PROMETHEUS - GRAFANA

```
- type: formation
  duration: 3d
  monitoring: systems
```

PRÉSENTATION

- Du formateur
- De **DocDoku**

À PROPOS DE VOUS

- Expérience avec les outils de supervision
- Ce que vous attendez de la formation
- Vos projets à venir utilisant Prometheus/Grafana

AGENDA

- [1] Introduction
- [2] Installation et configuration de Prometheus
- [3] Installation et configuration de Grafana
- [4] Requêtes PromQL
- [5] Alerting
- [6] Bonnes pratiques
- [7] Production et mise à l'échelle

DÉROULEMENT DES TP

Utilisation de machines virtuelles et conteneurs, déploiements de dashboards, mise en place d'alertes et optimisations.

MODALITÉS

- Horaires : 9h -> 17h30
- Pauses : 15 minutes matin et après midi
- Déjeuner : 1h30

[1] INTRODUCTION

- Présentation de prometheus
- Fonctionalités
- Les métriques
- Architecture
- Cas d'usages

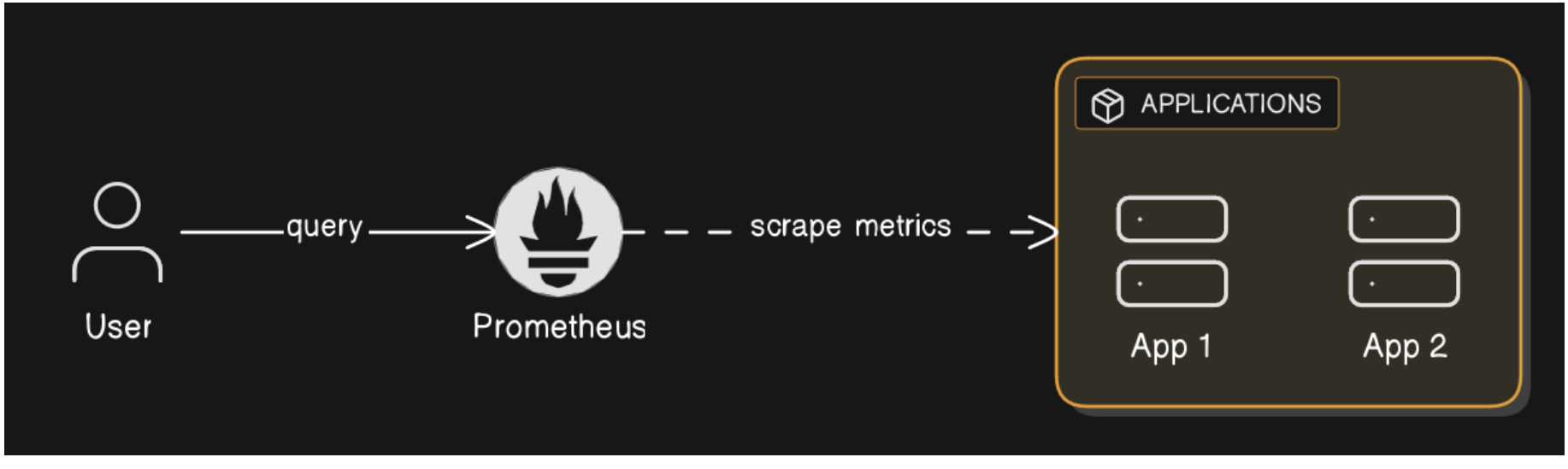
PRÉSENTATION

- Outil **open-source** de monitoring et d'alertes
- Origine : SoundCloud (2012)
- Adoption large par entreprises et organisations
- Communauté très active
- Projet autonome, indépendant
- Intégration **CNCF** en 2016
- Deuxième projet CNCF après Kubernetes
- Stockage **métriques** : séries temporelles
- Données avec **horodatage** + étiquettes (labels)

CNCF: Cloud Native Computing Foundation

FONCTIONNALITÉS DE BASE

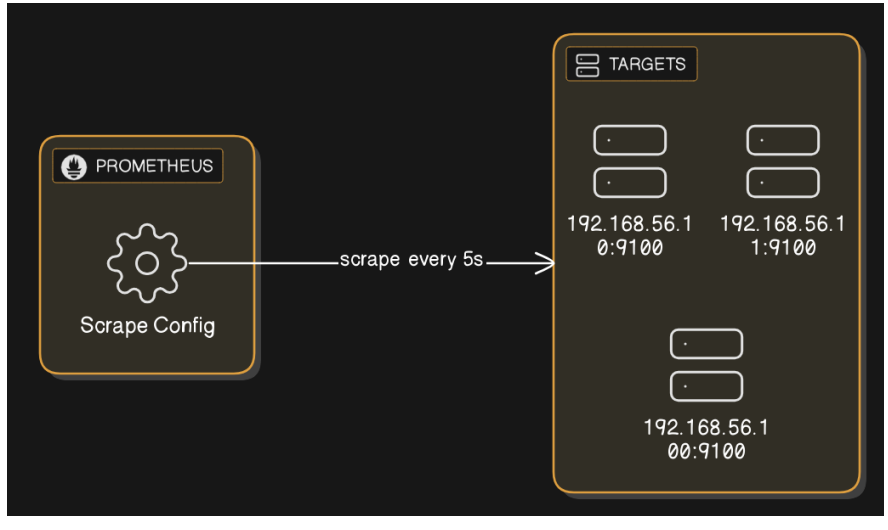
- **Collecte** des **métriques** (données numériques) à partir de différentes sources,
- **Stocke** les métriques,
- Permet de les **interroger**, de les **visualiser** et de déclencher des **alertes**.



SCRAPE - MODÈLE "PULL"

Prometheus "tire" (**scrape**) les métriques directement depuis les cibles configurées à **intervalles réguliers**.

```
scrape_configs:
  - job_name: my-job-name
    scrape_interval: 5s
    static_configs:
      - targets:
          - 192.168.56.10:9100
          - 192.168.56.11:9100
          - 192.168.56.100:9100
```



Fichier: **/etc/prometheus/prometheus.yml**

*Regroupement de cibles par **jobs***

LES MÉTRIQUES

- **Séries temporelles** : données + horodatage
- **Labels** : paires clé-valeur pour filtrage
- **Métriques Pull** : récupération par Prometheus
- **Format exposition** : endpoints HTTP `/metrics`
- **PromQL** : langage de requêtes métriques

Exemple de métriques collectées (format texte)

```
http_requests_total{method="GET", status="200"} = 15432
cpu_usage_seconds_total{core="0"} = 128.45
memory_usage_bytes{process="backend"} = 734003200
```

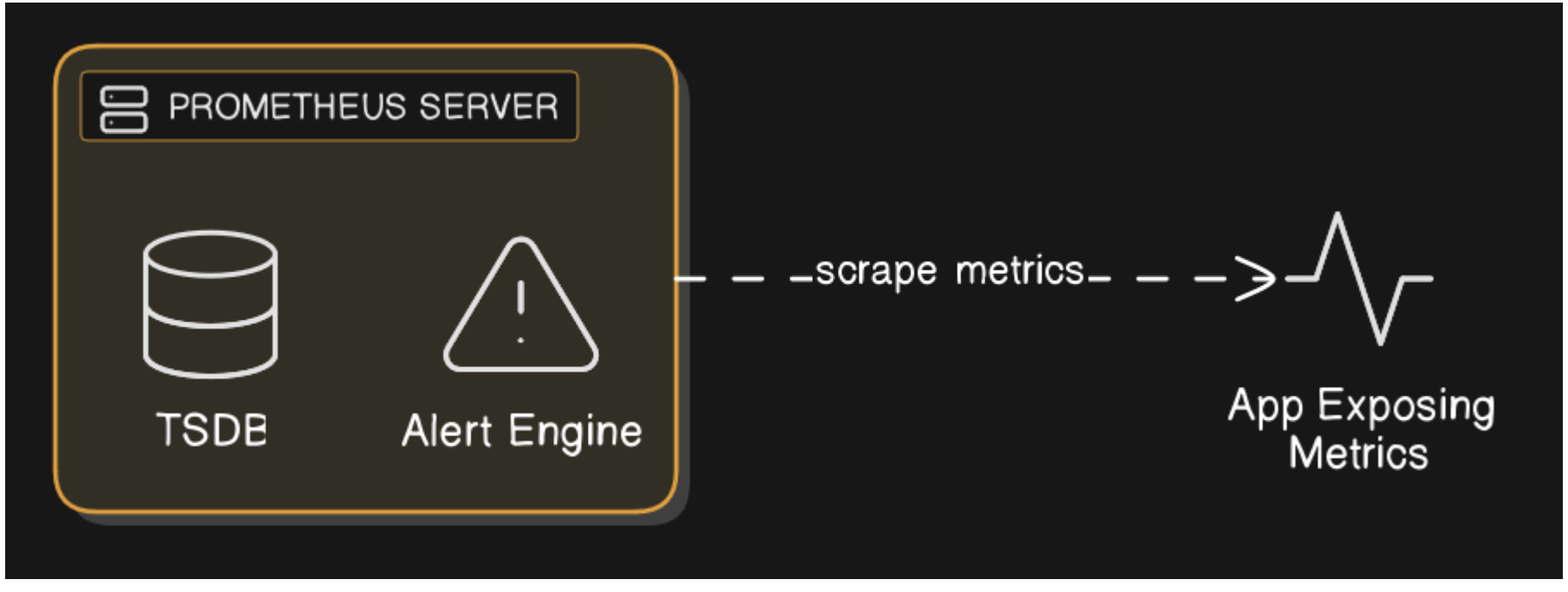
ARCHITECTURE

Prometheus embarque plusieurs composants :

- Serveur
- Exporters
- PushGateway
- AlertManager
- Moteur PromQL

PROMETHEUS SERVEUR

- Scrape les métriques
- Base de données de séries temporelles (TSBD)
- Exécute les règles d'enregistrement et d'alerte.



