

---

# Supervision, Monitoring et Métrologie — SNMP, Prometheus & Grafana

Guide complet de la supervision réseau : concepts fondamentaux, protocole SNMP, MIB, sécurité, outils du marché, déploiement de Prometheus + Grafana sur Docker, agents Windows/Linux/macOS, seuils, alertes et dashboards graphiques.

**90 min de lecture** **Niveau Intermédiaire**

---

Document généré le 27/06/2026 à 21h39 · [nouv.fr/wiki/supervision-monitoring-metrologie-snmp-prometheus-grafana](https://nouv.fr/wiki/supervision-monitoring-metrologie-snmp-prometheus-grafana)

# Sommaire

61 section(s) · 90 min de lecture

## 1. Concepts fondamentaux

- ↳ Supervision, Monitoring, Métrologie — quelle différence ?
- ↳ Pourquoi superviser ?
- ↳ Modèle de supervision en couches

## 2. SNMP — Simple Network Management Protocol

- ↳ Rôle et principe
- ↳ Les 3 versions de SNMP
- ↳ Les différentes trames SNMP
- ↳ Ports SNMP
- ↳ Community strings (SNMPv1/v2c)

## 3. MIB — Management Information Base

- ↳ Principe
- ↳ Structure de l'arbre MIB
- ↳ OIDs courants
- ↳ Types de MIB

## 4. Sécurité et SNMP

- ↳ Risques de SNMPv1/v2c
- ↳ Bonnes pratiques
- ↳ SNMPv3 — Niveaux de sécurité
- ↳ Configuration SNMPv3 (exemple Cisco IOS)

## 5. Outils de supervision du marché

- ↳ Comparatif des solutions
- ↳ Prometheus + Grafana — pourquoi ce choix ?

## 6. Architecture Prometheus + Grafana

## 7. Installation — Docker Compose

- ↳ Prérequis
- ↳ Structure des fichiers
- ↳ docker-compose.yml
- ↳ prometheus/prometheus.yml

↳ Lancer la stack

## 8. Installation des agents

↳ Agent Linux / macOS — `node_exporter`

↳ Agent Windows — `windows_exporter`

## 9. Surveillance des ressources

↳ Métriques CPU

↳ Métriques RAM

↳ Métriques disque

↳ Métriques réseau

## 10. Surveillance des services réseau

↳ Blackbox Exporter — sonder les services

↳ Requêtes de disponibilité

## 11. Équipements réseau — SNMP Exporter

↳ Configuration `snmp.yml`

↳ Métriques SNMP dans Prometheus

## 12. Seuils et alertes

↳ Fichier de règles d'alertes

↳ Configuration Alertmanager — envoi d'alertes par email (Mailjet)

## 13. Plugins et exporters supplémentaires

↳ Exporters courants

↳ Ajouter cAdvisor pour Docker

## 14. Dashboards Grafana

↳ Connecter Alertmanager à Grafana

↳ Provisionner la source de données Prometheus automatiquement

↳ Importer des dashboards communautaires

↳ Exemple de panel PromQL dans Grafana

↳ Alertes Grafana (en complément d'Alertmanager)

## 15. Supervision des équipements réseau (switches, routeurs)

↳ Activer SNMP sur un switch Cisco (exemple)

↳ Tester SNMP depuis le serveur de supervision

↳ Métriques SNMP dans Grafana



# 1. Concepts fondamentaux

---

## Supervision, Monitoring, Métrologie — quelle différence ?

Concept	Définition	Exemple
<b>Supervision</b>	Vue globale de l'état d'un SI : services UP/DOWN, alertes critiques	"Le serveur web est tombé à 14h32"
<b>Monitoring</b>	Surveillance continue de métriques en temps réel	CPU à 95 % depuis 5 min
<b>Métrologie</b>	Collecte et analyse historique de métriques pour tendances et capacité	"La RAM augmente de 2 % par mois"

Ces trois niveaux sont complémentaires :

```
Métrologie → données historiques → capacité planning
Monitoring → temps réel → détection anomalie
Supervision → vision globale → prise de décision
```

📄 Copier

## Pourquoi superviser ?

- **Disponibilité** : détecter les pannes avant les utilisateurs
- **Performance** : identifier les goulets d'étranglement
- **Sécurité** : repérer des comportements anormaux
- **Conformité** : prouver les SLA et niveaux de service
- **Capacité** : anticiper les besoins en ressources

