

Monitoring Prometheus - Grafana

Sommaire


1. Monitoring Linux + ESXi avec Prometheus
2. Grafana, Alertes et Dashboards VMware
3. Logs, supervision complète et cas d'usage

Monitoring Linux + ESXi avec Prometheus

Monitoring Linux + ESXi avec Prometheus

Qu'est-ce que le monitoring

Le **monitoring informatique** consiste à **surveiller en temps réel** l'état de santé, les performances et le comportement d'un système, d'une infrastructure ou d'une application.

“  Objectif : détecter les problèmes **avant** qu'ils n'affectent les utilisateurs ou le business. ”

Monitoring Linux + ESXi avec Prometheus

Les trois piliers du monitoring (concepts clés)

Pilier	Contenu surveillé	Exemple d'outils
Métriques	Données chiffrées, numériques	Prometheus, Grafana
Logs	Fichiers texte contenant les événements	ELK (Elasticsearch, Logstash, Kibana), Grafana Loki
Traces	Suivi d'une requête à travers plusieurs services (ex. microservices)	Jaeger, OpenTelemetry

Monitoring Linux + ESXi avec Prometheus

Types de supervision

a. Supervision système

- CPU, RAM, disque, réseau, charge de la machine.
- Outils : Node Exporter, Telegraf.

b. Supervision applicative

- Erreurs, latence, taux de requêtes, état des services.
- Outils : Spring Actuator, Micrometer, Prometheus.

c. Supervision réseau

- Latence, disponibilité des services, ports ouverts.
- Outils : Blackbox Exporter, Nagios, Zabbix.

Monitoring Linux + ESXi avec Prometheus

Types de données collectées

Type de donnée	Exemple
Gauge	Valeur instantanée (ex. RAM libre)
Counter	Valeur qui ne fait qu'augmenter (ex. nombre de requêtes)
Histogram	Distribution de valeurs (latences)
Summary	Moyennes, percentiles, quantiles

Monitoring Linux + ESXi avec Prometheus

Pull vs Push Monitoring

Mode	Description	Exemple
Pull	Le système de monitoring interroge les sources	Prometheus
Push	Les systèmes envoient leurs données au serveur	Telegraf, StatsD

Monitoring Linux + ESXi avec Prometheus

Monitoring vs Observability

Concept	Monitoring	Observability
Définition	Surveillance des métriques connues	Capacité à comprendre un système inconnu à partir des données
Données	Collecte planifiée	Exploration libre et corrélation
But	Détection	Diagnostic

Monitoring Linux + ESXi avec Prometheus

Alerting et actions

- **Définir des seuils** sur les métriques (ex. CPU > 80%)
- **Déclencher des alertes** (email, Slack, webhook)
- **Réagir automatiquement** (ex. scaling, redémarrage)

Monitoring Linux + ESXi avec Prometheus

Alerting et actions

- Surveiller les bons indicateurs : pas trop, pas trop peu.
- Utiliser des dashboards simples et lisibles.
- Corréler métriques, logs et traces.
- Mettre à jour les alertes régulièrement.
- Tester vos alertes (simulation de panne).

Monitoring Linux + ESXi avec Prometheus

Introduction à Prometheus

- **Prometheus** est un **système de monitoring open-source** conçu pour collecter, stocker et interroger des **métriques temporelles** (time series).
- Il a été créé par SoundCloud et fait maintenant partie de la **Cloud Native Computing Foundation (CNCF)** aux côtés de Kubernetes.

Monitoring Linux + ESXi avec Prometheus

Introduction à Prometheus

Fonctionnalité	Description
Modèle Pull	Prometheus interroge (scrape) les cibles à intervalles réguliers
Stockage local	Base de données en série temporelle intégrée
PromQL	Langage de requêtage puissant pour interroger et agréger les métriques
Alertmanager intégré	Pour gérer les alertes (email, Slack, etc.)
Sans dépendance externe	Fonctionne sans base de données externe
Exporters	Modules pour exposer les métriques des systèmes, applis, bases, etc.

