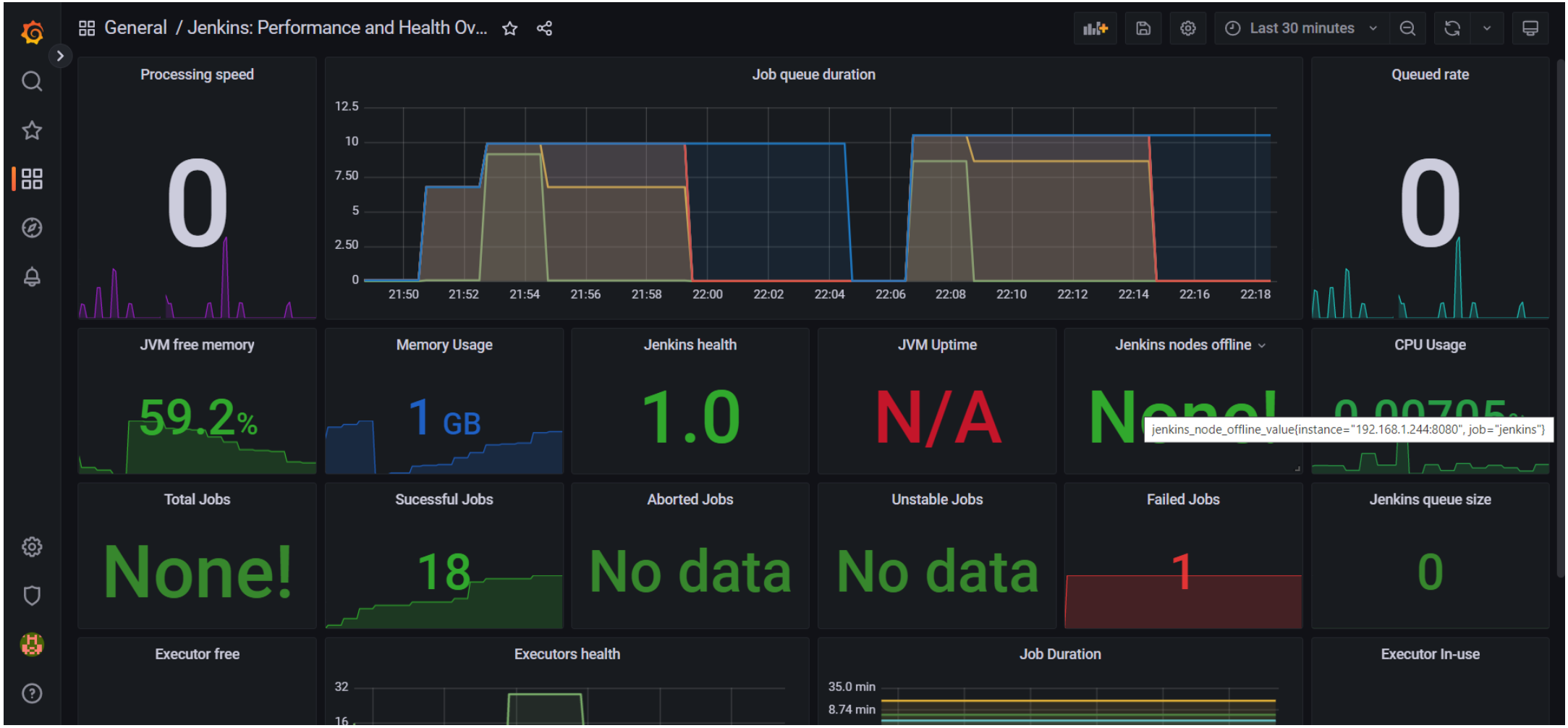
✓ 2ème manière: Importer un dashboard



Choisir la source des données

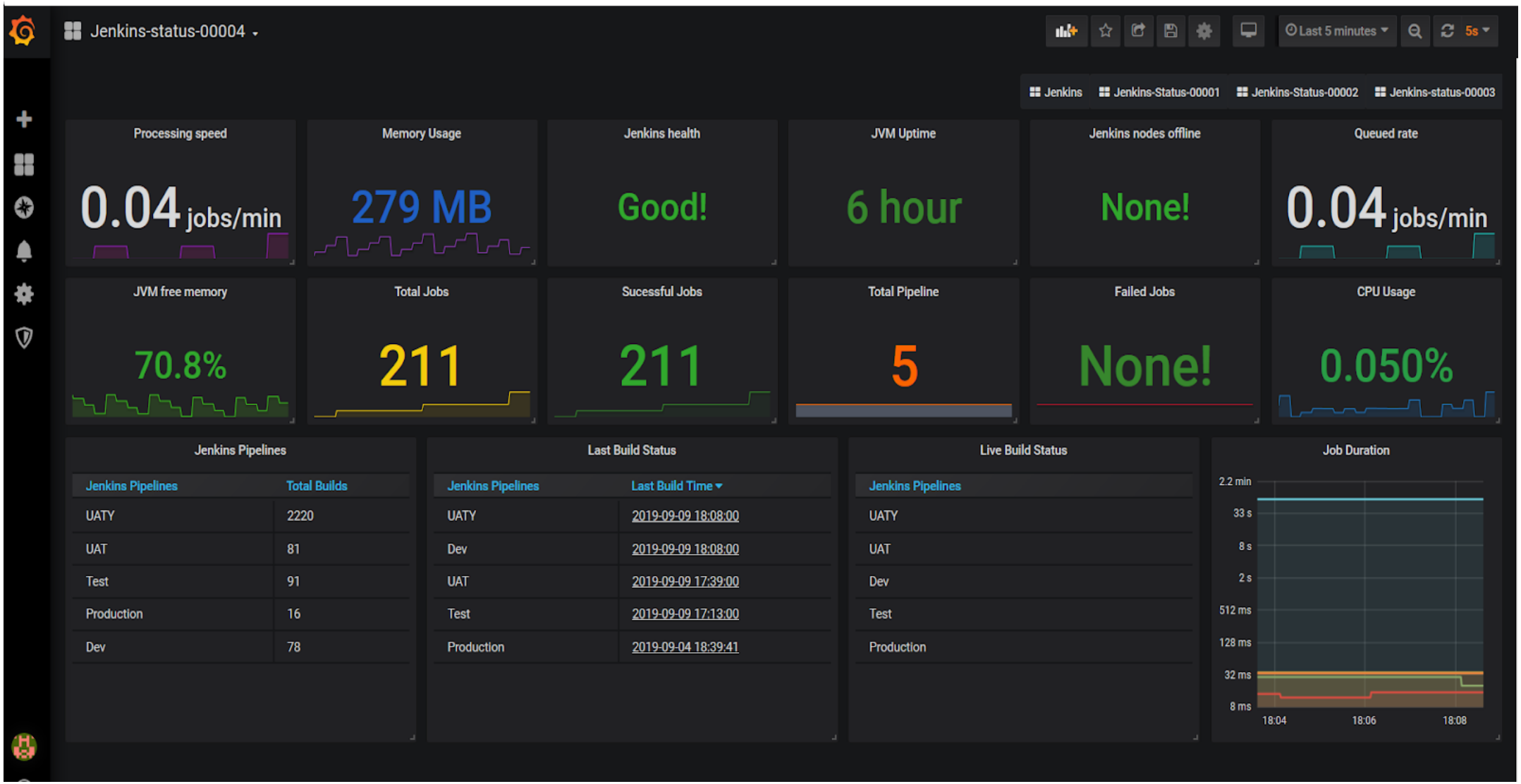
Grafana - Configuration

✓ 2ème manière: Importer un dashboard