## Modèle de supervision en couches

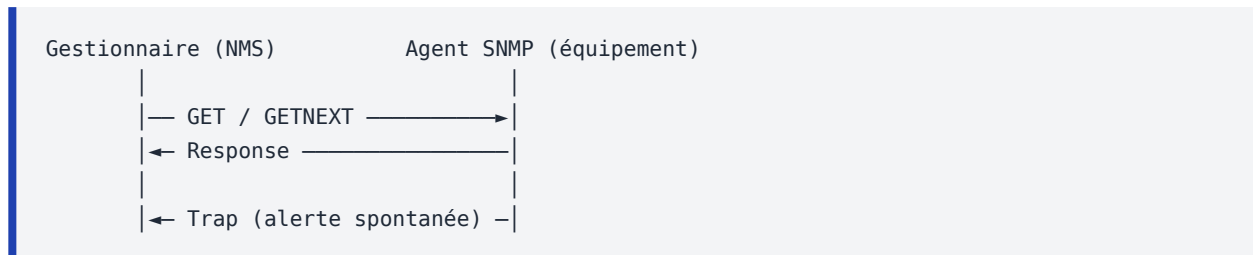
Couche	Éléments supervisés	Métriques typiques
<b>Matériel</b>	Serveurs, switches, routeurs	CPU, RAM, température, interfaces
<b>Réseau</b>	Liens, VLANs, WAN	Bande passante, latence, pertes
<b>Services</b>	HTTP, DNS, SMTP, BDD	Disponibilité, temps de réponse
<b>Applicatif</b>	Apps métier, APIs	Erreurs, transactions par seconde
<b>Utilisateur</b>	Expérience end-user	Temps de chargement, satisfaction

---

## 2. SNMP — Simple Network Management Protocol

### Rôle et principe

**SNMP** est le protocole standard pour superviser et gérer les équipements réseau (routeurs, switches, serveurs, imprimantes...). Il fonctionne en mode **client/serveur** :



📄 Copier

Composant	Rôle
<b>Manager (NMS)</b>	Logiciel de supervision qui interroge les agents (ex: Zabbix, PRTG)
<b>Agent</b>	Processus sur l'équipement supervisé qui répond aux requêtes
<b>MIB</b>	Base de données décrivant les variables accessibles via SNMP
<b>OID</b>	Identifiant unique d'une variable dans la MIB

### Les 3 versions de SNMP

Version	Authentification	Chiffrement	Usage recommandé
<b>SNMPv1</b>	Community string (texte clair)	Aucun	Obsolète, à éviter
<b>SNMPv2c</b>	Community string (texte clair)	Aucun	Encore répandu mais non sécurisé
<b>SNMPv3</b>	Utilisateur + mot de passe	AES/DES	Recommandé en production

### Les différentes trames SNMP

#### GET — Lecture d'une valeur

Le manager demande la valeur d'un OID précis.



📄 Copier

#### GETNEXT — Parcours de la MIB

Récupère l'OID suivant dans l'arbre MIB (utilisé pour explorer).

```
Manager — GETNEXT (OID: 1.3.6.1.2.1.1) → Agent
Manager ← Response (OID suivant + valeur) — Agent
```

📄 Copier

## GETBULK (SNMPv2/v3) — Récupération en masse

Optimise les requêtes en récupérant plusieurs OIDs en une seule trame.

## SET — Écriture d'une valeur

Le manager modifie un paramètre sur l'équipement (ex: redémarrage d'une interface).

```
Manager — SET (OID: ifAdminStatus = down) → Agent
Manager ← Response (OK) — Agent
```

📄 Copier

## TRAP — Notification spontanée

L'agent envoie une alerte **sans être interrogé** (ex: lien réseau tombé).

```
Agent — TRAP (linkDown, interface Gi0/1) → Manager
```

📄 Copier

## INFORM (SNMPv2/v3) — Trap avec accusé de réception

Comme un Trap mais l'agent attend une confirmation du manager.

```
Agent — INFORM → Manager
Agent ← Response (ACK) — Manager
```

📄 Copier

## Ports SNMP

Port	Protocole	Usage
161/UDP	SNMP	Requêtes GET/SET du manager vers l'agent
162/UDP	SNMP Trap	Alertes de l'agent vers le manager

## Community strings (SNMPv1/v2c)

Les community strings sont des **mots de passe en clair** qui contrôlent l'accès :

Community	Droit	Valeur par défaut (à changer !)
public	Lecture seule	Par défaut sur la plupart des équipements
private	Lecture/Écriture	Par défaut — très dangereux

## 3. MIB — Management Information Base

### Principe

La **MIB** est un annuaire hiérarchique (arbre) qui décrit toutes les variables accessibles via SNMP sur un équipement. Chaque variable est identifiée par un **OID** (Object Identifier).

### Structure de l'arbre MIB

```
. (root)
├── 1 (iso)
│   ├── 3 (org)
│   │   ├── 6 (dod)
│   │   │   ├── 1 (internet)
│   │   │   │   ├── 2 (mgmt)
│   │   │   │   │   ├── 1 (mib-2)
│   │   │   │   │   │   ├── 1 (system) ← Infos système
│   │   │   │   │   │   ├── 2 (interfaces) ← Interfaces réseau
│   │   │   │   │   │   ├── 4 (ip) ← Statistiques IP
│   │   │   │   │   │   ├── 6 (tcp) ← Statistiques TCP
│   │   │   │   │   │   └── 11 (snmp) ← Stats SNMP
│   │   │   │   └── 4 (private)
│   │   │   │       ├── 1 (enterprises)
│   │   │   │       │   ├── 9 (Cisco)
│   │   │   │       │   ├── 311 (Microsoft)
│   │   │   │       └── 2636 (Juniper)
```