Monitoring Linux + ESXi avec Prometheus

Introduction à Prometheus

Architecture de Prometheus

```
[ Exporters ] ← (Node, MySQL, Blackbox, etc.)  
    ↓  
Prometheus Server  
    ↓  
[ Time Series DB ]  
    ↓  
[ PromQL + HTTP API ]  
    ↓  
[ Grafana (visualisation) ]  
    ↓  
[ Alertmanager (alertes) ]
```

Monitoring Linux + ESXi avec Prometheus

Introduction à Prometheus

Types de données gérés

Prometheus collecte des **métriques formatées en texte** :

```
# HELP http_requests_total Total number of HTTP requests
# TYPE http_requests_total counter
http_requests_total{method="GET",code="200"} 1027
http_requests_total{method="POST",code="500"} 3
```

Chaque métrique a :

- un **nom**
- des **étiquettes (labels)** pour identifier la source
- une **valeur**
- un **horodatage**

Monitoring Linux + ESXi avec Prometheus

Introduction à Prometheus

Fonctionnement basique

- Vous définissez dans `prometheus.yml` **les cibles à scraper**.
- Prometheus récupère périodiquement leurs **métriques via HTTP**.
- Vous interrogez ces données via l'**interface Web ou PromQL**.
- Vous pouvez **visualiser** les résultats via **Grafana**.
- Vous configurez des **règles d'alerte** pour être notifié.

Monitoring Linux + ESXi avec Prometheus

Introduction à Prometheus

Cas d'usage courants

Cas d'usage	Exemple
Monitoring système	Node Exporter : CPU, RAM, disque, réseau
Supervision base de données	MySQL/PostgreSQL Exporter
Disponibilité réseau	Blackbox Exporter : ping, HTTP check, DNS
Monitoring Kubernetes	kube-prometheus-stack, kube-state-metrics
Monitoring applicatif	Spring Boot avec Micrometer, Express.js, etc.

Monitoring Linux + ESXi avec Prometheus

Introduction à Prometheus

Avantages de Prometheus

- ✓ Léger et autonome
- ✓ Adapté aux environnements cloud/microservices
- ✓ Énorme écosystème d'exporters
- ✓ Requête très puissante avec PromQL
- ✓ Intégration facile avec Grafana et Alertmanager

Monitoring Linux + ESXi avec Prometheus

Étapes d'installation de Prometheus

1. Mise à jour de la VM

```
sudo apt update && sudo apt upgrade -y
```

2. Création d'un utilisateur système pour Prometheus

```
sudo useradd --no-create-home --shell /bin/false prometheus
```

3. Création des répertoires nécessaires

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

4. Téléchargement et extraction de Prometheus

```
cd /tmp  
curl -LO https://github.com/prometheus/prometheus/releases/download/v2.52.0/prometheus-2.52.0.linux-amd64.tar.gz  
tar -xzf prometheus-2.52.0.linux-amd64.tar.gz  
cd prometheus-2.52.0.linux-amd64
```

Monitoring Linux + ESXi avec Prometheus

Étapes d'installation de Prometheus

◆ 5. Déplacement des fichiers

```
sudo cp prometheus /usr/local/bin/  
sudo cp promtool /usr/local/bin/  
sudo cp -r consoles /etc/prometheus  
sudo cp -r console_libraries /etc/prometheus  
sudo cp prometheus.yml /etc/prometheus
```

◆ 6. Attribution des droits

```
sudo chown prometheus:prometheus /usr/local/bin/prometheus  
sudo chown prometheus:prometheus /usr/local/bin/promtool  
sudo chown -R prometheus:prometheus /etc/prometheus  
sudo chown -R prometheus:prometheus /var/lib/prometheus
```

Monitoring Linux + ESXi avec Prometheus

Étapes d'installation de Prometheus

◆ 7. Création du service systemd

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

➡ Contenu du fichier :

```
[Unit]
Description=Prometheus Monitoring
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
```

Monitoring Linux + ESXi avec Prometheus

Étapes d'installation de Prometheus

8. Démarrage de Prometheus

```
sudo systemctl daemon-reexec  
sudo systemctl daemon-reload  
sudo systemctl start prometheus  
sudo systemctl enable prometheus
```