PROMETHEUS EXPORTERS

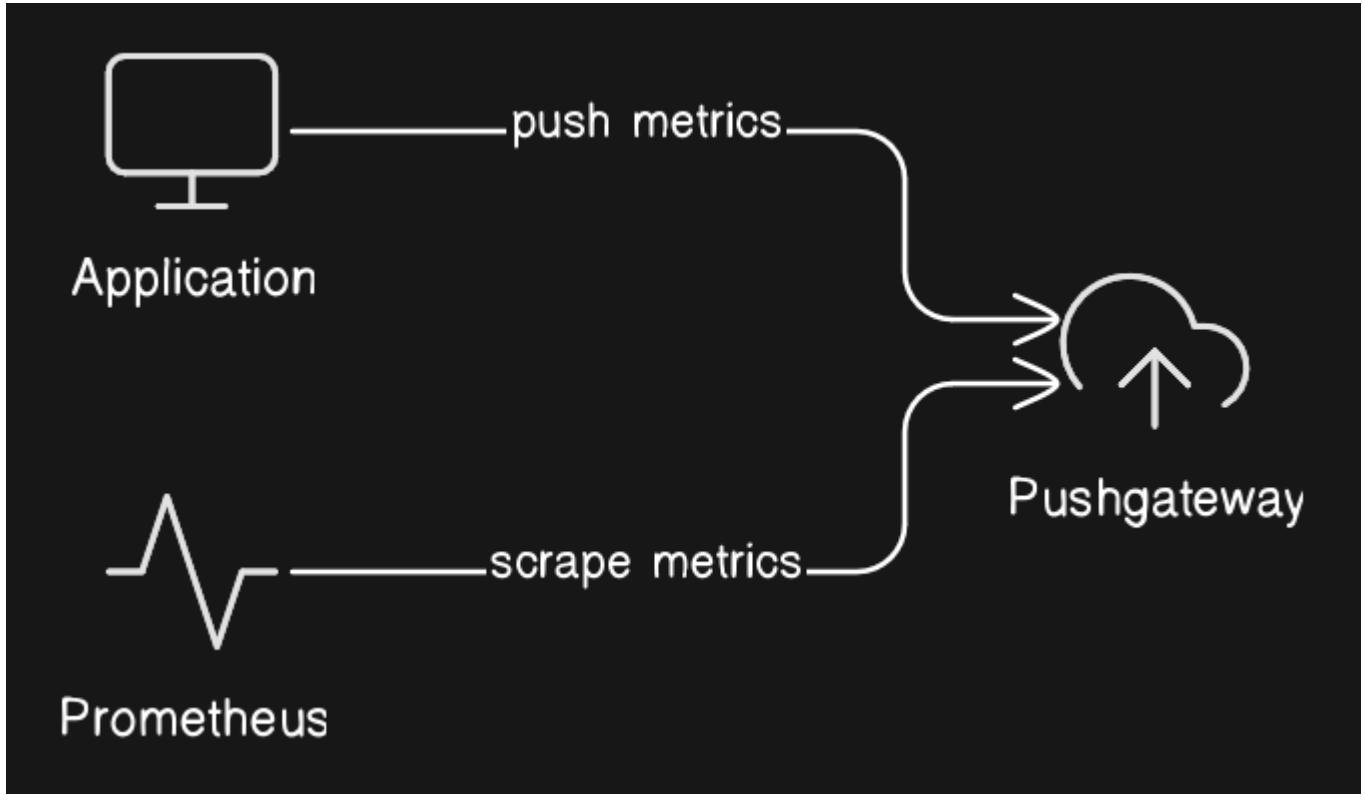
Agents installés sur les **systèmes** ou **applications** à **surveiller**.

- Faible empreinte
- Exposent les métriques (HTTP /metrics)
- Supportent différents types de données :
  - Serveurs Linux (Node Exporter)
  - Bases de données
  - Applications web
  - etc.

PUSHGATEWAY

Serveur d'écoute intégré (mode **push**)

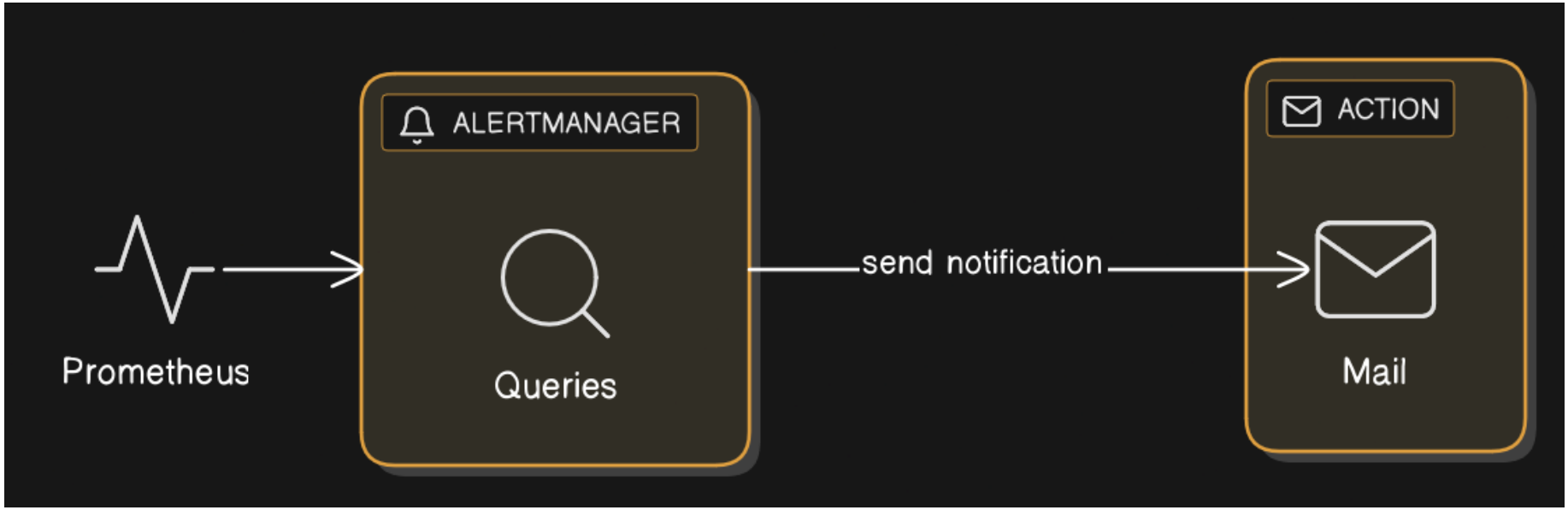
- Permet de collecter des métriques qui ne peuvent pas être scrapés directement par Prometheus.



ALERTMANAGER

Moteur d'alertes.

- Gère les alertes envoyées par le serveur Prometheus.
- Alertes basées sur des conditions (requêtes)
- Actions en sortie (e-mail, Slack, agerDuty, etc.).



PROMQL

Prometheus Query Language

- **Langage** de **requête** puissant et flexible pour interroger les métriques stockées.
- Permet d'**agréger**, de **filtrer** et de **transformer** les données pour créer des **graphiques** ou des **alertes**.

INTERFACE WEB

Prometheus embarque une **interface graphique**, sous forme de webapp

- Tableaux de métriques
- Champs de requête PromQL
- Graphiques séries temporelles
- Résultats en mode “table” ou “graphe”
- Sélection de l'intervalle de temps
- Indicateurs de santé des cibles

INTERFACE WEB : APERÇU

Affichage de la charge (load_1m) de 3 cibles :

CARACTÉRISTIQUES ET AVANTAGES

- **Flexibilité** pour les **requêtes** et l'analyse : métriques paires clé-valeur.
- **Scalabilité** : milliers de cibles, millions de métriques.
- **Intégration** native Kubernetes, **découverte automatique** de nouvelles cibles.
- **Communauté** et **écosystème** : beaucoup d'exporters, d'intégrations et de ressources disponibles.
- Intégration avec **Grafana** : datasource

CAS D'UTILISATION TYPIQUES

- **Surveillance** d'**infrastructures** : Serveurs, machines virtuelles, conteneurs.
- **Surveillance** d'**applications** : Performances, erreurs, latence.
- **Surveillance** de **microservices** : Santé et performance des différents services distribués.
- **Alerting** : Notifications automatiques en cas de problèmes détectés.
- **Analyse** de **performance** et de capacité : Comprendre l'utilisation des ressources et anticiper les besoins futurs.

STANDARD DE FACTO

Prometheus est devenu un standard de facto pour la surveillance dans les environnements cloud-native grâce à sa conception robuste et son adaptabilité.

[2] INSTALLATION ET CONFIGURATION DE PROMETHEUS

- Installation
- Configuration
- Exporters
- Collecte de métriques

INSTALLATION PORTABLE

Dernières version : https://prometheus.io/download/

https://github.com/prometheus/prometheus/releases/download/v3.7.3/prometheus-3.7.3.linux-amd64.tar.gz

Téléchargement de l'archive, extraction, lancement.

```
tar xf prometheus-3.7.3.linux-amd64.tar.gz
cd prometheus-3.7.3.linux-amd64
./prometheus
```

Accéder à http://localhost:9090

INSTALLATION DEPUIS LES DÉPOTS

Disponible dans les dépots Debian (trixie)

```
sudo apt install prometheus -y
```

Sera installé en tant que service.

```
systemcl status prometheus
```

Accéder à http://localhost:9090

INSTALLATION AVEC DOCKER

Étapes

- Télécharger l'image **officielle** de Prometheus.
- Créer un fichier de configuration `prometheus.yml`.
- Lancer avec **docker run** ou **docker compose**.
- Ouvrir l'interface sur le **port 9090**.

```
docker run \
    --name prometheus \
    --detach \
    -p 9090:9090 \
    -v prometheus.yml:/etc/prometheus/prometheus.yml \
    prom/prometheus
```

Accéder à http://localhost:9090

INSTALLATION AVEC DOCKER COMPOSE

Utilisation dans docker compose

```yaml
version: '3'

services:
  prometheus:
    image: prom/prometheus:latest
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"
```

```
docker compose up -d
```

```
docker logs -f prometheus
```

Accéder à http://localhost:9090

CONFIGURATION

La **configuration** de prometheus se fait via un **fichier** (prometheus.yml), ou bien dans les **options** de la ligne de commande.

Sections :

- Global (paramètres général)
- Scrape config (liste des cibles)
- Alerting (spécifique alertmanager)
- Storage (tsbd)

PROMETHEUS.YML

Fichier de **configuration**. Exemple basique pour récupérer des métriques :

```yaml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: my-metrics
    static_configs:
      - targets:
          - 192.168.56.10:9100
          - 192.168.56.11:9100
          - 192.168.56.100:9100
        labels:
          group: my-group
          custom_label: custom_value
```

Support de labels custom pour filtrer nos requêtes !

EXPORTERS

De nombreux exporters sont présents dans les **dépots officiels**. Dépots Debian (trixie) :

```
apt search prometheus exporter

prometheus-apache-exporter
prometheus-bind-exporter
prometheus-dnsmasq-exporter
prometheus-elasticsearch-exporter
prometheus-mongodb-exporter
prometheus-nginx-exporter
...
```

Les **métriques** exposées sont **préfixées** :

```
apache_up 1
elasticsearch_cluster_health_up 1
...
```

EXPORTERS

Liste complète :

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

Bases de données, métriques hardware, gestionnaires de tickets, intégration continue, brokers, file storage, serveurs web, APIs, loggers, etc.