📄 Copier

### OIDs courants

OID	Nom	Description
1.3.6.1.2.1.1.1.0	sysDescr	Description du système
1.3.6.1.2.1.1.3.0	sysUpTime	Durée de fonctionnement
1.3.6.1.2.1.1.5.0	sysName	Nom de l'équipement
1.3.6.1.2.1.2.2.1.10	ifInOctets	Octets reçus sur une interface
1.3.6.1.2.1.2.2.1.16	ifOutOctets	Octets émis sur une interface
1.3.6.1.2.1.25.3.3.1.2	hrProcessorLoad	Charge CPU (Host Resources MIB)

### Types de MIB

Type	Description	Exemple
<b>MIB-II (RFC 1213)</b>	MIB standard pour tout équipement IP	Interfaces, IP, TCP, UDP
<b>MIB propriétaire</b>	Fournie par le fabricant	Cisco IOS MIB, HP ProCurve MIB
<b>MIB d'extension</b>	Ajoutée par des logiciels tiers	Net-SNMP, Windows SNMP Extension

Les fichiers MIB sont au format `.mib` ou `.txt` et peuvent être chargés dans les outils de supervision pour avoir des noms lisibles plutôt que des OIDs numériques.

## 4. Sécurité et SNMP

### Risques de SNMPv1/v2c

Risque	Description
<b>Community string en clair</b>	Capturée par un simple sniff réseau (Wireshark)
<b>Accès en écriture (SET)</b>	Peut modifier la configuration d'un équipement
<b>Énumération</b>	Un attaquant peut parcourir toute la MIB avec <code>snmpwalk</code>
<b>Amplification DDoS</b>	SNMP UDP peut être utilisé en réflexion pour des attaques

### Bonnes pratiques

- Utiliser **SNMPv3** exclusivement en production
- Restreindre l'accès SNMP par **ACL réseau** (liste d'IPs autorisées)
- Désactiver SNMP sur les équipements qui n'en ont pas besoin
- **Changer** les community strings public et private
- Utiliser des community strings **complexes** si SNMPv2c obligatoire
- Filtrer les ports 161 et 162 UDP au pare-feu (accès uniquement depuis le serveur de supervision)

### SNMPv3 — Niveaux de sécurité

Niveau	Authentification	Chiffrement	Usage
noAuthNoPriv	Non	Non	Tests uniquement
authNoPriv	MD5 ou SHA	Non	Authentification sans chiffrement
authPriv	MD5 ou SHA	DES ou AES	<b>Recommandé en production</b>

### Configuration SNMPv3 (exemple Cisco IOS)

```
! Créer un groupe SNMP avec accès lecture seule
snmp-server group SUPERVISION v3 priv read iso
```

```
! Créer un utilisateur
snmp-server user admin-snmp SUPERVISION v3 auth sha MonMotDePasse priv aes 128
MaCleChiffrement
```

```
! Restreindre aux IPs autorisées
snmp-server community public R0 10
ip access-list standard 10
permit 192.168.1.50
```