9. Vérification

- Accès via navigateur :

```
http://<IP_VM>:9090
```

- Vérification du service :

```
sudo systemctl status prometheus
```

Monitoring Linux + ESXi avec Prometheus

Configuration Prometheus

Structure typique de Prometheus

```
/etc/prometheus/  
├── prometheus.yml          # ✅ Fichier principal de configuration  
├── rules/  
│   ├── alerting_rules.yml  # 🚨 Règles d'alerte  
│   └── recording_rules.yml # 📊 Règles d'enregistrement  
├── file_sd/  
│   └── targets.json        # 📄 Cibles statiques via fichiers JSON/YAML  
├── consoles/  
└── console_libraries/     # 📖 Fichiers pour interface web (console UI)  
/var/lib/prometheus/      # 📁 Librairies pour les consoles personnalisées  
                           # 🗄 Base de données locale (TSDB)
```

Monitoring Linux + ESXi avec Prometheus

Configuration Prometheus

Qu'est-ce que `prometheus.yml` ?

C'est le **fichier central** dans lequel Prometheus est configuré :

- Quelle fréquence de collecte ? (`scrape_interval`)
- Quelles cibles superviser ? (`scrape_configs`)
- Où sont les règles d'alerte ou d'enregistrement ? (`rule_files`)
- À quel Alertmanager envoyer les alertes ? (`alerting`)

Monitoring Linux + ESXi avec Prometheus

Configuration Prometheus

Structure générale

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'node_exporter'
    static_configs:
      - targets: ['192.168.1.10:9100']

rule_files:
  - "rules/alerting_rules.yml"
  - "rules/recording_rules.yml"

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

Monitoring Linux + ESXi avec Prometheus

Configuration Prometheus

Détail par section

global

Paramètres globaux appliqués à toutes les cibles (sauf si redéfinis localement).

```
global:
  scrape_interval: 15s           # Fréquence de scraping des cibles
  scrape_timeout: 10s           # Délai maximum pour une réponse
  evaluation_interval: 15s      # Fréquence d'évaluation des règles (alerting/recording)
```

Monitoring Linux + ESXi avec Prometheus

Configuration Prometheus

Détail par section

scrape_configs

Liste des **cibles à superviser** (les exporters, services ou applications).

```
scrape_configs:  
  - job_name: 'node_exporter'  
    static_configs:  
      - targets: ['192.168.1.10:9100', '192.168.1.11:9100']
```

Chaque **job** représente un groupe logique de cibles.

Tu peux aussi :

- Utiliser **labels** :

```
- targets: ['host1:9100']  
  labels:  
    env: prod
```

- Utiliser **relabel_configs** (pour modifier les labels dynamiquement)

Monitoring Linux + ESXi avec Prometheus

Configuration Prometheus

Détail par section

`rule_files`

Liste des fichiers contenant des **alerting rules** ou **recording rules** :

```
rule_files:
- "rules/alerting_rules.yml"
- "rules/recording_rules.yml"
```

📌 Ces fichiers contiennent des blocs `groups:` avec `rules:` à l'intérieur.

`alerting`

Définit **où envoyer les alertes** (vers Alertmanager).

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

🧠 Prometheus ne gère pas lui-même les notifications, il envoie les alertes à **Alertmanager**.

Monitoring Linux + ESXi avec Prometheus

Configuration Prometheus

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node_exporter'
    static_configs:
      - targets: ['192.168.1.10:9100']
        labels:
          instance: server01
      - targets: ['192.168.1.11:9100']
        labels:
          instance: server02
rule_files:
  - "rules/alerting_rules.yml"
  - "rules/recording_rules.yml"
alerting:
  alertmanagers:
    - static_configs:
        - targets: ['localhost:9093']
```

Monitoring Linux + ESXi avec Prometheus

Accès SSH entre les VMs

Étape 1 : Générer une paire de clés SSH sur la VM Prometheus

Sur la VM Prometheus :

```
ssh-keygen -t rsa -b 4096 -C "prometheus@vm" -f ~/.ssh/id_rsa
```

- Appuyez sur **Entrée** pour toutes les questions.
- Cela crée :
 - `~/.ssh/id_rsa` (clé privée)
 - `~/.ssh/id_rsa.pub` (clé publique)

Monitoring Linux + ESXi avec Prometheus

Accès SSH entre les VMs

Étape 2 : Copier la clé publique vers les autres VMs

Pour chaque VM cible :