NODE EXPORTER

Objectif

- Exporter des **métriques système** (CPU, RAM, disque).
- Fonctionne avec **Prometheus** par défaut.

Installation sur serveur Debian

```
sudo apt install -y prometheus-node-exporter
```

Ecoute sur le port 9100

```
curl localhost:9100/metrics
node_cpu_seconds_total{cpu="0",mode="softirq"} 2.9
node_cpu_seconds_total{cpu="0",mode="user"} 63.74
node_cpu_seconds_total{cpu="1",mode="idle"} 6608.99
node_cpu_seconds_total{cpu="1",mode="iowait"} 2.04
node_cpu_seconds_total{cpu="1",mode="irq"} 0
...
```

LANCER NODE EXPORTER AVEC DOCKER

Nécessite de passer quelques paramètres supplémentaires :

```
docker run -d \
  --net="host" \
  --pid="host" \
  -v "/:/host:ro,rslave" \
  quay.io/prometheus/node-exporter:latest \
  --path.rootfs=/host
```

- Ouvrir dans un navigateur : http://localhost:9100/metrics
- La page doit afficher des **métriques brutes**.

AJOUT DANS PROMETHEUS

Configuration prometheus

- Ajouter **Node Exporter** comme cible Prometheus.
- Utiliser un **job** dans `prometheus.yml`.

```
scrape_configs:
  - job_name: my-machines
    static_configs:
      - targets:
        - machine1:9100
        - machine2:9100
        - machine3:9100
```

RECHARGER PROMETHEUS

Selon le type d'installation choisie (exemple docker compose)

```
docker compose restart prometheus
```

✅ Vérification

- Accéder à l'interface : http://localhost:9090/targets
- Vérifier que **my-machines** est dans la liste et **UP**.

🔍 Tester une requête PromQL

```
node_cpu_seconds_total
```

[3] INSTALLATION ET CONFIGURATION DE GRAFANA

- Présentation de Grafana
- Installation
- Sources de données et connexion à Prometheus
- Création de tableaux de bord simples
- Exploration communautaire et partage

PRÉSENTATION DE GRAFANA

- Outil open source de visualisation et d'alertes
- Maintenu par la société GRAFANA LABS
- AGPL-3.0 https://github.com/grafana/grafana

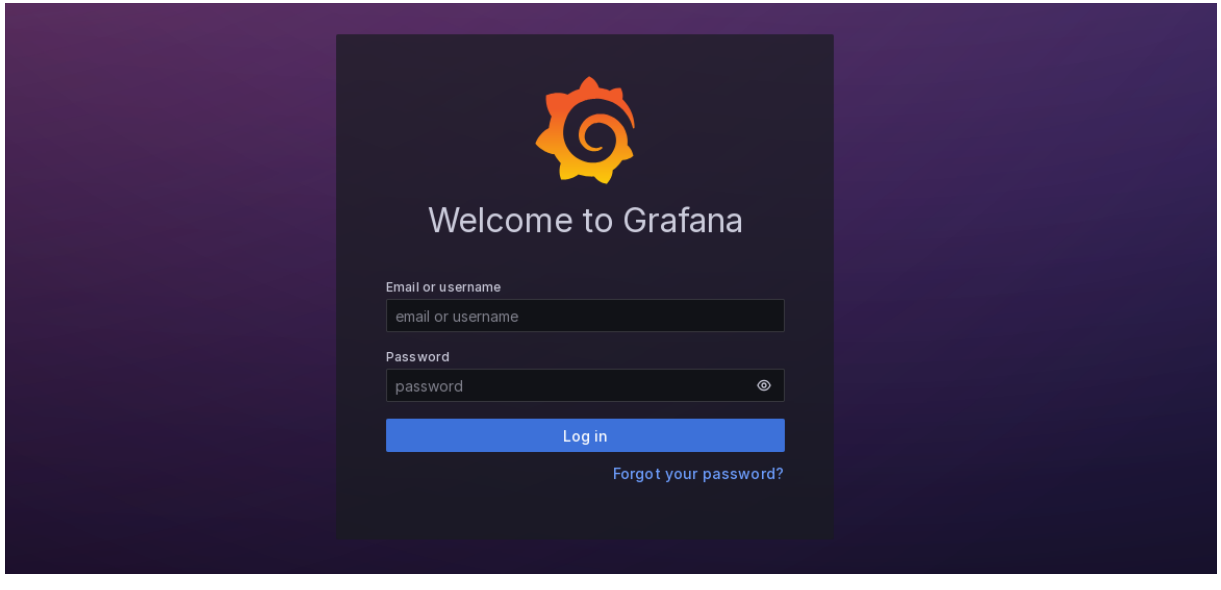
Fonctionalités principales :

- Visualiser les **métriques** de Prometheus.
- Créer des **tableaux de bord** dynamiques.
- Supporte plusieurs **sources de données**.
- Interface **web**, intuitive et personnalisable.

INTERFACE GRAPHIQUE

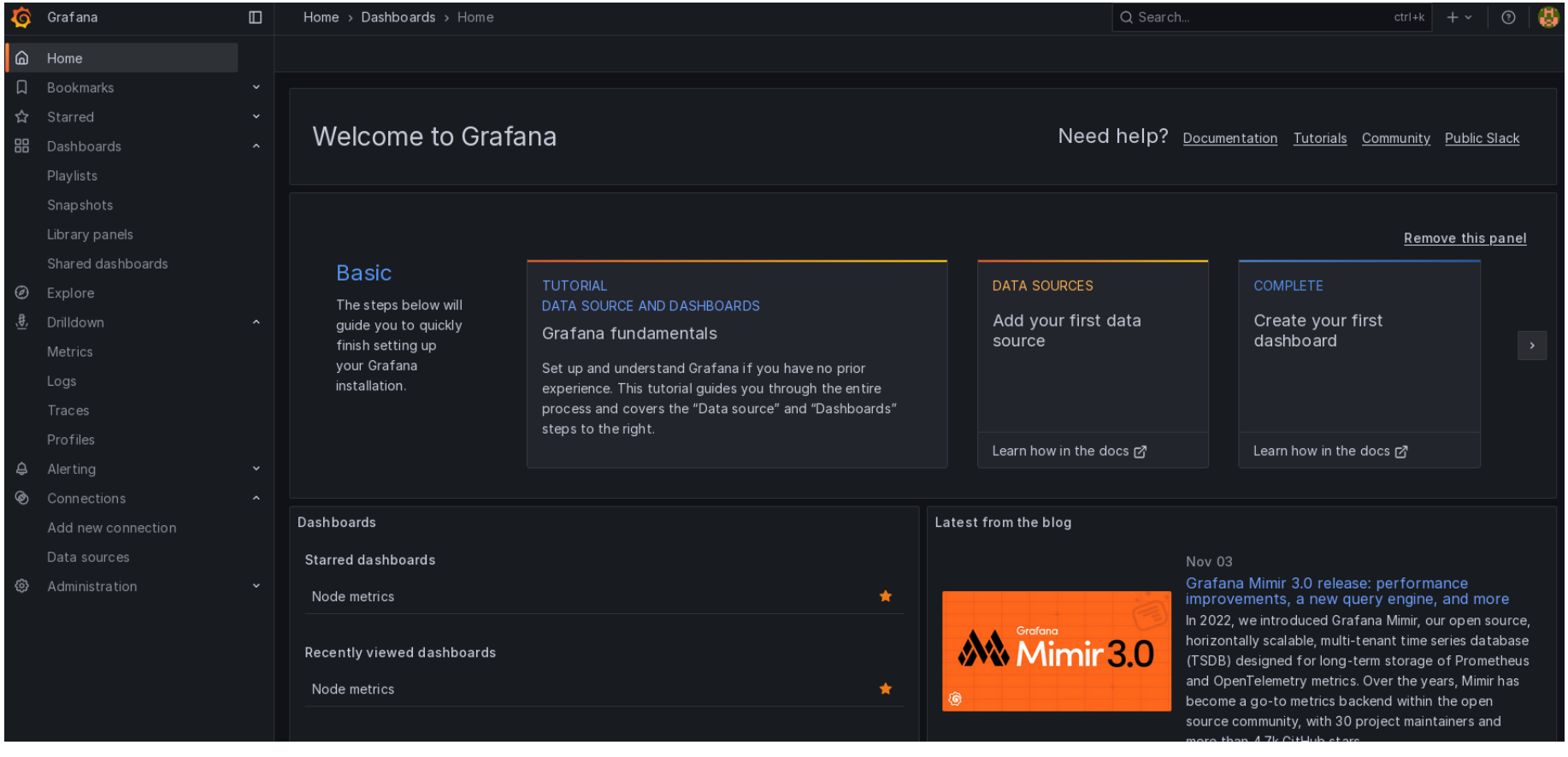
Interface WEB moderne

- Accès local : http://localhost:3000
- Login par défaut : **admin / admin**
- Modifier le mot de passe au **premier accès**



