Supervision du VPS avec Prometheus & Grafana

Métriques système, nginx et Nextcloud centralisées dans des tableaux de bord Grafana

1. Contexte et objectif

Grafana est une plateforme open source de visualisation de métriques. Associée à Prometheus (base de données de séries temporelles) et à des exporters spécialisés, elle permet de superviser en temps réel l'état d'un serveur : charge CPU, mémoire, espace disque, trafic réseau, mais aussi l'activité d'applications comme nginx ou Nextcloud.

Cette documentation présente le déploiement de Grafana et Prometheus via Docker sur un VPS Ubuntu, la configuration de trois exporters (node_exporter, nginx-prometheus-exporter, nextcloud-exporter), la résolution d'un problème de connectivité réseau entre conteneurs Docker, ainsi que la création de tableaux de bord dédiés.

2. Installation de Grafana

Grafana est déployé sous forme de conteneur Docker, avec ses données persistées dans un volume nommé. Les identifiants administrateur et l'URL publique sont définis via des variables d'environnement :

```
services:
  grafana:
    image: grafana/grafana:latest
    restart: unless-stopped
    network_mode: host
    volumes:
      - grafana_data:/var/lib/grafana
    environment:
      GF_SECURITY_ADMIN_USER: <utilisateur>
      GF_SECURITY_ADMIN_PASSWORD: ********
      GF_SERVER_ROOT_URL: https://grafana.example.com
      GF_SERVER_DOMAIN: grafana.example.com

volumes:
  grafana_data:
```

Démarrage du conteneur :

```
docker compose up -d
```

3. Exposition via nginx et certificat SSL

Un vhost nginx dédié redirige le sous-domaine grafana.example.com vers Grafana, qui écoute localement sur le port 3000 :

```
server {
    listen 80;
    server_name grafana.example.com;

    location / {
        proxy_pass http://127.0.0.1:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
```

```
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

Un certificat Let's Encrypt est ensuite généré et activé automatiquement avec Certbot :

```
certbot --nginx -d grafana.example.com --non-interactive --agree-tos -m <email> --redirect
```

4. Installation de Prometheus et des exporters

Prometheus et les trois exporters sont regroupés dans une seconde stack Docker, exécutés en network_mode: host afin de pouvoir accéder directement aux services locaux du serveur (nginx, Nextcloud) via 127.0.0.1 :

```
services:
  prometheus:
    image: prom/prometheus:latest
    network_mode: host
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
      - prom_data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.listen-address=127.0.0.1:9090'

  node_exporter:
    image: prom/node-exporter:latest
    pid: host
    network_mode: host
    volumes:
      - /proc:/host/proc:ro
      - /sys:/host/sys:ro
      - /:/rootfs:ro
    command:
      - '--path.procfs=/host/proc'
      - '--path.sysfs=/host/sys'
      - '--path.rootfs=/rootfs'
      - '--web.listen-address=127.0.0.1:9100'

  nginx_exporter:
    image: nginx/nginx-prometheus-exporter:latest
    network_mode: host
    command:
      - '--nginx.scrape-uri=http://127.0.0.1:8081/nginx_status'
      - '--web.listen-address=127.0.0.1:9113'

  nextcloud_exporter:
    image: xperimental/nextcloud-exporter:latest
    network_mode: host
    environment:
      NEXTCLOUD_SERVER: "https://cloud.example.com"
      NEXTCLOUD_AUTH_TOKEN: "<token serverinfo>"
      NEXTCLOUD_LISTEN_ADDR: "127.0.0.1:9205"

volumes:
  prom_data:
```

Configuration de Prometheus (prometheus.yml), listant les trois cibles à interroger toutes les 15 secondes :

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'node'
    static_configs:
      - targets: ['127.0.0.1:9100']
  - job_name: 'nginx'
    static_configs:
      - targets: ['127.0.0.1:9113']
  - job_name: 'nextcloud'
    static_configs:
      - targets: ['127.0.0.1:9205']
```

5. Activation des métriques nginx (stub_status)

Le module stub_status de nginx expose des statistiques basiques (connexions actives, requêtes traitées). Un vhost restreint à la boucle locale l'active sur un port dédié :

```
server {
    listen 127.0.0.1:8081;
    server_name _;
    location /nginx_status {
        stub_status;
        allow 127.0.0.1;
        deny all;
    }
}
```