```
ssh-copy-id -i ~/.ssh/id_rsa.pub user@IP_DE_LA_VM_CIBLE
```

Exemple :

```
ssh-copy-id -i ~/.ssh/id_rsa.pub ubuntu@192.168.1.11
```

“ Cela va ajouter la clé publique dans le fichier `~/.ssh/authorized_keys` de la VM cible. ”

Monitoring Linux + ESXi avec Prometheus

Accès SSH entre les VMs

Étape 3 : Test de la connexion

```
ssh user@192.168.1.11
```

- Vous devez pouvoir vous connecter **sans saisie de mot de passe**.

Étape 4 : (Optionnel) Modifier `sshd_config` sur les VMs cibles

Dans `/etc/ssh/sshd_config` :

```
PermitRootLogin no  
PasswordAuthentication no  
PubkeyAuthentication yes
```

Puis :

```
sudo systemctl restart ssh
```


Node Exporter

Node Exporter

Qu'est ce que Node Exporter

- **Node Exporter** est un outil développé par Prometheus qui permet de **collecter des métriques système** (CPU, mémoire, disque, réseau, etc.) sur un **système Linux** (et partiellement sur d'autres OS).
- Ces métriques sont ensuite exposées via HTTP pour que Prometheus puisse les récupérer et les stocker.

Node Exporter

Qu'est ce que Node Exporter

Fonctionnement

1. **Node Exporter** s'installe sur la machine à surveiller.
2. Il expose les métriques sur le port **9100** (par défaut), via une page HTTP accessible sur `http://<IP>:9100/metrics`.
3. Prometheus vient régulièrement **scraper** (collecter) les données exposées.
4. Ces données sont affichables dans **Grafana**, utilisables pour des **alertes**, ou analysées avec **PromQL**.

Node Exporter

Qu'est ce que Node Exporter

Exemples de métriques collectées

- `node_cpu_seconds_total` : temps CPU total par cœur.
- `node_memory_MemAvailable_bytes` : mémoire disponible.
- `node_disk_io_time_seconds_total` : temps d'I/O disque.
- `node_network_receive_bytes_total` : trafic réseau entrant.
- `node_filesystem_free_bytes` : espace disque libre.

Node Exporter

Qu'est ce que Node Exporter

Installation rapide (Linux)

```
wget https://github.com/prometheus/node_exporter/releases/latest/download/node_exporter-*.linux-amd64.tar.gz  
tar xvfz node_exporter-*.tar.gz  
cd node_exporter-*  
./node_exporter
```

Puis accessible via :

👉 `http://localhost:9100/metrics`

Node Exporter

Installation de Node Exporter sur plusieurs VMs Linux.

Installer **Node Exporter** sur chaque VM Linux pour que **Prometheus** puisse collecter :

- l'usage CPU,
- la mémoire,
- l'espace disque,
- le réseau,
- et les métriques système.

Node Exporter

Installation de Node Exporter sur plusieurs VMs Linux.

1. Connexion à la VM Linux

```
ssh user@<ip_vm>
```

2. Création d'un utilisateur système (optionnel)

```
sudo useradd --no-create-home --shell /bin/false node_exporter
```

3. Téléchargement de Node Exporter

```
cd /tmp
curl -LO https://github.com/prometheus/node_exporter/releases/download/v1.8.0/node_exporter-1.8.0.linux-amd64.tar.gz
tar -xzf node_exporter-1.8.0.linux-amd64.tar.gz
sudo cp node_exporter-1.8.0.linux-amd64/node_exporter /usr/local/bin/
```

Node Exporter

Installation de Node Exporter sur plusieurs VMs Linux.

4. Création du service `systemd`

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

➡ Contenu du fichier :

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

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

[Install]
WantedBy=default.target
```


Node Exporter

Installation de Node Exporter sur plusieurs VMs Linux.

