

Stack de monitoring homelab — Prometheus + Grafana + Cloudflare Tunnel

Contexte et besoin

Gérer plusieurs serveurs — Pi local, serveur NAS, VPS chez différents hébergeurs — sans supervision centralisée revient à naviguer à l'aveugle. Disque plein, service tombé, pic CPU : on l'apprend par l'incident, pas avant.

Les besoins concrets :

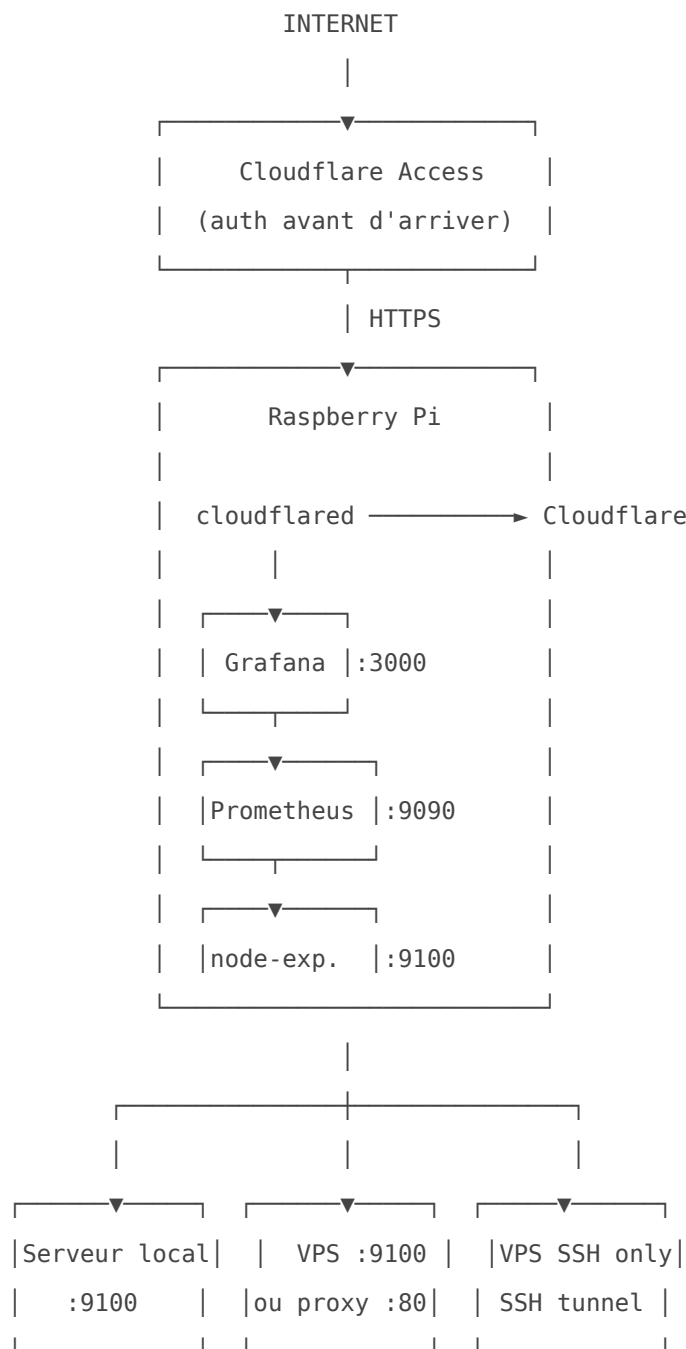
- Visualiser CPU, RAM, disque et uptime de chaque machine en un coup d'œil
- Être alerté avant saturation plutôt qu'après
- Accéder au dashboard depuis n'importe où, sans VPN
- Ne pas exposer de ports entrants sur le routeur domestique

Solution retenue

Besoin	Outil
Collecte des métriques sur chaque serveur	Node Exporter (agent léger, port 9100)
Agrégation et stockage time-series	Prometheus (scrape toutes les 15s)
Visualisation et dashboards	Grafana
Exposition publique sécurisée sans port ouvert	Cloudflare Tunnel + Cloudflare Access

Tout tourne sur un Raspberry Pi via Docker Compose. Les serveurs distants sont scrapés directement ou via tunnel SSH selon les contraintes réseau de chaque hébergeur.