📄 Copier

## 5. Outils de supervision du marché

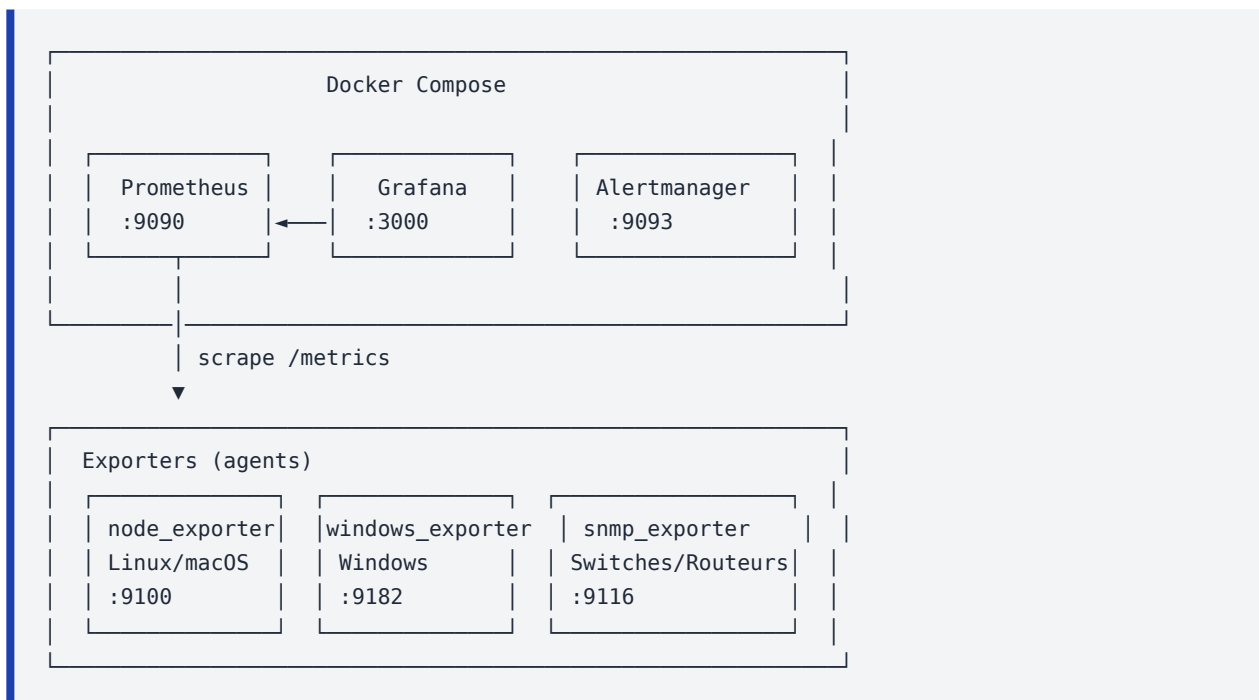
### Comparatif des solutions

Outil	Type	Licence	Points forts	Limite
<b>Nagios / Icinga</b>	Supervision active	Open source	Très mature, nombreux plugins	Interface vieillissante
<b>Zabbix</b>	Supervision complète	Open source	SNMP natif, dashboards, alertes	Configuration complexe
<b>PRTG</b>	Supervision complète	Commercial	Interface intuitive, tout-en-un	Payant (licences par sonde)
<b>Prometheus + Grafana</b>	Métrologie / Monitoring	Open source	Cloud-native, puissant, extensible	Pas de supervision active native
<b>Datadog</b>	SaaS monitoring	Commercial	SaaS clé en main, IA intégrée	Coût élevé à l'échelle
<b>Netdata</b>	Monitoring temps réel	Open source	Léger, installation rapide	Moins adapté long terme
<b>LibreNMS</b>	Supervision réseau	Open source	Auto-découverte SNMP, focus réseau	Moins adapté serveurs
<b>Checkmk</b>	Supervision complète	Open source / Commercial	Auto-configuration, plugins riches	Courbe d'apprentissage

### Prometheus + Grafana — pourquoi ce choix ?

- **Cloud-native** : conçu pour les environnements conteneurisés
- **Pull model** : Prometheus scrape les métriques (vs push en SNMP)
- **PromQL** : langage de requête puissant pour les métriques
- **Grafana** : dashboards riches, alertes, compatible de nombreuses sources
- **Ecosystem riche** : des centaines d'exporters disponibles
- **Gratuit et open source**

## 6. Architecture Prometheus + Grafana



📄 Copier

## 7. Installation — Docker Compose

Le repo GitHub de la stack est disponible ici :

<https://github.com/Nouvy/prometheus-grafana-docker>

### Prérequis

- Docker + Docker Compose installés
- Ports libres : 3000 (Grafana), 9090 (Prometheus), 9093 (Alertmanager)

### Structure des fichiers

```
supervision/  
├─ docker-compose.yml  
├─ prometheus/  
│   └─ prometheus.yml      ← Configuration Prometheus  
│       └─ rules/  
│           └─ alerts.yml  ← Règles d'alertes  
└─ grafana/  
    └─ provisioning/  
        ├─ datasources/  
        │   └─ prometheus.yml  
        └─ dashboards/  
            └─ dashboards.yml
```

📄 Copier

# docker-compose.yml

```
version: '3.8'

networks:
  supervision:
    driver: bridge

volumes:
  prometheus_data:
  grafana_data:

services:

# — Prometheus —————
prometheus:
  image: prom/prometheus:latest
  container_name: prometheus
  restart: unless-stopped
  networks:
    - supervision
  ports:
    - "9090:9090"
  volumes:
    - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
    - ./prometheus/rules:/etc/prometheus/rules
    - prometheus_data:/prometheus
  command:
    - "--config.file=/etc/prometheus/prometheus.yml"
    - "--storage.tsdb.path=/prometheus"
    - "--storage.tsdb.retention.time=30d"
    - "--web.enable-lifecycle"

# — Grafana —————
grafana:
  image: grafana/grafana:latest
  container_name: grafana
  restart: unless-stopped
  networks:
    - supervision
  ports:
    - "3000:3000"
  volumes:
    - grafana_data:/var/lib/grafana
    - ./grafana/provisioning:/etc/grafana/provisioning
  environment:
    - GF_SECURITY_ADMIN_USER=admin
    - GF_SECURITY_ADMIN_PASSWORD=admin123
    - GF_USERS_ALLOW_SIGN_UP=false
  depends_on:
    - prometheus

# — Alertmanager —————
alertmanager:
  image: prom/alertmanager:latest
  container_name: alertmanager
  restart: unless-stopped
  networks:
    - supervision
  ports:
    - "9093:9093"
  volumes:
    - ./prometheus/alertmanager.yml:/etc/alertmanager/alertmanager.yml

# — SNMP Exporter (switches, routeurs) —————
```

```
snmp-exporter:  
  image: prom/snmp-exporter:latest  
  container_name: snmp-exporter  
  restart: unless-stopped  
  networks:  
    - supervision  
  ports:  
    - "9116:9116"  
  volumes:  
    - ./snmp-exporter/snmp.yml:/etc/snmp_exporter/snmp.yml
```