5. Attribution des droits

```
sudo chown node_exporter:node_exporter /usr/local/bin/node_exporter
```

6. Démarrage du service

```
sudo systemctl daemon-reload  
sudo systemctl enable node_exporter  
sudo systemctl start node_exporter
```

7. Vérification

Depuis votre navigateur ou avec `curl` :

```
curl http://<ip_vm>:9100/metrics
```

✓ Vous devez voir des lignes comme :

```
# HELP node_cpu_seconds_total ...  
# TYPE node_cpu_seconds_total counter  
node_cpu_seconds_total{cpu="0",mode="user"} 1234.56
```

Node Exporter

Installation de Node Exporter sur plusieurs VMs Linux.

✚ Configuration côté Prometheus

Dans `/etc/prometheus/prometheus.yml` sur la VM Prometheus :

```
scrape_configs:  
  - job_name: 'node_exporters'  
    static_configs:  
      - targets:  
        - '192.168.1.10:9100'  
        - '192.168.1.11:9100'  
        - '192.168.1.12:9100'
```

Redémarrer Prometheus :

```
sudo systemctl restart prometheus
```

Node Exporter

Qu'est ce que les metrics

Définition

“ Une **métrique** dans Prometheus est une **valeur numérique collectée à un instant donné**, associée à un **nom** et à des **étiquettes (labels)**. Elle représente un **comportement observable du système**, par exemple : ”

- le nombre de requêtes HTTP,
- l'usage du CPU,
- l'espace disque utilisé,
- le nombre de connexions réseau.

Chaque métrique est une **série temporelle**, c'est-à-dire une suite de valeurs associées à des instants précis.

Node Exporter

Qu'est ce que les metrics

Structure d'une métrique

Une métrique Prometheus a cette forme :

```
<nom_de_la_métrique>{label1="valeur1", label2="valeur2"} valeur @timestamp
```

🔍 Exemple :

```
http_requests_total{method="GET", status="200", instance="192.168.1.10:8080"} 12345 @1686723600
```

- `http_requests_total` : nom de la métrique
- `{...}` : labels qui ajoutent du contexte
- `12345` : valeur observée
- `@1686723600` : horodatage (timestamp)

Node Exporter

Qu'est ce que les metrics

À quoi servent les métriques ?

- Créer des **graphes** dans Grafana
- Déclencher des **alertes** (via Alertmanager)
- Faire de l'analyse de performance
- Observer le comportement d'un service dans le temps

Node Exporter

Qu'est ce que les metrics

Types de métriques

Type	Description	Exemple typique
Counter	Cumul qui augmente seulement	http_requests_total
Gauge	Valeur qui peut monter et descendre	memory_usage_bytes, cpu_temp
Histogram	Découpe des valeurs en intervalles	http_request_duration_seconds_bucket
Summary	Calcule des quantiles + count + somme	rpc_duration_seconds

Node Exporter











Qu'est ce que les metrics

Règle générale

```
<contexte>_<type de donnée>_<unité éventuelle>
```

Cela suit la convention [Prometheus Naming Best Practices](#).

Node Exporter

Préfixe	Signification	Source habituelle
node_	 Données système (machine, OS)	node_exporter
http_	 Requêtes HTTP (serveur ou client)	Applications instrumentées
rpc_	 Appels RPC (Remote Procedure Call)	Services distribués / gRPC
process_	 Processus local supervisé	Prometheus lui-même ou apps
go_	 Statistiques sur la VM Go	App écrite en Go
kube_	 État de Kubernetes (Pods, etc.)	kube-state-metrics
container_	 Statistiques conteneurs	cAdvisor / kubelet
nginx_	 Métriques NGINX	nginx_exporter
mysql_	 Métriques MySQL	mysqld_exporter
redis_	 Métriques Redis	redis_exporter

Node Exporter

Qu'est ce que les metrics

Exemples concrets

node_ (machine physique ou VM)

```
node_cpu_seconds_total{mode="idle"}  
node_memory_Active_bytes  
node_network_receive_bytes_total
```

 Signifie : CPU, mémoire ou réseau **de la machine.**

Node Exporter

Qu'est ce que les metrics

Exemples concrets

`http_` (trafic HTTP)

```
http_requests_total{method="GET"}  
http_request_duration_seconds_bucket{le="0.5"}
```

 Requêtes HTTP traitées, avec détails sur la méthode et les temps de réponse.