Architecture



Rôles :

- **Node Exporter** — agent sur chaque serveur, expose CPU/RAM/disque/réseau sur `:9100/metrics`
- **Prometheus** — scrape les métriques toutes les 15s et les stocke (time-series DB)
- **Grafana** — visualise les données de Prometheus via dashboards
- **Cloudflare Tunnel** — expose Grafana publiquement sans ouvrir de port entrant sur le routeur

1. Node Exporter — installation sur chaque serveur

Avec sudo (Debian/Ubuntu)

```
sudo apt-get install -y prometheus-node-exporter
sudo systemctl enable --now prometheus-node-exporter
```

Écoute sur `0.0.0.0:9100` par défaut. Vérification : `curl http://localhost:9100/metrics | head -5`

Sans sudo (service utilisateur)

```
# Téléchargement
VERSION=1.8.2
ARCH=$(dpkg --print-architecture) # amd64 ou arm64
wget -q
https://github.com/prometheus/node_exporter/releases/download/v${VERSION}/node_exporter-
${VERSION}.linux-${ARCH}.tar.gz -O /tmp/ne.tar.gz
tar xzf /tmp/ne.tar.gz -C /tmp
mkdir -p ~/.local/bin
cp /tmp/node_exporter-${VERSION}.linux-${ARCH}/node_exporter ~/.local/bin/

# Service systemd utilisateur
mkdir -p ~/.config/systemd/user
cat > ~/.config/systemd/user/node-exporter.service << 'EOF'
[Unit]
Description=Prometheus Node Exporter
After=network.target

[Service]
ExecStart=%h/.local/bin/node_exporter --web.listen-address=0.0.0.0:9100
Restart=always

[Install]
WantedBy=default.target
EOF
```

```
loginctl enable-linger $USER    # démarre au boot sans session active
systemctl --user daemon-reload
systemctl --user enable --now node-exporter
```

2. Stack monitoring sur le Pi — Docker Compose

Arborescence

```
~/monitoring/
├─ docker-compose.yml
├─ prometheus.yml
├─ tunnel-entrypoint.sh      # si tunnel SSH nécessaire
└─ provisioning/
    └─ datasources/
        └─ prometheus.yml
```

docker-compose.yml

```
services:

  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    networks:
      - monitoring_default
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml:ro
      - prometheus_data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.retention.time=30d'
    restart: unless-stopped
```

```
node-exporter-pi:
  image: prom/node-exporter:latest
  container_name: node-exporter-pi
  networks:
    - monitoring_default
  pid: host
  volumes:
    - /:/host:ro,rslave
  command:
    - '--path.rootfs=/host'
  restart: unless-stopped
```

```
grafana:
  image: grafana/grafana:latest
  container_name: grafana
  networks:
    - monitoring_default
    - portainer_default    # si cloudflared est dans ce réseau
  volumes:
    - grafana_data:/var/lib/grafana
    - ./provisioning:/etc/grafana/provisioning
  environment:
    - GF_SERVER_HTTP_PORT=3000
    - GF_SERVER_ROOT_URL=https://grafana.mondomaine.com
    - GF_USERS_ALLOW_SIGN_UP=false
    - GF_SECURITY_COOKIE_SECURE=true
    - GF_SECURITY_STRICT_TRANSPORT_SECURITY=true
  restart: unless-stopped
```

```
networks:
  monitoring_default:
    driver: bridge
  portainer_default:
    external: true
```

```
volumes:
  prometheus_data:
  grafana_data:
```



Note réseau : Grafana doit être dans le même réseau Docker que cloudflared pour que le tunnel puisse l'atteindre via `http://grafana:3000`.