📄 Copier

**prometheus/prometheus.yml**

```
global:
  scrape_interval:     15s # Fréquence de collecte
  evaluation_interval: 15s # Fréquence d'évaluation des règles
  scrape_timeout:      10s

# Règles d'alertes
rule_files:
- "rules/alerts.yml"

# Alertmanager
alerting:
  alertmanagers:
  - static_configs:
    - targets: ["alertmanager:9093"]

scrape_configs:

# Prometheus lui-même
- job_name: "prometheus"
  static_configs:
  - targets: ["localhost:9090"]

# Serveurs Linux / macOS (node_exporter)
- job_name: "linux"
  static_configs:
  - targets:
    - "192.168.1.10:9100" # SRV-NOUVY
    - "192.168.1.30:9100" # SRV-WSUS
  labels:
    env: "production"
    os: "linux"

# Postes et serveurs Windows (windows_exporter)
- job_name: "windows"
  static_configs:
  - targets:
    - "192.168.1.20:9182" # PC-WIN01
    - "192.168.1.21:9182" # PC-WIN02
  labels:
    env: "production"
    os: "windows"

# Équipements réseau via SNMP
- job_name: "snmp"
  static_configs:
  - targets:
    - "192.168.1.1" # Routeur
    - "192.168.1.2" # Switch core
  metrics_path: /snmp
  params:
    module: [if_mib]
  relabel_configs:
  - source_labels: [__address__]
    target_label: __param_target
  - source_labels: [__param_target]
    target_label: instance
  - target_label: __address__
    replacement: snmp-exporter:9116
```

## Lancer la stack

```
mkdir -p supervision/prometheus/rules supervision/grafana/provisioning/datasources
cd supervision
docker compose up -d

# Vérifier que les conteneurs tournent
docker compose ps

# Logs Prometheus
docker compose logs -f prometheus
```

📄 Copier

Accès :

- **Prometheus** : <http://localhost:9090>
- **Grafana** : <http://localhost:3000> (admin / admin123)
- **Alertmanager** : <http://localhost:9093>

---

## 8. Installation des agents

---

### Agent Linux / macOS — node\_exporter

**node\_exporter** expose les métriques système (CPU, RAM, disque, réseau) sur `:9100/metrics`.

#### Linux (systemd)

```
# Télécharger node_exporter
wget
https://github.com/prometheus/node_exporter/releases/latest/download/node_exporter-1.11.0.linux-amd64.tar.gz
tar xvf node_exporter-*.tar.gz
sudo cp node_exporter-*/node_exporter /usr/local/bin/

# Créer un utilisateur dédié
sudo useradd --no-create-home --shell /bin/false node_exporter

# Créer le service systemd
sudo tee /etc/systemd/system/node_exporter.service <<EOF
[Unit]
Description=Node Exporter
After=network.target

[Service]
User=node_exporter
ExecStart=/usr/local/bin/node_exporter
Restart=on-failure

[Install]
WantedBy=multi-user.target
EOF

sudo systemctl daemon-reload
sudo systemctl enable --now node_exporter

# Vérifier
curl http://localhost:9100/metrics | head -20
```

📋 Copier

## macOS (homebrew)

```
brew install node_exporter

# Démarrer en tant que service
brew services start node_exporter

# Vérifier
curl http://localhost:9100/metrics | head -20
```

📋 Copier

Ajouter l'IP du Mac dans `prometheus.yml` sous le job `linux`.

---

## Agent Windows — windows\_exporter

**windows\_exporter** (anciennement `wmi_exporter`) expose les métriques Windows sur `:9182/metrics`.

## Téléchargement

Télécharger le fichier `.msi` correspondant à l'architecture de la machine (amd64 ou arm64) :

[https://github.com/prometheus-community/windows\\_exporter/releases](https://github.com/prometheus-community/windows_exporter/releases)

## Installation via l'installateur graphique

Double-cliquer sur le `.msi` en tant qu'**administrateur**.

### Écran 1 — Custom Setup :

- Activer **Firewall Exception** en cliquant sur l'icône **X** → cela ouvre automatiquement le port 9182 dans le pare-feu Windows
- Cliquer **Next**

### Écran 2 — Configuration :

Champ	Valeur
Collectors	<code>cpu,cs,logical_disk,memory,net,os,process,service,system</code>
Additional command line flags	laisser vide
Port to listen	9182
Path to config file	<code>config.yaml</code>

Cliquer **Next** → **Install**. Le service `windows_exporter` démarre automatiquement.

### Vérifier l'installation

```
Get-Service windows_exporter
Invoke-WebRequest http://localhost:9182/metrics | Select-Object -First 5
```

📄 Copier

### Collectors utiles

Collector	Métriques exposées
<code>cpu</code>	Utilisation CPU par cœur
<code>memory</code>	RAM utilisée/disponible
<code>logical_disk</code>	Espace disque, I/O
<code>net</code>	Trafic réseau par interface
<code>service</code>	État des services Windows
<code>process</code>	Processus actifs
<code>os</code>	Uptime, utilisateurs

## 9. Surveillance des ressources

---

### Métriques CPU

```
# Utilisation CPU globale (en %)
100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)

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

✂ Copier

## Métriques RAM

```
# RAM disponible Linux (en %)
(node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes) * 100

# RAM utilisée Windows (en %)
(1 - (windows_os_physical_memory_free_bytes / windows_cs_physical_memory_bytes)) * 100
```

✂ Copier

## Métriques disque

```
# Espace disque libre par partition Linux
(node_filesystem_avail_bytes{fstype!="tmpfs"} / node_filesystem_size_bytes{fstype!="tmpfs"})
* 100