Vérification :

```
$ curl -s http://127.0.0.1:8081/nginx_status
Active connections: 2
server accepts handled requests
 19099 19099 45975
Reading: 0 Writing: 2 Waiting: 0
```

6. Génération du token API Nextcloud (serverinfo)

L'application serverinfo, intégrée à Nextcloud, expose un point d'accès JSON (CPU, mémoire, utilisateurs, stockage, base de données). Son accès est protégé par un jeton généré via occ :

```
php occ config:app:set serverinfo token --value='<jeton généré>'
```

Test du point d'accès :

```
curl -s "https://cloud.example.com/ocs/v2.php/apps/serverinfo/api/v1/info?format=json" -H "NC-Token: <jeto
```

7. Connexion de Grafana à Prometheus

7.1 Problème rencontré : isolation réseau Docker

Grafana avait initialement été déployé sur un réseau Docker de type bridge (grafana_default), avec un mappage de port vers l'hôte. Prometheus et les exporters, eux, tournent en network_mode: host pour accéder directement aux services locaux. Ces deux conteneurs se trouvaient donc sur des espaces réseau différents : la source de données Prometheus configurée dans Grafana renvoyait

systématiquement « No data » sur tous les tableaux de bord.

Un test de connectivité depuis le conteneur Grafana confirme le problème : la passerelle du pont Docker (host.docker.internal → 172.17.0.1) ne parvient pas à joindre Prometheus, qui n'écoute que sur l'interface de boucle locale de l'hôte :

```
$ docker exec grafana-grafana-1 wget -qO- http://host.docker.internal:9090/api/v1/status/config
wget: can't connect to remote host (172.17.0.1): Operation timed out
```

7.2 Solution : basculer Grafana en network_mode: host

Plutôt que d'exposer Prometheus publiquement pour le rendre joignable depuis le pont Docker, Grafana est reconfiguré pour partager directement l'espace réseau de l'hôte, comme Prometheus et les exporters. Les deux services communiquent alors simplement via 127.0.0.1, sans exposition réseau supplémentaire :

```
services:
  grafana:
    image: grafana/grafana:latest
    network_mode: host
    volumes:
      - grafana_data:/var/lib/grafana
    environment:
      GF_SECURITY_ADMIN_USER: <utilisateur>
      GF_SECURITY_ADMIN_PASSWORD: ********
      GF_SERVER_ROOT_URL: https://grafana.example.com
```

La configuration nginx est mise à jour en conséquence (proxy_pass vers 127.0.0.1:3000, port d'écoute par défaut de Grafana), puis la source de données Prometheus est ajoutée dans Grafana avec l'URL http://127.0.0.1:9090. Un test de santé confirme la connexion :

```
$ curl -s -u <utilisateur>:******** https://grafana.example.com/api/datasources/uid/<uid>/health
{"details":{"application":"Prometheus"},"message":"Successfully queried the Prometheus API.","status":"OK"
```

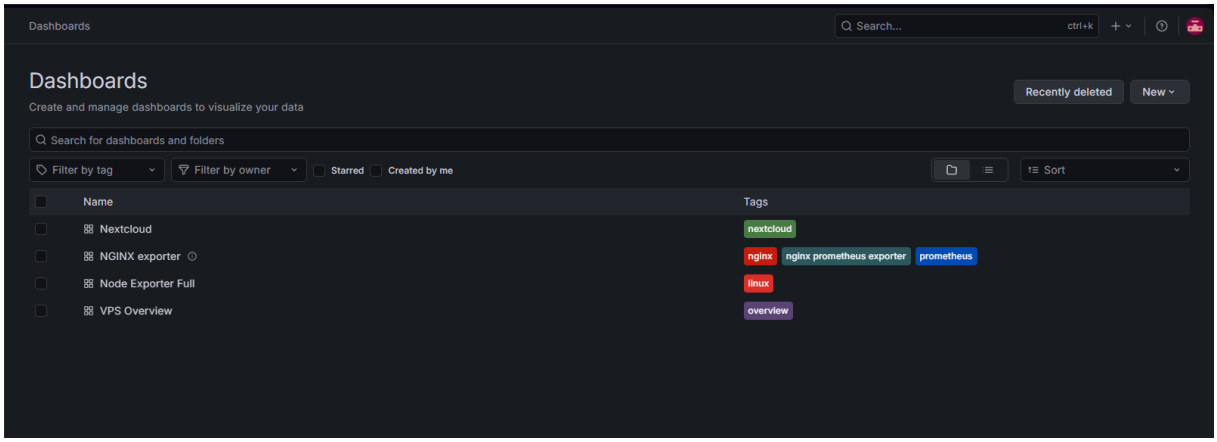
8. Création des tableaux de bord

Quatre tableaux de bord sont mis en place dans Grafana :