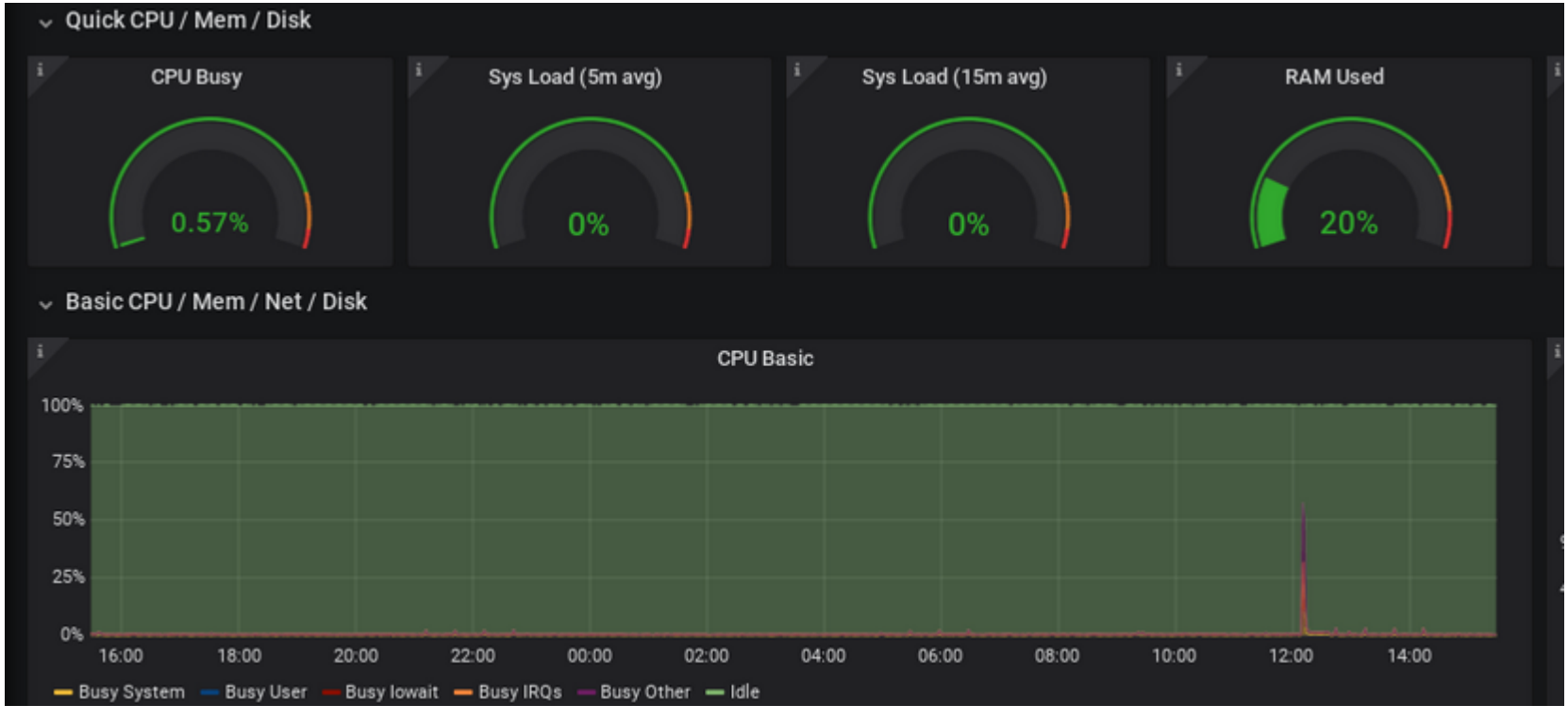
PAGE D'ACCUEIL

Home page avec accès aux principales fonctionalités (dashboards, datasources)



FONCTIONNALITÉS CLÉS

- Graphiques **temps réel**.
- Support de **PromQL**.
- Création de **panels**, **alertes**, **dashboards partagés**.
- Intégration facile avec **Prometheus**, **Loki**, etc.



INSTALLATION PORTABLE

https://grafana.com/grafana/download

Dernière version https://dl.grafana.com/grafana/release/12.2.1/grafana_12.2.1_18655849634_linux_amd64.tar.gz

Téléchargement, extraction, et lancement :

```
tar xf grafana_12.2.1_18655849634_linux_amd64.tar.gz
cd grafana-12.2.1
bin/grafana server
```

Accéder à http://localhost:3000

INSTALLATION DEPUIS LES DÉPOTS

Non disponibles dans les dépots officiel Debian (trixie).

```
# Outillage d'installation
sudo apt-get install -y apt-transport-https software-properties-common wget

# Ajout des dépots grafana
sudo mkdir -p /etc/apt/keyrings/
wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor | \
  sudo tee /etc/apt/keyrings/grafana.gpg > /dev/null
echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg]
 https://apt.grafana.com stable main" | \
  sudo tee -a /etc/apt/sources.list.d/grafana.list
echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg]
 https://apt.grafana.com beta main" | \
  sudo tee -a /etc/apt/sources.list.d/grafana.list

# Installation
sudo apt-get update && sudo apt-get install grafana -y
```

INSTALLATION AVEC DOCKER

Création d'un volume pour la persistance des données (dashboards, alertes, etc.)

```
docker volume create grafana-storage
docker run -d \
  --name=grafana \
  -v grafana-storage:/var/lib/grafana
  -p 3000:3000 \
  grafana/grafana
```

Accéder à http://localhost:3000

INSTALLATION AVEC DOCKER COMPOSE

```yaml
version: '3'

services:
  grafana:
    image: grafana/grafana
    ports:
      - "3000:3000"
    volumes:
      - grafana-storage:/var/lib/grafana

volumes:
  grafana-storage:
```

```
docker compose up -d
```

Accéder à http://localhost:3000

CONFIGURATION

Chemin par défaut `/etc/grafana/grafana.ini`

Surcharge de l'emplacement en ligne de commande avec l'option `--config`

Surcharge des paramètres en environement. Les variables sont préfixées par `GF_`, exemples :

```
export GF_DEFAULT_INSTANCE_NAME=my-instance
export GF_SECURITY_ADMIN_USER=owner
export GF_AUTH_GOOGLE_CLIENT_SECRET=newS3cretKey
export GF_PLUGIN_GRAFANA_IMAGE_RENDERER_RENDERING_IGNORE_HTTPS_ERRORS=true
export GF_FEATURE_TOGGLES_ENABLE=newNavigation
```

GRAFANA.INI

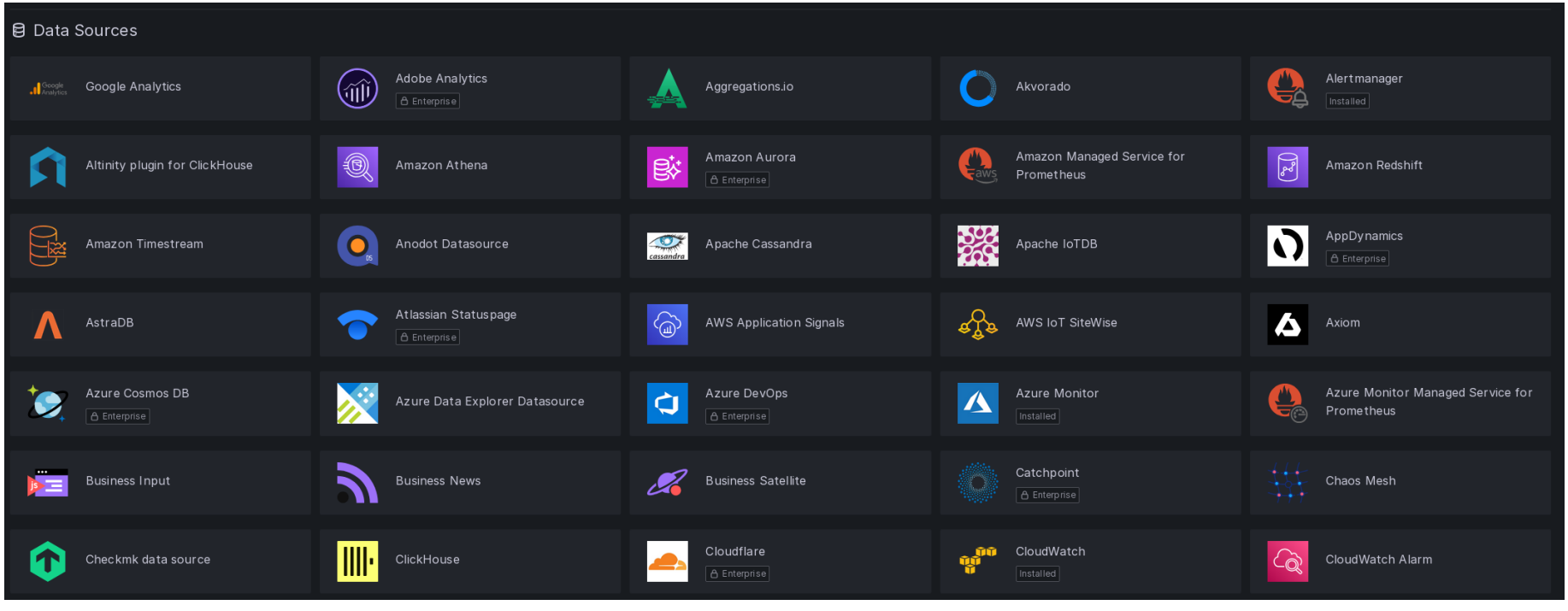
Organisé par sections [section_name]

```
# default section
instance_name = ${HOSTNAME}

[security]
admin_user = admin

[auth.google]
client_secret = 0ldS3cretKey

[plugin.grafana-image-renderer]
rendering_ignore_https_errors = true

[feature_toggles]
enable = newNavigation
```

Configuration des chemins, de la sécurité, des fonctionalités, etc.

SOURCES DE DONNÉES

Support natif de nombreuses sources de données.

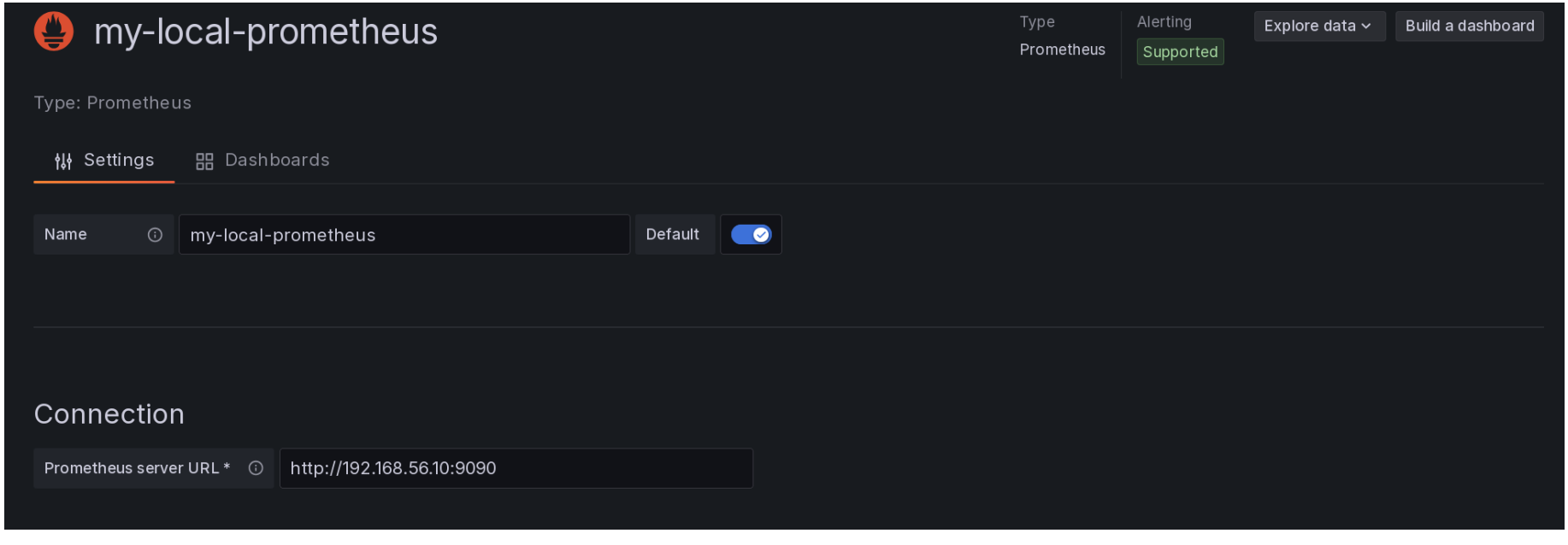



```
Alertmanager, AWS CloudWatch, Azure Monitor, Elasticsearch,
Google Cloud Monitoring, Graphite, InfluxDB, Jaeger, Loki,
Microsoft SQL Server (MSSQL), MySQL, OpenTSDB, PostgreSQL,
Prometheus, Pyroscope, Tempo, Testdata, Zipkin, ...
```

CONNEXION À PROMETHEUS

Ajouter Prometheus comme source

1. Aller dans **Configuration → Data Sources**
2. Cliquer sur **"Add data source"**
3. Choisir **Prometheus**
4. Paramétrer l'**URL** et l'authentification



CONFIGURATION PROMETHEUS

Exemple de configuration

- **URL** : `http://prometheus:9090` (ou `localhost:9090`)
- **Access** : `Server`
- **Scrape Interval** : laisser par défaut
- Cliquer sur **"Save & Test"**

✅ Vérification

- Le message **"Data source is working"** doit apparaître.
- Prometheus est maintenant connecté à **Grafana**.