Node Exporter

Qu'est ce que les metrics

Exemples concrets

`rpc_` (appels distants)

```
rpc_duration_seconds{quantile="0.99"}  
rpc_calls_total
```

 Appels RPC effectués par ou vers un service distribué (ex : gRPC).

Node Exporter

Qu'est ce que les metrics

Exemples concrets

`process_` (processus Prometheus ou application)

```
process_cpu_seconds_total  
process_resident_memory_bytes
```

 Utilisation CPU et mémoire **du processus lui-même.**

Node Exporter

Qu'est ce que les metrics

Exemples concrets

`go_` (runtime Go)

```
go_goroutines  
go_memstats_alloc_bytes
```

 Informations sur la mémoire et les goroutines **du runtime Go.**

Node Exporter

PromQL

Introduction à PromQL

PromQL est le langage utilisé par Prometheus pour **interroger, filtrer, agréger et transformer** les séries temporelles de métriques.

Il permet :

- d'afficher les valeurs actuelles,
- de calculer des taux, des moyennes, des percentiles,
- de grouper les métriques par labels,
- de déclencher des alertes (avec Alertmanager),
- d'afficher des graphiques dans Grafana.

Node Exporter

PromQL

1. Structure de base

```
<metric_name>{<label_filters>} <operator> <expression>
```

Exemples :

```
http_requests_total  
http_requests_total{method="GET", status="200"}
```

Node Exporter

PromQL

3. Fonctions de base

3.1 — `rate()` : taux d'évolution (par seconde)

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

“ Nombre moyen de requêtes HTTP par seconde sur les 1 dernières minutes. ”

3.2 — `sum()` : somme

```
sum(rate(http_requests_total[5m]))
```

“ Total des requêtes HTTP par seconde sur toutes les instances. ”

Node Exporter

PromQL

3. Fonctions de base

3.3 – `avg()`, `min()`, `max()`, `count()`

```
avg(node_memory_Active_bytes)
```

“ Moyenne de la mémoire utilisée sur toutes les machines. ”

3.4 – `irate()` : **taux instantané**

```
irate(node_network_receive_bytes_total[1m])
```

“ Taux immédiat (plus sensible aux pics que `rate()`). ”

3.5 – `increase()` : **variation sur une période**

```
increase(http_requests_total[1h])
```

“ Nombre de requêtes HTTP supplémentaires sur la dernière heure. ”

Node Exporter

PromQL

3. Fonctions de base

3.3 – `avg()`, `min()`, `max()`, `count()`

```
avg(node_memory_Active_bytes)
```

“ Moyenne de la mémoire utilisée sur toutes les machines. ”

3.4 – `irate()` : **taux instantané**

```
irate(node_network_receive_bytes_total[1m])
```

“ Taux immédiat (plus sensible aux pics que `rate()`). ”

3.5 – `increase()` : **variation sur une période**

```
increase(http_requests_total[1h])
```

“ Nombre de requêtes HTTP supplémentaires sur la dernière heure. ”

Node Exporter

PromQL

3. Fonctions de base

3.6 — `histogram_quantile()`

```
histogram_quantile(0.95, sum(rate(http_request_duration_seconds_bucket[5m])) by (le))
```

“ Donne le **95e percentile** du temps de réponse HTTP.

”

Node Exporter

PromQL

4. Opérateurs PromQL

Opérateurs arithmétiques

```
node_memory_MemFree_bytes / node_memory_MemTotal_bytes
```

“ Pourcentage de mémoire libre. ”

Opérateurs logiques

```
up == 0
```

“ Filtre les services "down". ”

Opérateurs de jointure `on()` et `ignoring()`

```
rate(http_requests_total[1m]) / on(instance) rate(http_errors_total[1m])
```

“ Taux d’erreur par instance. ”

Node Exporter

PromQL

5. Agrégations avec `by()` ou `without()`

Exemple :

```
sum(rate(http_requests_total[5m])) by (job)
```

“ Somme du taux de requêtes HTTP groupée **par job**. ”