prometheus.yml

```
global:
  scrape_interval: 15s

scrape_configs:

  - job_name: 'pi-home'
    static_configs:
      - targets: ['node-exporter-pi:9100']    # container dans le même réseau Docker

  - job_name: 'serveur-local'
    static_configs:
      - targets: ['192.168.1.X:9100']        # réseau local, port direct

  - job_name: 'vps-standard'
    static_configs:
      - targets: ['IP_VPS:9100']             # VPS avec port 9100 accessible

  - job_name: 'vps-proxy'
    metrics_path: /metrics
    static_configs:
      - targets: ['IP_VPS:80']               # VPS avec proxy nginx sur port 80

  - job_name: 'vps-tunnel'
    static_configs:
      - targets: ['tunnel-container:9100']    # VPS via tunnel SSH
```

provisioning/datasources/prometheus.yml

Grafana charge automatiquement cette datasource au démarrage — pas besoin de configurer manuellement dans l'UI.

```
apiVersion: 1
datasources:
  - name: Prometheus
    type: prometheus
    url: http://prometheus:9090 # nom du container, même réseau Docker
    isDefault: true
    editable: true
```

Démarrage

```
cd ~/monitoring
docker compose up -d
docker compose ps # vérifier que tout est "Up"
```

3. Firewall — trois cas de figure

Cas 1 — Port 9100 accessible

Ouvrir uniquement depuis l'IP du Pi (ne jamais ouvrir à `0.0.0.0`) :

```
sudo ufw allow from IP_PI to any port 9100 proto tcp comment 'Prometheus'
sudo ufw reload
```

Diagnostic depuis le Pi :

```
nc -zv -w 5 IP_VPS 9100
# "succeeded"      → OK
# "timed out"      → bloqué par le provider (cas 2)
# "refused"        → port fermé côté serveur
```

Cas 2 — Port 9100 bloqué par le provider, mais 80/443 accessible

Proxy Nginx sur le VPS qui relaie `/metrics` vers node-exporter local :

```
# /etc/nginx/sites-available/node-exporter
server {
    listen 80;
    server_name IP_VPS;

    location /metrics {
        allow IP_PI;
        deny all;
        proxy_pass http://127.0.0.1:9100/metrics;
    }

    location / { return 404; }
}
```

```
sudo ln -s /etc/nginx/sites-available/node-exporter /etc/nginx/sites-enabled/
sudo nginx -t && sudo systemctl reload nginx
```

Dans `prometheus.yml` : `targets: ['IP_VPS:80']` + `metrics_path: /metrics`

Cas 3 — Seul le port 22 (SSH) est accessible

Tunnel SSH dans un container Docker dédié. Le container crée un tunnel persistant et expose le port 9100 du VPS dans le réseau Docker du Pi. Prometheus scrape le container comme s'il était local.

```
Pi                                VPS distant
tunnel-container:9100 —SSH→ localhost:9100
    ▲
Prometheus scrape
```

tunnel-entrpoint.sh :

```
#!/bin/sh
apk add --no-cache openssh-client > /dev/null 2>&1
while true; do
    ssh \
        -o StrictHostKeyChecking=no \
        -o ServerAliveInterval=30 \
```

```
-o ExitOnForwardFailure=yes \  
-i /root/.ssh/id_rsa \  
-N -L 0.0.0.0:9100:localhost:9100 \  
user@IP_VPS  
echo "Tunnel coupé, reconnexion dans 5s..."  
sleep 5  
done
```

Service dans docker-compose.yml :

```
tunnel-vps:  
  image: alpine:latest  
  container_name: tunnel-vps  
  networks:  
    - monitoring_default  
  volumes:  
    - /home/user/.ssh/id_rsa:/root/.ssh/id_rsa:ro  
    - ./tunnel-entrypoint.sh:/entrypoint.sh:ro  
  entrypoint: ["/entrypoint.sh"]  
  restart: unless-stopped
```

La clé SSH doit être copiée sur le Pi au préalable et avoir les bons droits :

```
chmod 600 ~/.ssh/id_rsa
```

4. Cloudflare Tunnel — exposition sécurisée

Cloudflare Tunnel crée une connexion **sortante** chiffrée du Pi vers Cloudflare. Aucun port entrant n'est ouvert sur le routeur. Le trafic entre par Cloudflare et ressort vers Grafana via le tunnel.