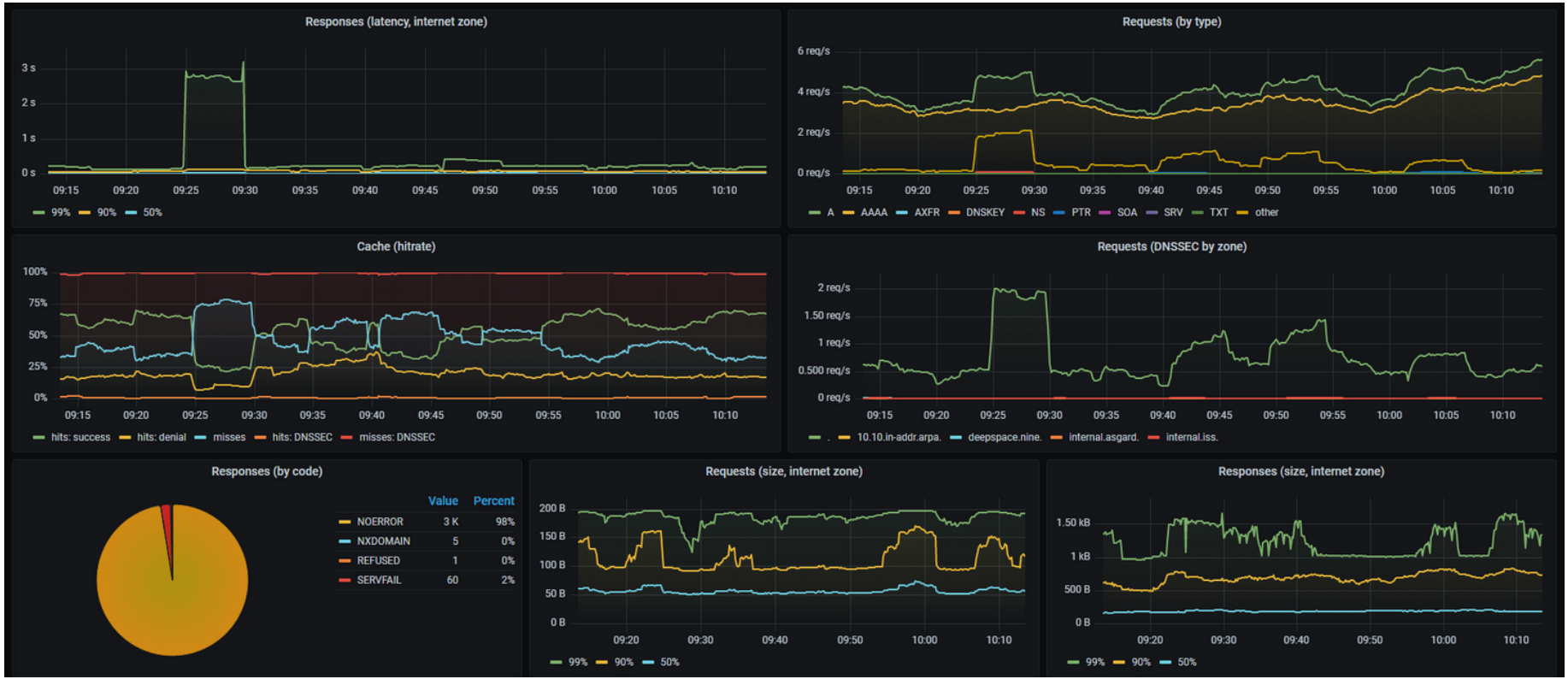
CONFIGURATION DE SOURCES AVEC LES FICHIERS

/etc/grafana/provisioning/datasources/datasources.yaml

```yaml
apiVersion: 1
datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9090

  - name: Tempo
    type: tempo
    access: proxy
    url: http://tempo:3200
```

TABLEAUX DE BORD

- Ensemble de panneaux
- Regroupés en lignes / colonnes
- Visualisations des données



NOUVEAU TABLEAU DE BORD

Étapes

1. Aller dans **Dashboards** → **New** → **New Dashboard**
2. Cliquer sur **"Add a new panel"**
3. Choisir la **source de données** (Prometheus)
4. Créer un panneau
5. Écrire une **requête PromQL**

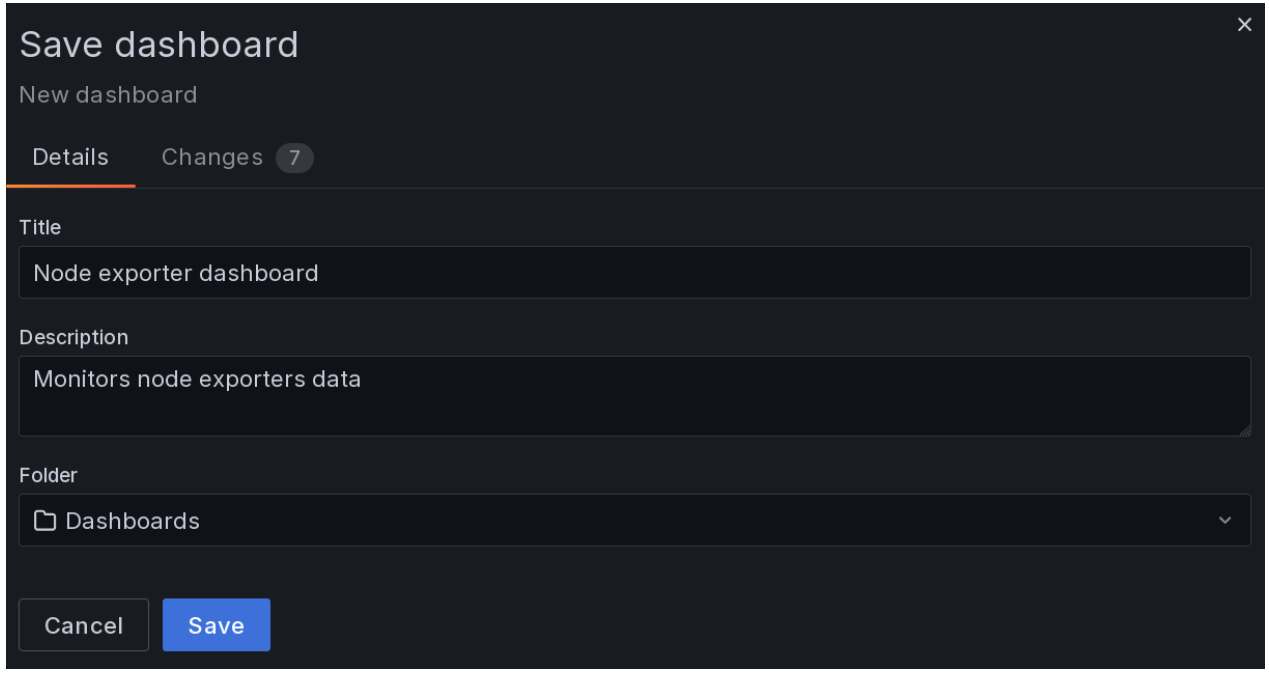
Exemple de requête

```
node_cpu_seconds_total
```

SAUVEGARDE D'UN DASHBOARD

📌 Sauvegarde

- Cliquer sur **"Save dashboard"**
- Donner un **nom** et une **description**
- **Organisation** par **dossiers**



EXPLORATION COMMUNAUTAIRE

🌍 Importer des dashboards publics

1. Aller dans **Create → Import**
2. Entrer un **ID** depuis Grafana.com/dashboards
3. Choisir une **source de données**

*Exemple : ID **1860** pour un dashboard Node Exporter*

EXPLORATION COMMUNAUTAIRE

🔗 Partager un dashboard

- Cliquer sur **l'icône de partage** (en haut à droite)
- Copier un **lien public ou privé**
- Ou exporter au **format JSON**

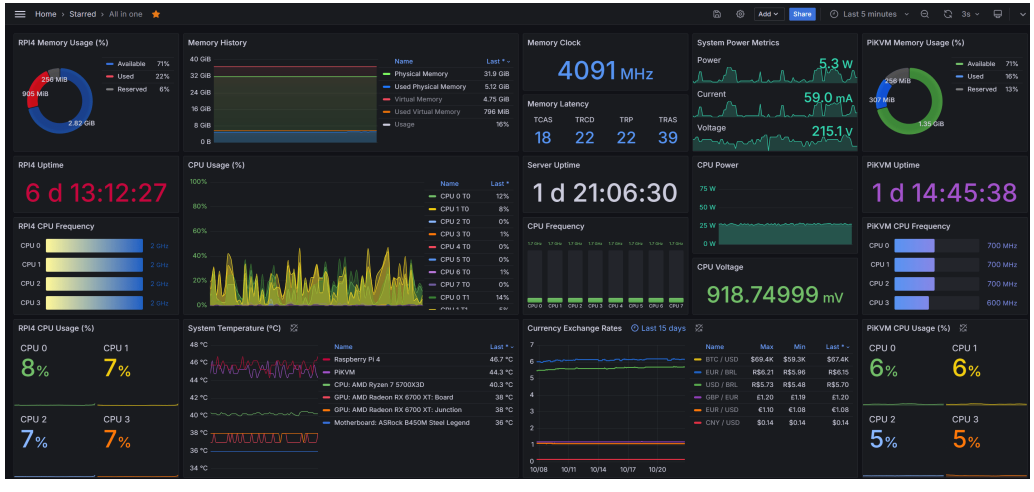
EXPLORATION COMMUNAUTAIRE

💾 Exporter / réutiliser

- Dans les **paramètres du dashboard**
- Cliquer sur **"JSON model"** pour copier ou enregistrer