# Windows
(windows_logical_disk_free_bytes / windows_logical_disk_size_bytes) * 100
```

✂ Copier

## Métriques réseau

```
# Débit entrant en Mo/s
rate(node_network_receive_bytes_total{device!="lo"}[5m]) / 1024 / 1024

# Débit sortant en Mo/s
rate(node_network_transmit_bytes_total{device!="lo"}[5m]) / 1024 / 1024
```

✂ Copier

---

## 10. Surveillance des services réseau

---

### Blackbox Exporter — sonder les services

**blackbox\_exporter** permet de tester la disponibilité de services (HTTP, DNS, TCP, ICMP).

Ajouter dans `docker-compose.yml` :

```
blackbox-exporter:
  image: prom/blackbox-exporter:latest
  container_name: blackbox-exporter
  restart: unless-stopped
  networks:
    - supervision
  ports:
    - "9115:9115"
  volumes:
    - ./blackbox/blackbox.yml:/etc/blackbox_exporter/config.yml
```

📄 Copier

### **blackbox/blackbox.yml :**

```
modules:
  http_2xx:
    prober: http
    timeout: 5s
    http:
      valid_http_versions: ["HTTP/1.1", "HTTP/2.0"]
      valid_status_codes: [] # 2xx par défaut
      follow_redirects: true

  http_post_2xx:
    prober: http
    http:
      method: POST

  tcp_connect:
    prober: tcp
    timeout: 5s

  icmp:
    prober: icmp
    timeout: 5s
    icmp:
      preferred_ip_protocol: "ip4"

  dns_check:
    prober: dns
    dns:
      query_name: "google.com"
      query_type: "A"
```

📄 Copier

Ajouter dans prometheus.yml :

```
# Sondes HTTP
- job_name: "blackbox_http"
  metrics_path: /probe
  params:
    module: [http_2xx]
  static_configs:
    - targets:
      - http://192.168.1.10      # Site interne
      - https://monsite.com    # Site externe
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: blackbox-exporter:9115

# Sondes TCP (vérifier un port ouvert)
- job_name: "blackbox_tcp"
  metrics_path: /probe
  params:
    module: [tcp_connect]
  static_configs:
    - targets:
      - 192.168.1.10:443      # HTTPS
      - 192.168.1.10:3306    # MySQL
      - 192.168.1.30:8530    # WSUS
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: blackbox-exporter:9115

# Sondes ICMP (ping)
- job_name: "blackbox_icmp"
  metrics_path: /probe
  params:
    module: [icmp]
  static_configs:
    - targets:
      - 192.168.1.1          # Routeur
      - 192.168.1.2          # Switch
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: blackbox-exporter:9115
```

📄 Copier

## Requêtes de disponibilité

```
# Service HTTP disponible (1 = up, 0 = down)
probe_success{job="blackbox_http"}

# Temps de réponse HTTP
probe_duration_seconds{job="blackbox_http"}

# Certificat SSL – jours avant expiration
(probe_ssl_earliest_cert_expiry - time()) / 86400
```

📄 Copier

---

## 11. Équipements réseau — SNMP Exporter

---

### Configuration snmp.yml

Le fichier `snmp.yml` définit quels OIDs scraper. Il est généré avec le **SNMP Exporter Generator** ou téléchargé depuis le dépôt officiel.

**snmp-exporter/snmp.yml** (extrait pour IF-MIB) :

```
modules:
  if_mib:
    walk:
      - 1.3.6.1.2.1.2          # interfaces
      - 1.3.6.1.2.1.31.1.1   # ifXTable (interfaces étendues)
      - 1.3.6.1.2.1.1        # system
    metrics:
      - name: ifHCInOctets
        oid: 1.3.6.1.2.1.31.1.1.1.6
        type: counter
        help: "Octets reçus sur l'interface"
        indexes:
          - labelname: ifIndex
            type: gauge
      - name: ifHCOutOctets
        oid: 1.3.6.1.2.1.31.1.1.1.10
        type: counter
        help: "Octets émis sur l'interface"
        indexes:
          - labelname: ifIndex
            type: gauge
    auth:
      community: supervision-snmp
    version: 2
```

📄 Copier

### Métriques SNMP dans Prometheus

```
# Trafic entrant sur une interface (Mo/s)
rate(ifHCInOctets{instance="192.168.1.1"}[5m]) / 1024 / 1024

# Trafic sortant
rate(ifHCOutOctets{instance="192.168.1.2"}[5m]) / 1024 / 1024
```

📄 Copier

---

## 12. Seuils et alertes

---

### Fichier de règles d'alertes

**prometheus/rules/alerts.yml :**

```
groups:
  - name: infrastructure
    rules:

      # CPU > 90 % pendant 5 minutes
      - alert: CPUElevé
        expr: >
          100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100) >
90
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "CPU élevé sur {{ $labels.instance }}"
          description: "CPU à {{ $value | printf "%.1f" }}% depuis 5 min."

      # RAM disponible < 10 %
      - alert: RAMCritique
        expr: >
          (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes) * 100 < 10
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "Mémoire critique sur {{ $labels.instance }}"
          description: "Seulement {{ $value | printf "%.1f" }}% de RAM disponible."