- **VPS Overview** — synthèse : CPU, RAM, disque système, disque du volume de stockage, trafic réseau, connexions nginx, statut Nextcloud, uptime.
- **Node Exporter Full** — tableau de bord communautaire (ID 1860), détail complet des métriques serveur (CPU par cœur, mémoire, disques, réseau, processus).
- **NGINX exporter** — tableau de bord officiel de l'exporter nginx (ID 12708) : connexions actives, requêtes par seconde, taux d'erreurs.
- **Nextcloud** — tableau de bord personnalisé construit à partir des métriques serverinfo : statut, nombre d'utilisateurs et de fichiers, espace libre, taille de la base de données, répartition des partages par type.

Les tableaux de bord communautaires sont importés directement depuis grafana.com via l'API Grafana, en associant la source de données Prometheus créée précédemment :

```
curl -s -X POST -H "Content-Type: application/json" \
  -d @import_payload.json \
  https://grafana.example.com/api/dashboards/import
```



Les quatre tableaux de bord répertoriés dans Grafana (Dashboards).

9. Vérification finale

Après rechargement, les quatre tableaux de bord affichent des données en temps réel, rafraîchies toutes les 30 secondes. Le tableau Nextcloud confirme notamment un espace libre de 137 Gio sur le volume dédié, ainsi que le nombre de fichiers et d'utilisateurs actifs.

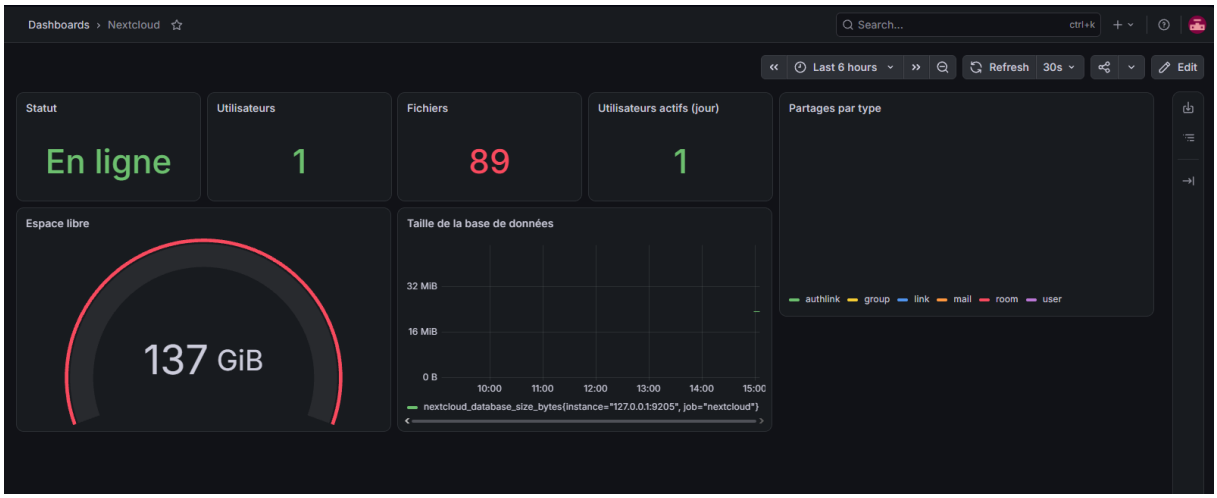


Tableau de bord Nextcloud : statut, utilisateurs, fichiers, espace libre, taille de la base de données et répartition des partages, alimentés en temps réel par l'exporter.

10. Conclusion

Cette mise en place permet de superviser en continu la santé du VPS (ressources système), du serveur web nginx et de l'instance Nextcloud, depuis une interface unique accessible à l'adresse https://grafana.example.com. Le principal point d'attention rencontré concernait l'isolation réseau entre conteneurs Docker utilisant des modes réseau différents (bridge vs host), résolu en unifiant l'ensemble des services de supervision sur le réseau de l'hôte.