LES DIFFÉRENTS PANNEAUX

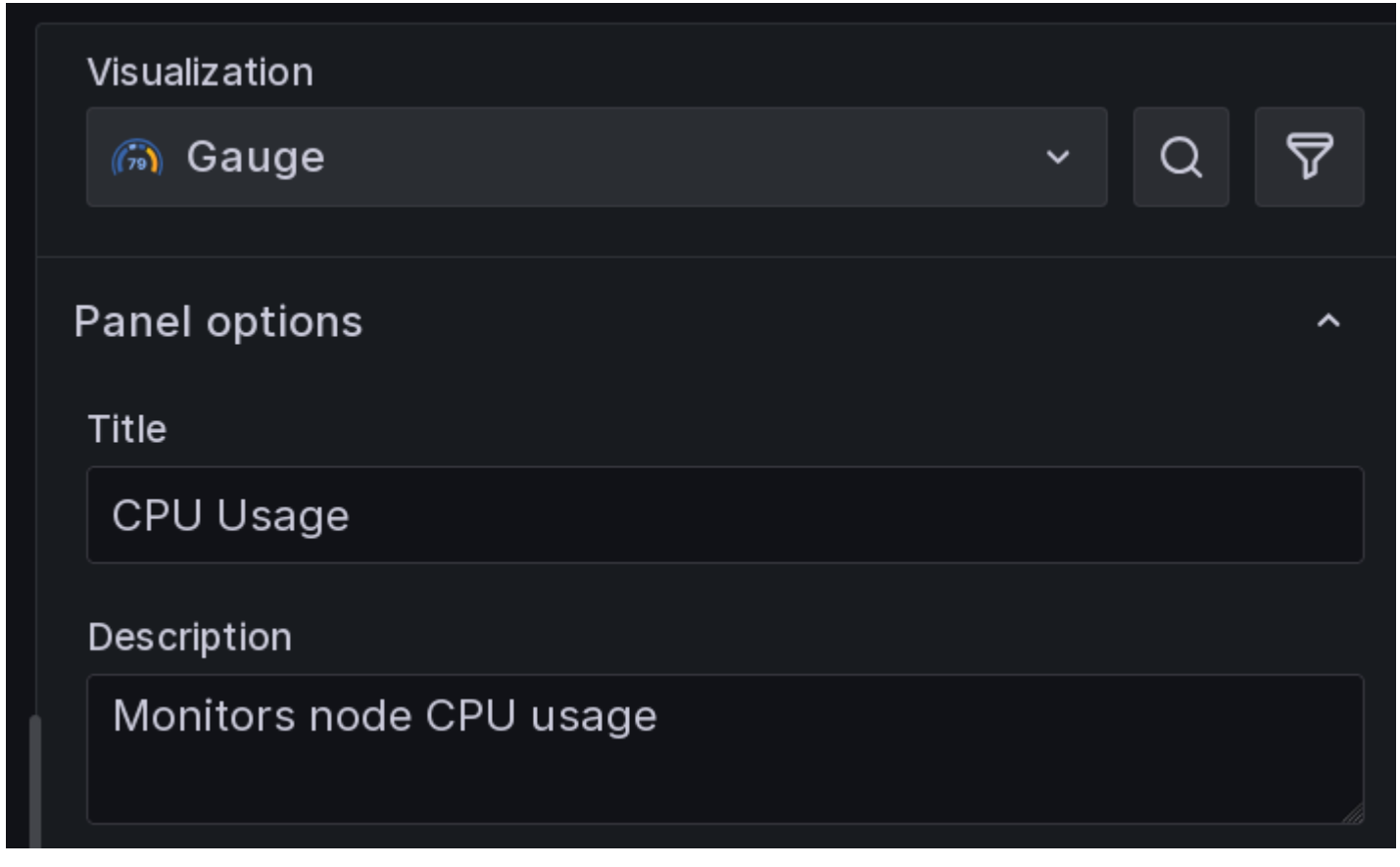
- Support natif de différents types
  - Time series, bar chart, stat, gauge, bar gauge, table, pie chart, state timeline, heatmap, status history, histogram, text, dashboard list, news (rss), annotation list, candlestick, canvas, flame graph, geomap, logs, node graph, trend (beta), chart



- Possibilité d'installer d'autres types via des plugins
  - https://grafana.com/grafana/plugins/panel-plugins/

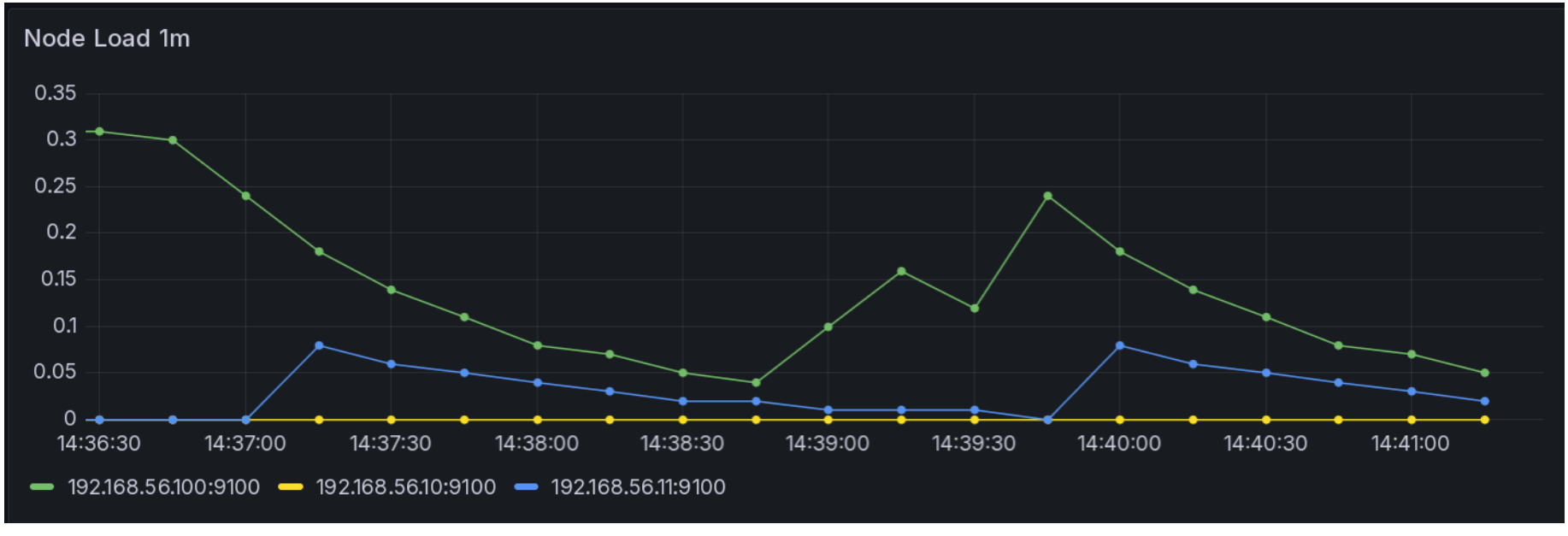
PERSONNALISER UN PANEL

- Type de panel : **Time series**, **Gauge**, **Bar Chart**, etc.
- Ajouter un **titre** et une **légende**
- Cliquer sur **"Apply"**



TIME SERIES

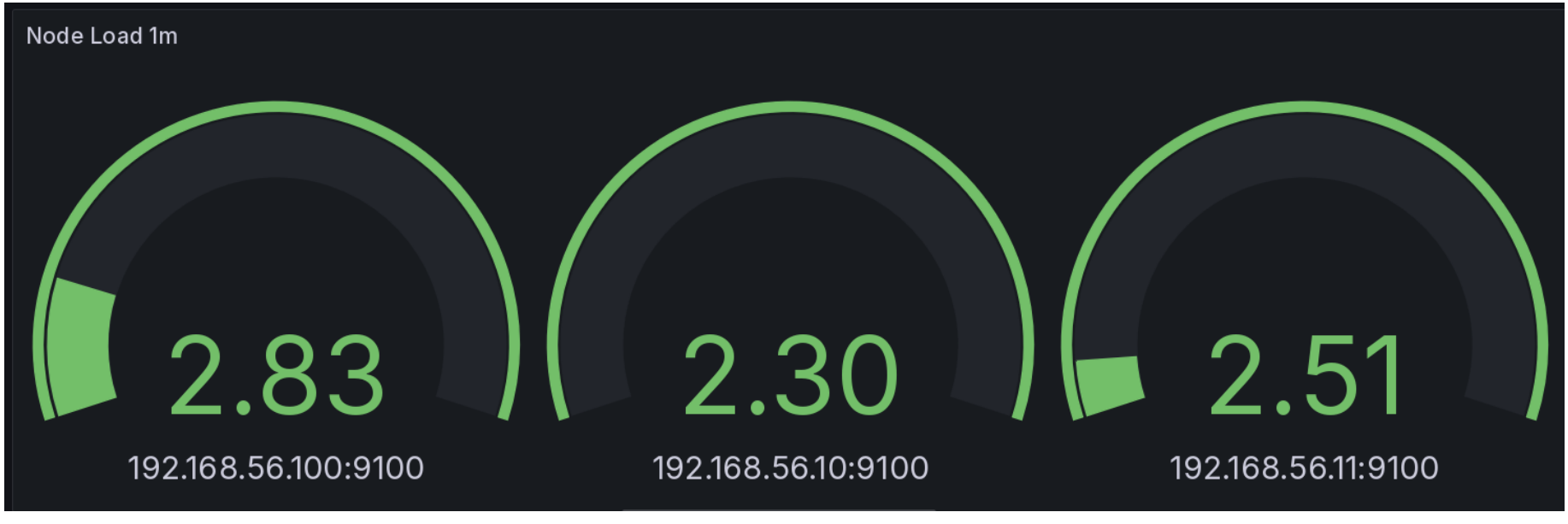
Permet d'afficher l'évolution d'une métrique (l'axe X sera toujours le temps)



Métrique utilisée (node exporter) `node_load1`

GAUGE

Permet d'afficher une jauge, les données sont aggrégées sur l'intervalle de temps choisi.