      # Disque < 15 % libre
      - alert: DisquePlein
        expr: >
          (node_filesystem_avail_bytes{fstype!="tmpfs"} /
node_filesystem_size_bytes{fstype!="tmpfs"}) * 100 < 15
        for: 10m
        labels:
          severity: warning
        annotations:
          summary: "Disque presque plein sur {{ $labels.instance }}"
          description: "Partition {{ $labels.mountpoint }} à {{ $value | printf "%.1f" }}%
libre."

      # Hôte non joignable (down)
      - alert: HôteIndisponible
        expr: up == 0
        for: 1m
```

```
labels:
  severity: critical
annotations:
  summary: "Hôte indisponible : {{ $labels.instance }}"
  description: "L'exporter {{ $labels.job }} ne répond plus."

# Service HTTP down
- alert: ServiceHTTPDown
  expr: probe_success{job="blackbox_http"} == 0
  for: 2m
  labels:
    severity: critical
  annotations:
    summary: "Service HTTP indisponible : {{ $labels.instance }}"

# Certificat SSL expire dans moins de 30 jours
- alert: CertificatSSLExpire
  expr: (probe_ssl_earliest_cert_expiry - time()) / 86400 < 30
  for: 1h
  labels:
    severity: warning
  annotations:
    summary: "Certificat SSL expire bientôt : {{ $labels.instance }}"
    description: "Expire dans {{ $value | printf "%.0f" }} jours."
```

📋 Copier

## Configuration Alertmanager — envoi d'alertes par email (Mailjet)

Les alertes sont automatiquement envoyées par email dès qu'une règle se déclenche. Le fichier `prometheus/alertmanager.yml` configure l'envoi via **Mailjet** (SMTP).

### 1. Récupérer les credentials Mailjet

Dans **Mailjet** → **Account Settings** → **SMTP & SEND API** :

- **API Key** → username SMTP
- **Secret Key** → password SMTP
- Vérifier l'adresse expéditeur dans **Senders & Domains**

### 2. Configurer alertmanager.yml

**prometheus/alertmanager.yml** :

```

global:
  smtp_smarthost: "in-v3.mailjet.com:587"
  smtp_from: "alertes@example.com"           # Expéditeur vérifié dans Mailjet
  smtp_auth_username: "MAILJET_API_KEY"     # API Key Mailjet
  smtp_auth_password: "MAILJET_SECRET_KEY"  # Secret Key Mailjet
  smtp_require_tls: true

route:
  group_by: ["alertname", "instance"]
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 4h
  receiver: "equipe-si"
  routes:
    - match:
        severity: critical
      receiver: "equipe-si-critical"

receivers:
  - name: "equipe-si"
    email_configs:
      - to: "si@example.com"
        send_resolved: true
        headers:
          Subject: "[SUPERVISION] {{ .GroupLabels.alertname }} - {{ .Status | toUpper }}"

  - name: "equipe-si-critical"
    email_configs:
      - to: "si-urgent@example.com"
        send_resolved: true
        headers:
          Subject: "[CRITIQUE] {{ .GroupLabels.alertname }} - {{ .Status | toUpper }}"

inhibit_rules:
  - source_match:
      severity: critical
    target_match:
      severity: warning
    equal: ["alertname", "instance"]

```

📄 Copier

### 3. Recharger Alertmanager sans redémarrage

```
docker compose exec alertmanager kill -HUP 1
```

📄 Copier

#### Flux des alertes

```

alerts.yml      → définit QUAND alerter (CPU > 90%, hôte down, disque plein...)
  ↓
Alertmanager   → reçoit l'alerte de Prometheus
  ↓
Mailjet SMTP    → envoie l'email
  ↓
si@example.com  → warning
si-urgent@example.com → critical

```

📄 Copier

Les alertes **warning** vont sur le destinataire standard, les alertes **critical** sur le destinataire urgent. Les deux reçoivent aussi un email quand l'alerte est résolue (`send_resolved: true`).

## 13. Plugins et exporters supplémentaires

### Exporters courants

Exporter	Port	Métriques	Installation
<b>node_exporter</b>	9100	Linux/macOS (CPU, RAM, disque, réseau)	Binaire ou package
<b>windows_exporter</b>	9182	Windows (même métriques)	MSI
<b>snmp_exporter</b>	9116	Équipements réseau SNMP	Docker
<b>blackbox_exporter</b>	9115	HTTP, TCP, ICMP, DNS	Docker
<b>mysqld_exporter</b>	9104	MySQL / MariaDB	Binaire
<b>postgres_exporter</b>	9187	PostgreSQL	Binaire
<b>nginx-prometheus-exporter</b>	9113	NGINX	Binaire
<b>cadvisor</b>	8080	Conteneurs Docker	Docker
<b>process_exporter</b>	9256	Processus Linux spécifiques	Binaire

### Ajouter cAdvisor pour Docker

```
# Ajouter dans docker-compose.yml
cadvisor:
  image: gcr.io/cadvisor/cadvisor:latest
  container_name: cadvisor
  restart: unless-stopped
  networks:
    - supervision
  ports:
    - "8080:8080"
  volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:ro
    - /sys:/sys:ro
    - /var/lib/docker:/var/lib/docker:ro
```

📄 Copier

Ajouter dans `prometheus.yml` :