Déploiement cloudflared

```
docker run -d \  
  --name cloudflared \  
  --restart unless-stopped \  
  cloudflare/cloudflared
```

```
--network portainer_default \    # même réseau que Grafana
cloudflare/cloudflared:latest \
tunnel --no-autoupdate run --token TOKEN
```

Le token se génère dans **Cloudflare Zero Trust → Networks → Tunnels → Create a tunnel**.

Configuration du hostname public

Zero Trust → Networks → Tunnels → votre tunnel → Public Hostnames → Add :

Champ	Valeur
Subdomain	grafana
Domain	mondomaine.com
Service type	HTTP
URL	grafana:3000 (nom container)

“ **Piège fréquent** : Si cloudflared et Grafana ne sont pas dans le même réseau Docker, utiliser le nom de container ne fonctionnera pas. Vérifier avec `docker network inspect NOM_RESEAU`.

Cloudflare Access — authentification avant Grafana

Sans Access, l'URL est publique. Access ajoute une page d'auth Cloudflare devant l'application.

Zero Trust → Access → Applications → Add → Self-hosted :

```
Application domain : grafana.mondomaine.com
Policy              : Allow
Condition           : Emails → liste d'emails autorisés
MFA                 : Recommandé (TOTP ou hardware key)
```

Flux résultant :

```
Navigateur → grafana.mondomaine.com
            → Cloudflare Access (login email + MFA)
            → cloudflared → Grafana (login Grafana)
```

5. Requêtes PromQL — référence

```
# CPU en %
100 - (avg(rate(node_cpu_seconds_total{mode="idle", job="NOM"}[2m])) * 100)

# RAM utilisée en %
100 - (node_memory_MemAvailable_bytes{job="NOM"} / node_memory_MemTotal_bytes{job="NOM"} *
100)

# Disque / utilisé en %
100 - (
  node_filesystem_avail_bytes{job="NOM", mountpoint="/", fstype!="tmpfs"} /
  node_filesystem_size_bytes{job="NOM", mountpoint="/", fstype!="tmpfs"} * 100
)

# Uptime en secondes (afficher avec unité "s" dans Grafana)
node_time_seconds{job="NOM"} - node_boot_time_seconds{job="NOM"}

# Statut up/down
up{job="NOM"}    # 1 = up, 0 = down
```

6. Récapitulatif sécurité

Surface	Accès	Protection
Grafana (:3000)	Interne Docker uniquement	Cloudflare Access + auth Grafana
Prometheus (:9090)	Interne Docker uniquement	Pas exposé
Node Exporter Pi (:9100)	Interne Docker uniquement	Pas exposé
Node Exporter VPS	Port 9100 restreint ou proxy nginx	ufw allow from IP_PI
VPS backup	SSH tunnel	Aucun port supplémentaire ouvert
URL publique	HTTPS uniquement via Cloudflare	Cloudflare Access + MFA

Point de vigilance : Les règles ufw allow from IP_PI sont liées à l'IP externe du Pi. Si le FAI change cette IP, les VPS ne seront plus scrapés. Vérifier périodiquement avec curl -4 https://ifconfig.me depuis le Pi.

7. Commandes utiles

```
# Vérifier l'état de tous les targets Prometheus
docker exec prometheus wget -qO- 'http://localhost:9090/api/v1/targets' | \
  python3 -c 'import sys,json; d=json.load(sys.stdin); \
    [print(t["labels"]["job"], t["health"]) for t in d["data"]["activeTargets"]]'

# Recharger la config Prometheus sans redémarrage
docker exec prometheus kill -HUP 1

# Mettre à jour la datasource Grafana via API
curl -X PUT -H 'Content-Type: application/json' \
  -u 'admin:PASSWORD' \
  http://GRAFANA_IP:3000/api/datasources/uid/UID \
  -d
'{"name":"Prometheus","type":"prometheus","url":"http://prometheus:9090","isDefault":true,"access":"proxy"}'

# Logs du tunnel SSH
docker logs tunnel-vps --tail 20 -f

# Redémarrer uniquement Prometheus
docker compose restart prometheus
```

Revision #2

Created 2026-03-23 13:47:59 UTC by Flavien

Updated 2026-03-23 13:48:19 UTC by Flavien