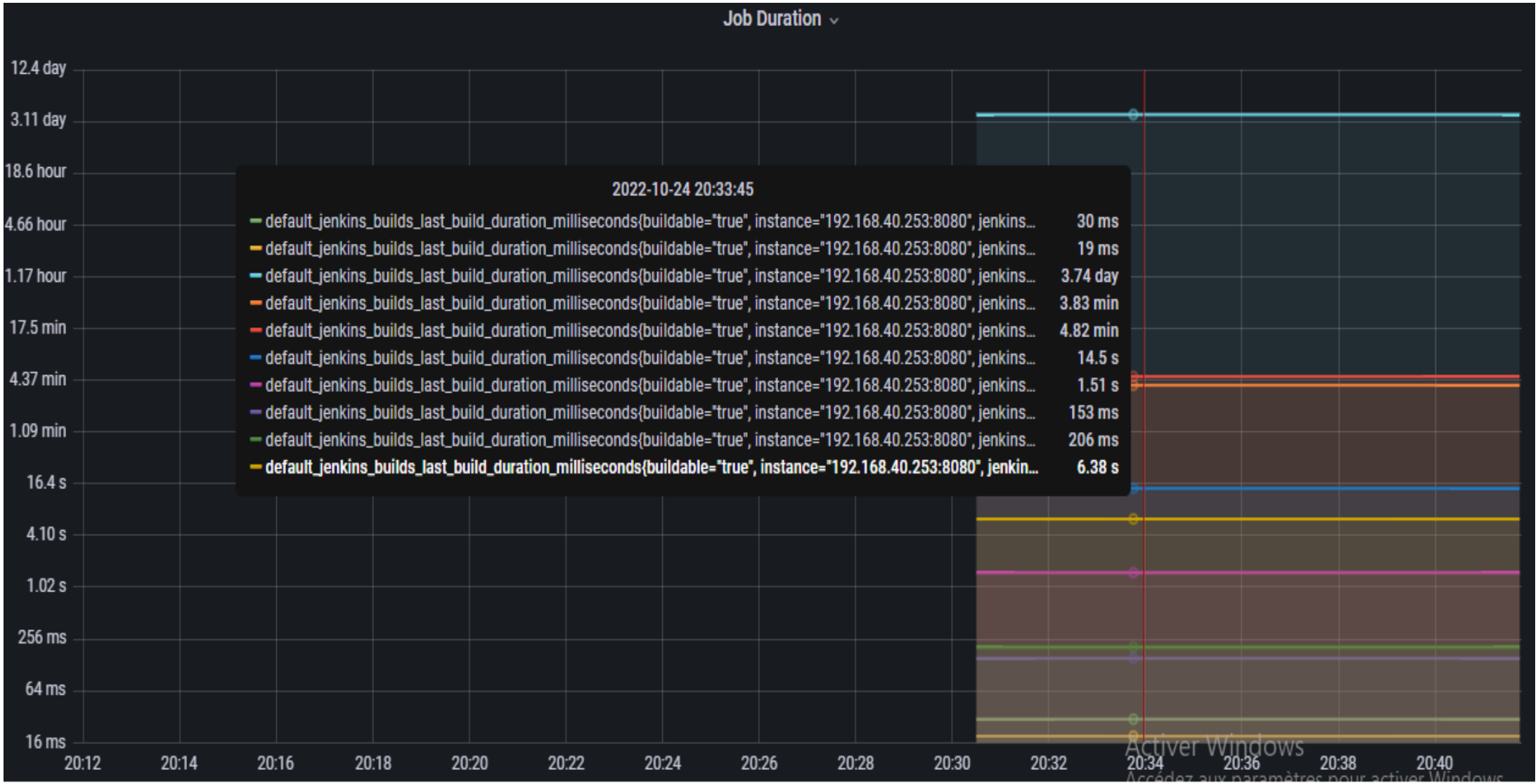


Grafana - Configuration

✓ 2ème manière: Chaque dashboard affiche des métriques bien précises

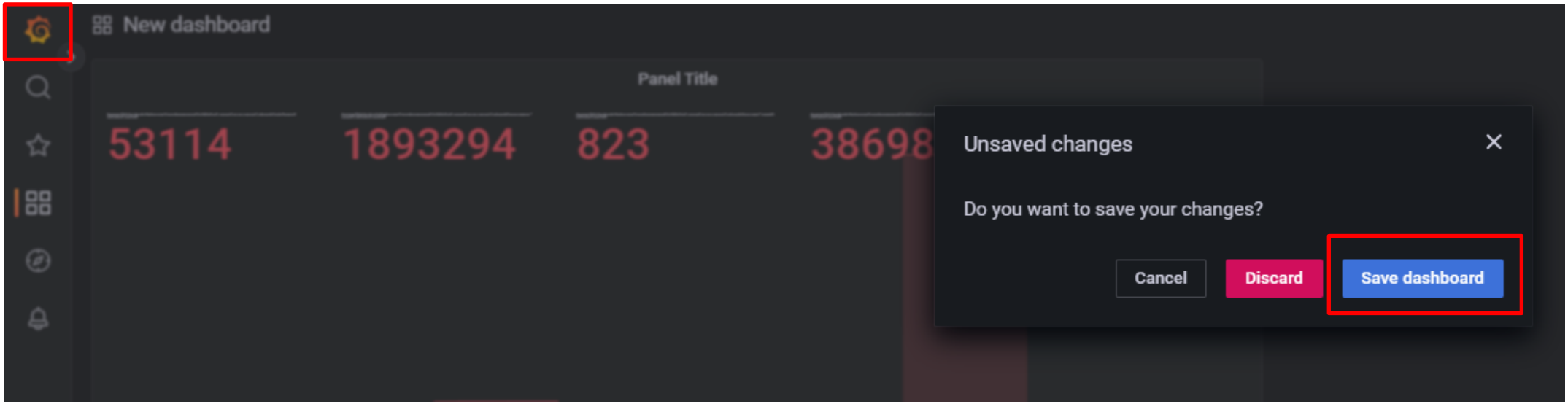


Grafana - Configuration