```
- job_name: "docker"
  static_configs:
    - targets: ["cadvisor:8080"]
```

📄 Copier

---

## 14. Dashboards Grafana

---

### Connecter Alertmanager à Grafana

**Grafana → Connections → Data sources → Add data source → Alertmanager**

Champ	Valeur
Name	Alertmanager
Implementation	Prometheus <small>△</small> ne pas choisir Mimir
URL	http://alertmanager:9093
Access	Server (default)

Cliquer **Save & Test**. Tu peux ensuite consulter et silencer les alertes depuis **Grafana → Alerting**.

*Si Grafana affiche "detected a Prometheus endpoint", c'est que **Implementation** est sur Mimir — change-le sur Prometheus.*

---

### Provisionner la source de données Prometheus automatiquement

**grafana/provisioning/datasources/prometheus.yml :**

```
apiVersion: 1

datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9090
    isDefault: true
    editable: false
```

📄 Copier

### Importer des dashboards communautaires

Dans Grafana → **Dashboards → Import** → entrer l'ID :

ID	Dashboard	Utilisation
1860	Node Exporter Full	Linux/macOS — CPU, RAM, disque, réseau
14694	Windows Exporter	Windows — toutes métriques
11168	SNMP Exporter	Interfaces réseau via SNMP
7249	Kubernetes Cluster Monitoring	Pods, nœuds, ressources K8s
13659	Blackbox Exporter	Disponibilité HTTP/TCP/ICMP
193	Docker & cAdvisor	Conteneurs

## Exemple de panel PromQL dans Grafana

### Panel CPU Linux (Time series) :

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

📄 Copier

### Panel RAM disponible (Gauge) :

```
(node_memory_MemAvailable_bytes{instance="$instance"} /  
node_memory_MemTotal_bytes{instance="$instance"}) * 100
```

📄 Copier

### Panel trafic réseau (Time series — débit entrant) :

```
rate(node_network_receive_bytes_total{instance="$instance", device!="lo"}[5m]) / 1024 / 1024
```

📄 Copier

## Alertes Grafana (en complément d'Alertmanager)

Dans un panel Grafana → **Alert** → **Create alert rule** :

1. Définir la requête PromQL
2. Définir le seuil (ex: > 90 pour CPU)
3. Choisir le canal de notification (email, Slack, Teams)

## 15. Supervision des équipements réseau (switches, routeurs)

---

### Activer SNMP sur un switch Cisco (exemple)

```
! SNMPv2c (simple, à utiliser sur réseau isolé uniquement)
snmp-server community supervision-snmp RO
snmp-server location "Salle Serveurs – NOUVY"
snmp-server contact si@nouvy.lan

! SNMPv3 (recommandé)
snmp-server group SUPERVISION v3 priv read iso
snmp-server user admin-snmp SUPERVISION v3 auth sha MonPassword priv aes 128 MaCle
snmp-server host 192.168.1.50 version 3 priv admin-snmp
```

📄 Copier

## Tester SNMP depuis le serveur de supervision

```
# SNMPv2c – récupérer la description du système
snmpwalk -v2c -c supervision-snmp 192.168.1.1 sysDescr

# SNMPv3
snmpwalk -v3 -l authPriv -u admin-snmp
  -a SHA -A MonPassword
  -x AES -X MaCle
  192.168.1.1 sysDescr

# Trafic sur toutes les interfaces
snmpwalk -v2c -c supervision-snmp 192.168.1.1 ifHCInOctets
```

📄 Copier

## Métriques SNMP dans Grafana

Requêtes PromQL pour les équipements réseau :

```
# Trafic entrant (Mo/s) sur l'interface 1 du routeur
rate(ifHCInOctets{instance="192.168.1.1", ifIndex="1"}[5m]) / 1024 / 1024

# Interfaces UP/DOWN
ifOperStatus{instance="192.168.1.2"}
# 1 = up, 2 = down, 3 = testing
```

📄 Copier

---

## Récapitulatif — Ordre de mise en place

---

Étape	Action	Fichier / Commande
1	Créer la structure des dossiers	<code>mkdir -p supervision/prometheus/rules</code>
2	Rédiger <code>docker-compose.yml</code>	Stack Prometheus + Grafana + Alertmanager
3	Rédiger <code>prometheus.yml</code>	Jobs de scrape
4	Lancer la stack	<code>docker compose up -d</code>
5	Installer <code>node_exporter</code> sur Linux/macOS	Binaire + <code>systemd</code>
6	Installer <code>windows_exporter</code> sur Windows	MSI
7	Activer SNMP sur switches/routeurs	CLI équipement
8	Ajouter les cibles dans <code>prometheus.yml</code>	<code>static_configs</code>
9	Recharger Prometheus	<code>curl -X POST ../reload</code>
10	Configurer les alertes	<code>rules/alerts.yml</code> + <code>alertmanager.yml</code>
11	Importer les dashboards Grafana	ID 1860, 14694, 11168, 13659, 193
12	Tester les sondes Blackbox	HTTP, TCP, ICMP