Node Exporter

PromQL

6. ⚠ Exemples utiles

Services qui ne répondent plus :

```
up == 0
```

CPU utilisé par core :

```
rate(node_cpu_seconds_total{mode="user"}[5m])
```

RAM utilisée en pourcentage :

```
100 * (1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes))
```

Répartition des requêtes par statut HTTP :

```
sum(rate(http_requests_total[5m])) by (status)
```

Mémoire utilisée par instance :

```
node_memory_Active_bytes{job="node_exporter"}
```

Nombre de requêtes dans le dernier quart d'heure :

```
increase(http_requests_total[15m])
```

Node Exporter

PromQL

6. ⚠ Exemples utiles

Services qui ne répondent plus :

```
up == 0
```

CPU utilisé par core :

```
rate(node_cpu_seconds_total{mode="user"}[5m])
```

RAM utilisée en pourcentage :

```
100 * (1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes))
```

Répartition des requêtes par statut HTTP :

```
sum(rate(http_requests_total[5m])) by (status)
```

Mémoire utilisée par instance :

```
node_memory_Active_bytes{job="node_exporter"}
```

Node Exporter

Visualisation des métriques système (CPU, RAM, disque)

1. Métriques disponibles avec Node Exporter

CPU

- `node_cpu_seconds_total`
- `rate(node_cpu_seconds_total{mode!="idle"}[5m])` (utilisation active)

Mémoire

- `node_memory_MemTotal_bytes`
- `node_memory_MemAvailable_bytes`
- Utilisation :

```
1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)
```

Disque

- `node_filesystem_size_bytes`
- `node_filesystem_free_bytes`

```
(node_filesystem_size_bytes - node_filesystem_free_bytes) / node_filesystem_size_bytes
```


Node Exporter

Visualisation des métriques système (CPU, RAM, disque)

2. Visualisation dans Prometheus UI (temporaire)

1. Allez sur :

```
http://<ip_prometheus>:9090
```

2. Tapez une requête comme :

```
rate(node_cpu_seconds_total{mode="user"}[1m])
```

3. Cliquez sur "Execute" → Vue en tableau ou graphique brut.

💡 Prometheus UI est utile pour **tester des requêtes**, mais **pas pour faire des dashboards**.

Node Exporter

Visualisation des métriques système (CPU, RAM, disque)

3. Création d'un dashboard Grafana (recommandé)

◆ Étape 1 : Se connecter à Grafana

```
http://<ip_grafana>:3000
```

→ Login par défaut : `admin / admin`

◆ Étape 2 : Ajouter une source de données

1. Aller dans "**Configuration**" > "**Data sources**"
2. Cliquer sur "**Add data source**"
3. Choisir **Prometheus**
4. Entrer l'URL :

```
http://<ip_prometheus>:9090
```

5. **Save & Test**

Node Exporter

Visualisation des métriques système (CPU, RAM, disque)

3. Création d'un dashboard Grafana (recommandé)

♦ Étape 3 : Créer un dashboard

1. Aller dans **"Dashboards" > "New" > "New Panel"**

2. Exemple : utilisation CPU

```
100 - (avg by(instance)(rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
```

3. Titre : CPU Usage %

4. Autres panels utiles :

- **Mémoire :**

```
(1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)) * 100
```

- **Disque (par mount point) :**

```
100 * (node_filesystem_size_bytes{fstype!~"tmpfs|aufs|overlay"} - node_filesystem_free_bytes{fstype!~"tmpfs|aufs|overlay"}) / node_filesystem_size_bytes{fstype!~"tmpfs|aufs|overlay"}
```

- **Uptime :**

```
node_time_seconds - node_boot_time_seconds
```

Node Exporter

Visualisation des métriques système (CPU, RAM, disque)

4. (Optionnel) Utiliser un dashboard préconstruit

Vous pouvez aussi importer un dashboard tout fait :

1. Aller dans Grafana > Dashboards > Import
2. Coller l'ID du dashboard communautaire :
1860 (Node Exporter Full - très complet)
3. Sélectionner votre source Prometheus

Grafana, Alertes et Dashboards VMware