Métrique utilisée (node exporter) `node_load1`

STAT

Permet d'afficher une valeur numérique

Métrique utilisée `count(up)`

[4] REQUÊTES PROMQL

- Syntaxe et fonctionnalités de base de PromQL
- Création de requêtes PromQL avancées

REQUÊTES PROMQL

- Objectif

- Interroger les **métriques collectées** par Prometheus
- Afficher les résultats dans **Grafana** ou l'interface Prometheus
- Utiliser une **syntaxe simple et puissante**

Exemples de requêtes (avec et sans filtre)

```
http_requests_total
http_requests_total{job="my-promehteus-job"}
http_requests_total{my-custom-label="custom-value"}
http_requests_total{foo="bar",name="alice"}
```

REQUÊTES PROMQL

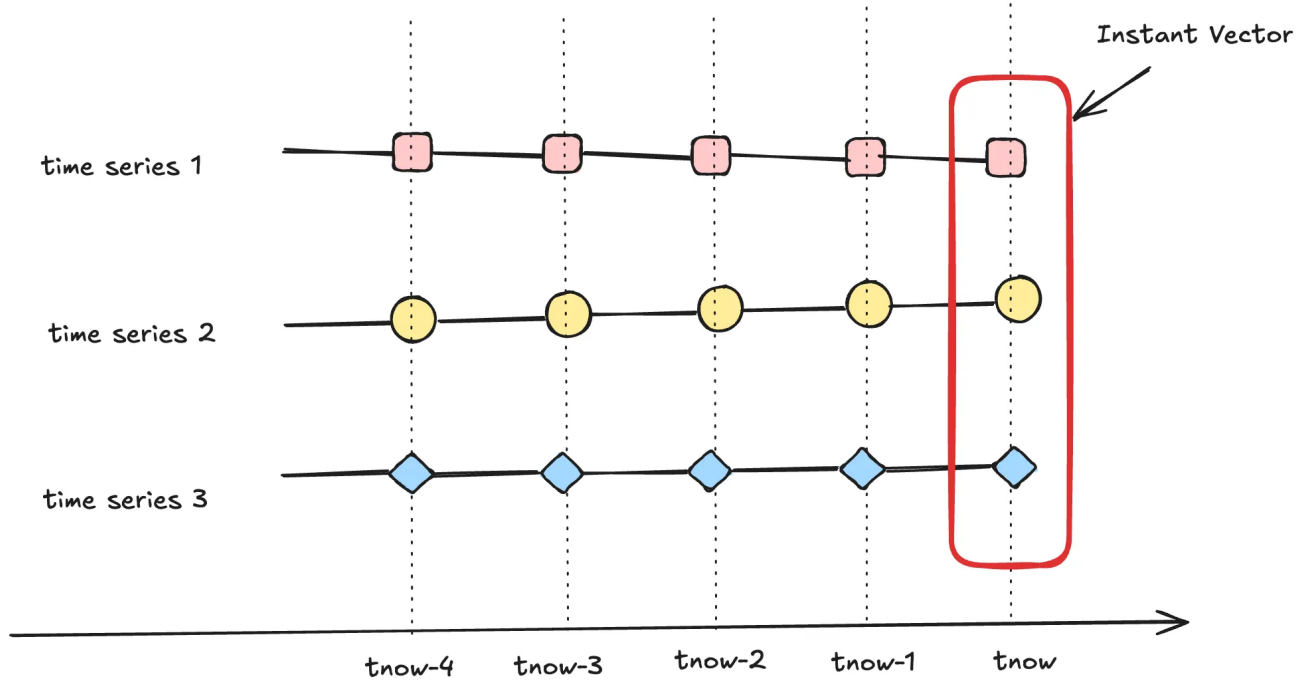
- Types de données

- **Instantané** : valeur à un **instant T** (instant vector)
- **Série temporelle** : valeurs sur une **durée**, notée avec crochets (range vector)

Exemple simple

```
# Instantanée
node_load1

# Lissage [interval]
rate(node_load1[1h])
```

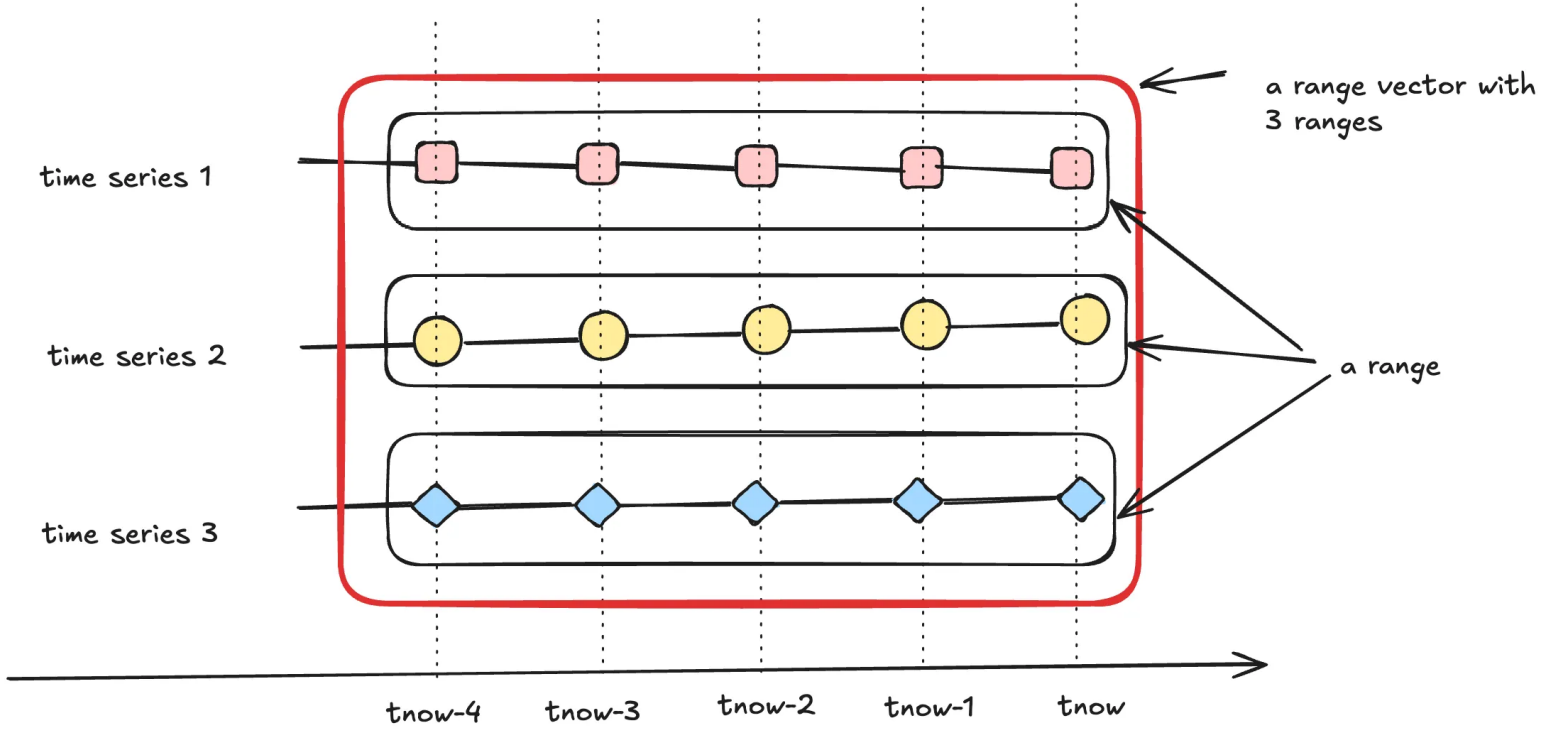
METRICS INSTANT VECTOR



PromQL :

```
node_load1
```

METRICS RANGE VECTOR



PromQL :

```
node_load1[5m]
```

SYNTAXE ET FONCTIONNALITÉS DE BASE

- Sélection d'une métrique, simplement via son nom

```
node_cpu_seconds_total
```

*Récupère les **temps CPU cumulés***

- Filtres par **labels**

```
node_cpu_seconds_total{mode="idle", instance="localhost:9100"}
```

*Filtre sur des **valeurs spécifiques***

OPÉRATEURS

- Opérateurs arithmétiques

- + (addition)
- - (soustracion)
- * (multiplication)
- / (division)
- % (modulo)
- ^ (puissance)

```
rate(http_requests_total[1m]) * 100
```

*Calcule un **taux** et le multiplie*

OPÉRATEURS

- Opérateurs de comparaison

- == (égal)
- != (différent)
- > (supérieur)
- < (inférieur)
- >= (supérieur ou égal)
- <= (inférieur ou égal)

OPÉRATEURS

- Opérateurs d'agrégation

- `sum(v)` (somme)
- `avg(v)` (moyenne)
- `min(v)` (valeur minimum)
- `max(v)` (valeur maximum)
- `bottomk(k, v)` (plus petits éléments)
- `topk(k, v)` (plus grands éléments)

OPÉRATEURS

- Opérateurs d'agrégation

- `count(v)` (nombre d'éléments dans un vecteur)
- `count_values(l, v)` (nombre d'éléments dans un vecteu
  ayant une valeur donnée)
- `stddev(v)` (déviation standard)
- `stdvar(v)` (variance standard)

FONCTIONS

- Fonctions utiles

- `rate()` : **variation par seconde**
- `avg()`, `sum()`, `min()`, `max()` : **agrégation**
- `count()` : nombre de **séries**

Exemple avec fonction

```
avg(rate(node_cpu_seconds_total{instance="192.168.56.12"}[5m]))
```

*Moyenne de l'utilisation **CPU** d'une machine*

FONCTIONS

- Liste complète

```
abs(), absent(), absent_over_time(), ceil(), changes(), clamp(),
clamp_max(), clamp_min(), day_of_month(), day_of_week(), day_of_year()
days_in_month(), delta(), deriv(), double_exponential_smoothing(),
exp(), floor(), histogram_avg(), histogram_count(), histogram_sum(),
histogram_fraction(), histogram_quantile(), histogram_stddev(),
histogram_stdvar(), hour(), idelta(), increase(), info(), irate(),
label_join(), label_replace(), ln(), log2(), log10(), minute(), month()
predict_linear(), rate(), resets(), round(), scalar(), sgn(), sort(),
sort_desc(), sort_by_label(), sort_by_label_desc(), sqrt(), time(),
timestamp(), vector(), year()
```

https://prometheus.io/docs/prometheus/latest/querying/functions/

PROMQL EXEMPLES UTILES

- Utilisation CPU groupées par instance

```
(sum by (instance)(rate(node_cpu_seconds_total{mode!="idle"}[1m])))*100
```

- Utilisation de la mémoire par instance

```
sum by (instance)(node_memory_Active_bytes/node_memory_MemTotal_bytes*100)
```

- Temps de collecte des métriques par instance

```
sum by (instance)(
    rate(node_scrape_collector_duration_seconds{job="lab-metrics"}[1m])
)
```

PROMQL EXEMPLES UTILES

- Agrégation par labels

```
sum by (instance) (rate(http_requests_total[1m]))
```

*Regroupe les requêtes par **instance***

- Comparaison entre métriques

```
rate(errors_total[5m]) / rate(requests_total[5m])
```

*Taux d'**erreurs** sur le total des requêtes*

REQUÊTES PROMQL AVANCÉES

- Expressions conditionnelles

```
node_filesystem_avail_bytes < 1024 * 1024 * 1024
```

*Détecte un **espace disque faible***

- Utilisation avec `offset` (décalage temporelle)

```
rate(cpu_usage_total[5m]) - rate(cpu_usage_total[5m] offset 1h)
```

*Compare l'usage CPU avec **il y a une heure***

REQUÊTES PROMQL AVANCÉES

- Exemple complet pour dashboard

```
100 - (
    avg by(instance)
        (rate(node_cpu_seconds_total{mode="idle"}[1m])) * 100
    )
```

Pourcentage d'utilisation CPU *par instance*

[5] ALERTING

- Configuration des alertes Prometheus
- Intégration des alertes avec des services tiers
- Gestion des alertes dans Grafana

PROMETHEUS ALERTING

🔹 Objectif

- Détecter des **anomalies** en temps réel
- Déclencher des **notifications automatiques**
- Fonctionne avec **Alertmanager**

🔹 Composants

- **Règles d'alerte** : dans Prometheus
- **Alertmanager** : gère les envois
- **Destinations** : e-mail, Slack, Teams, etc.