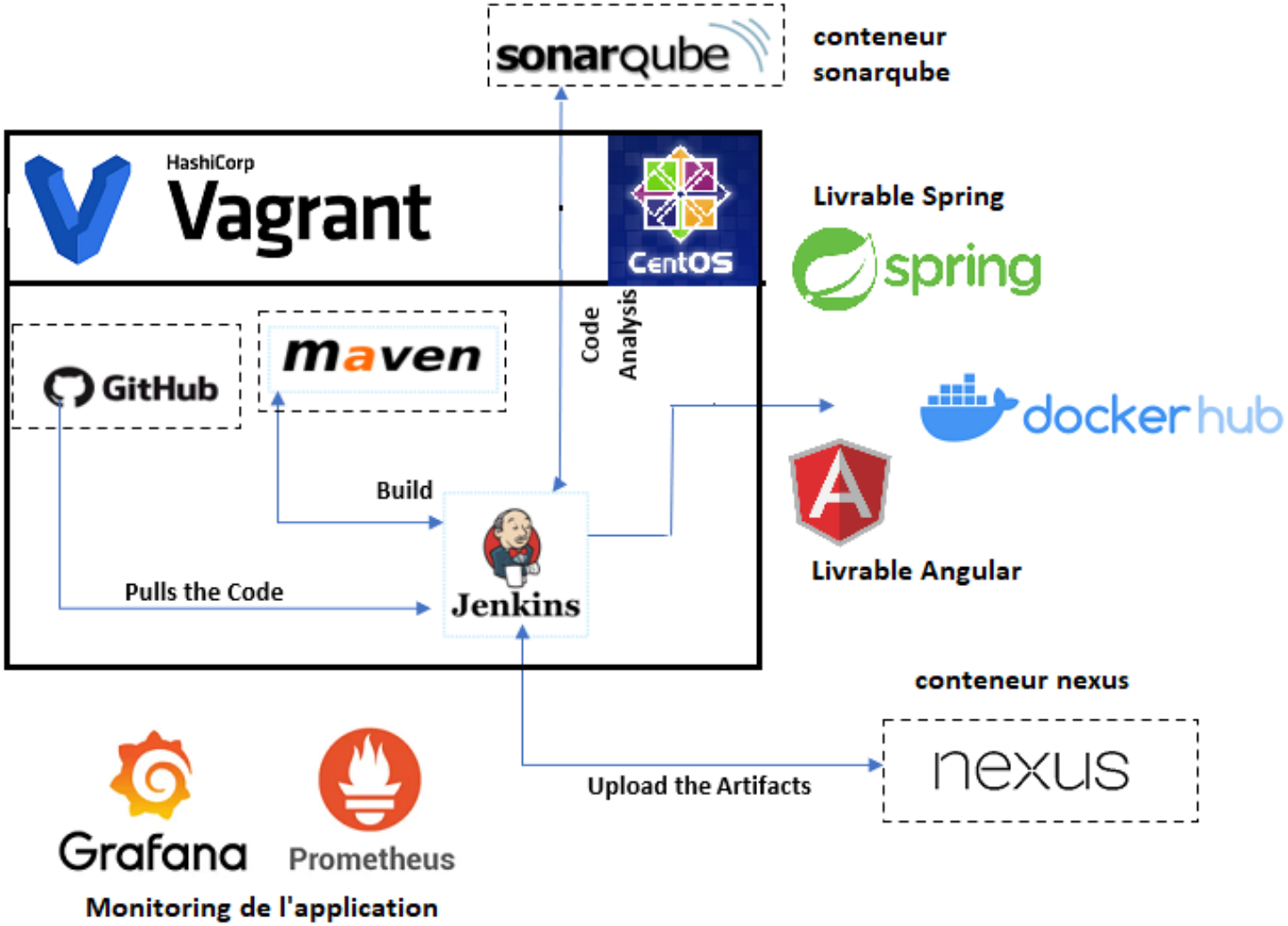
Grafana - Configuration

Pour sauvgarder le dashboard



Travail à faire

- Créer les dashborads pour superviser les différents serveurs.



Monitoring: Prometheus et Grafana

Si vous avez des questions, n'hésitez pas à nous contacter :

Département Informatique
UP Architectures des Systèmes d'Information

Bureau E204

Monitoring: Prometheus et Grafana

UP ASI
Bureau E204