PROMETHEUS ALERTING

Exemple simple d'alerte

```yaml
groups:
  - name: node-alerts
    rules:
      - alert: HighCPUUsage
        # Requete PromQL
        expr: >
          avg(rate(node_cpu_seconds_total{mode!="idle"}[2m])) > 0.8
        for: 1m
        labels:
          severity: warning
        annotations:
          summary: "Utilisation CPU élevée"
```

CONFIGURATION DES ALERTES

- Étapes

1. Créer un fichier **`alerts.yml`**
2. Ajouter le fichier dans **`prometheus.yml`**

📄 `prometheus.yml`

```
rule_files:
  - "alerts.yml"
```

3. recharger Prometheus

CONFIGURATION DES ALERTES

📄 alerts.yml

```yaml
groups:
  - name: node
    rules:
      - alert: DiskSpaceLow
        expr: >
          node_filesystem_avail_bytes / node_filesystem_size_bytes
          < 0.1
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "Espace disque faible"
```

INTÉGRATION DES ALERTES AVEC DES SERVICES TIERS

- Utiliser **Alertmanager**

- Fournit un **service centralisé** pour les alertes
- Supporte **Slack**, **email**, **webhooks**, **Teams**

📄 `alertmanager.yml`

```
receivers:
  - name: 'slack-notifications'
    slack_configs:
      - channel: '#alertes'
        send_resolved: true
        api_url: 'https://hooks.slack.com/services/xxx/yyy/zzz'
route:
  receiver: 'slack-notifications'
```

ALERTMANAGER

- Lancer Alertmanager avec Docker

```
docker run -d \
  -p 9093:9093 \
  -v $(pwd)/alertmanager.yml:/etc/alertmanager/alertmanager.yml \
  prom/alertmanager
```

- Lier Prometheus à Alertmanager

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            - 'localhost:9093'
```

GESTION DES ALERTES DANS GRAFANA

- Visualiser les alertes Prometheus

- Ajouter Prometheus comme **source de données**
- Activer l'**alerting UI** dans Grafana

- Créer des alertes dans un **panel**

1. Aller dans un **dashboard**
2. Cliquer sur un **panel → Edit → Alert**
3. Activer **"Alert"** et définir la **condition**

GESTION DES ALERTES DANS GRAFANA

- Exemple d'alerte dans Grafana

- Condition : `avg() of query (A) is above 80`
- Durée : `5m`
- Sévérité : `warning`
- Notifications : Slack, Email, Webhook…

- Gestion centralisée

- Menu : **Alerting → Alert Rules**
- Suivre les statuts : **OK / Pending / Alerting**

[6] BONNES PRATIQUES

- Naming
- Dashboards
- Recording rules
- Pushgateway

BONNES PRATIQUES DE NAMING

🔹 RÈGLES GÉNÉRALES

- Utiliser des **noms clairs, explicites**
- Préférer le format : `nom_ressource_type_unité`

📌 Exemples

```
http_requests_total          ✅ OK
cpu_usage_percent            ✅ OK
disk_space_left_bytes        ✅ OK
```

❌ À éviter

```
requests                     ❌ trop vague
temp1                        ❌ non descriptif
```

BONNES PRATIQUES DE NAMING

- Suffixes recommandés

- **_total** : compteur cumulatif
- **_seconds** : durée
- **_bytes** : taille / mémoire
- **_ratio / _percent** : pourcentages

BONNES PRATIQUES POUR LES DASHBOARDS

- Simplicité

- Limiter à **1 type d'information** par panneau
- Grouper par **fonction ou service**

- Lisibilité

- Ajouter des **titres explicites**
- Utiliser des **unités cohérentes**

BONNES PRATIQUES POUR LES DASHBOARDS

- Requêtes optimisées

- Utiliser des **`recording rules`** pour les calculs lourds
- Préférer `rate()` et `avg()` à `irate()` sauf cas spécifiques

- Exemple de structure

- **CPU / RAM / Disque** : panels par serveur
- **HTTP / API / Erreurs** : panels applicatifs
- **Alertes en haut**, **métriques brutes en bas**

BONNES PRATIQUES DES RECORDING RULES

- Objectif

- **Pré-calculer** des métriques lourdes
- Réduire la **charge** sur Prometheus

📄 `recording_rules.yml`

```yaml
groups:
  - name: app_rules
    rules:
      - record: job:http_requests:rate5m
        expr: rate(http_requests_total[5m])
```

BONNES PRATIQUES DES RECORDING RULES

- 🔹 Intégration

Dans `prometheus.yml` :

```
rule_files:
  - "recording_rules.yml"
```

✅ Bonnes pratiques

- Nommer les règles comme des **métriques classiques**
- Indiquer l'**agrégation** ou **fenêtre** dans le nom ex : `job:http_latency_seconds:avg5m`

BONNES PRATIQUES AVEC PUSHGATEWAY

🔹 Objectif

- Permettre à des **jobs batch** de **pousser** des métriques
- Utilisé quand Prometheus ne peut pas faire de `scrape`

❗ Cas d'usage

- **Jobs courts**, **scripts cron**, **CI/CD**

📄 Exemple de push (bash + `curl`)

```
echo "build_duration_seconds 12.5" | curl --data-binary \
 @- http://localhost:9091/metrics/job/build/instance/ci-runner
```

BONNES PRATIQUES AVEC PUSHGATEWAY

📄 Scrape config dans `prometheus.yml`

```yaml
scrape_configs:
  - job_name: 'pushgateway'
    static_configs:
      - targets: ['localhost:9091']
```

✅ Bonnes pratiques

- Supprimer les métriques **après le push** si plus utiles
- Éviter d'en faire un **point critique** du monitoring
- Bien nommer les **jobs** et **instances** poussés

[7] PRODUCTION ET MISE À L'ÉCHELLE

- Sécurité
- Sauvegarde et restauration
- Prometheus Federation

SÉCURITÉ

- Objectifs

- Restreindre l’accès à **Prometheus**
- Protéger l’accès à l’**interface web** et aux **endpoints**

- Authentification

- Prometheus **ne gère pas l’authentification** nativement
- Utiliser un **reverse proxy** (Nginx, Traefik…)

REVERSE PROXY

Nginx + basic auth

nginx.conf

```
location / {
  auth_basic "Prometheus Auth";
  auth_basic_user_file /etc/nginx/.htpasswd;
  proxy_pass http://localhost:9090;
}
```

AUTRES POINTS DE SÉCURITÉ

🔐 HTTPS

- Mettre en place un **proxy TLS** (Nginx, Caddy, etc.)
- Exemple avec **Let's Encrypt** pour TLS automatique

🔒 Autres pratiques

- Limiter l'accès au **port 9090**
- Restreindre l'accès aux **endpoints `/metrics`** sur les exporters

SAUVEGARDE ET RESTAURATION

- Objectif

- Protéger les **données des métriques** en cas d'incident
- Restaurer une instance Prometheus rapidement

- Données à sauvegarder

- Dossier `**/prometheus/data/**` (base de données)
- Fichiers de configuration :
  - `prometheus.yml`
  - `rules.yml`
  - `alertmanager.yml`

SAUVEGARDE ET RESTAURATION

▶️ Exemple de sauvegarde

```
docker stop prometheus
tar czvf prometheus-backup.tar.gz /path/to/prometheus/data
docker start prometheus
```

*Toujours **arrêter Prometheus** avant la sauvegarde pour éviter les corruptions*

▶️ Exemple de restauration

```
docker stop prometheus
tar xzvf prometheus-backup.tar.gz -C /path/to/prometheus/
docker start prometheus
```

PROMETHEUS FEDERATION

- Objectif

- Créer une **hiérarchie Prometheus**
- **Scalabilité horizontale** : agréger des métriques de plusieurs serveurs
- Réduire la charge sur les **Prometheus locaux**

- Cas d'usage

- Environnements **multi-cluster**
- Collecte **globale + locale**
- Réplication de métriques **filtrées** vers un Prometheus central

PROMETHEUS FEDERATION

📄 prometheus.yml

```
scrape_configs:
  - job_name: 'federate'
    metrics_path: '/federate'
    params:
      'match[]':
        - '{__name__=~"job:.*"}'
    static_configs:
      - targets:
          - 'prometheus-us-east:9090'
          - 'prometheus-eu-west:9090'
```

PROMETHEUS FEDERATION

🔹 Fonctionnement

- Appelle `/federate` sur les instances cibles
- Récupère **uniquement les métriques spécifiées**
- Doit être **utilisé avec parcimonie**

✅ Bonnes pratiques

- Fédérer uniquement les **métriques agrégées**
- Ne pas importer de **séries brutes à fort cardinalité**
- Ajouter un préfixe aux métriques si besoin (`record:`)

FIN DE LA FORMATION

Merci de votre attention

DOCUMENTATIONS OFFICIELLES

- https://prometheus.io/docs
- https://grafana.com/docs

DOCDOKU

Retrouvez nous sur https://www.docdoku.com

- Twitter : https://twitter.com/docdoku
- Facebook : https://www.facebook.com/docdoku
- Linkedin : https://fr.linkedin.com/company/docdoku