Mise en place Prometheus & Grafana — module mineur

1) But

- **Prometheus** : récupère régulièrement des métriques de nos services.
- **Grafana** : visualise ces métriques en dashboards.
- Objectif : voir la santé du système (backend/API, conteneurs) et détecter les problèmes tôt.

2) Ce qu'on a mis en place

- **Backend (Fastify + prom-client)** expose `GET /metrics` :
    - Histogramme de durée des requêtes `http_request_duration_seconds_bucket` (+ `*_count`, `*_sum`).
    - Compteur de requêtes.
    - Métriques process (CPU, mémoire) : `process_cpu_seconds_total`, `process_resident_memory_bytes`, etc.
- **cAdvisor** : métriques Docker par conteneur (CPU, RAM, réseau, disque).
- **Prometheus** :
    - Scrape `backend:8000/metrics`, `cadvisor:8080/metrics`, `prometheus:9090/metrics`, `grafana:3000/metrics`.
    - Stocke avec historique (rétention 15 jours).
- **Grafana** :
    - Datasource Prometheus.
    - Dashboard ft_transcendence (req/s, p95, erreurs 5xx, CPU/Mem, health).
- **Alertmanager** :
    - Reçoit les alertes de Prometheus (routing simple via webhook par défaut).

---

3) Fichiers clés

3.1 `backend/src/metrics.ts`

- Initialise `prom-client`.
- Enregistre les **default metrics** (`collectDefaultMetrics`).
- Expose `GET /metrics`.
- Ajoute des hooks pour mesurer **latence** et **compter** les requêtes.

Sans ça, Prometheus n'a rien à scraper côté app.

3.2 `ops/prometheus/prometheus.yml`

- Configure Prometheus (scrapes + alerting).
  Targets (réseau Docker, pas `localhost`) :
  - `backend:8000/metrics`
  - `cadvisor:8080/metrics`
  - `prometheus:9090/metrics`
  - `grafana:3000/metrics` (avec `metrics_path: /metrics`)

- Intervalle : `scrape_interval: 15s`
- Section `alerting` → `alertmanager:9093`

3.3 `ops/prometheus/rules.yml`

- **Recording rules** : calculs utiles (ex. p95).
  Alerting rules : `BackendDown`, `HighErrorRate (5xx)`, `HighLatencyP95`, etc.

3.4 (Option) `ops/grafana/provisioning/datasources/datasource.yml`

- Déclare automatiquement Prometheus (`http://prometheus:9090`) comme datasource par défaut.
  Si non utilisé, ajoute la datasource via l'UI (Connections → Data sources).

3.5 Volumes (`docker-compose.yml`)

```
volumes:
  grafana-data:       # persiste dashboards, users, config Grafana
  prometheus-data:    # persiste séries temporelles Prometheus
```

3.6 Services ajoutés (`docker-compose.yml`)

- `prometheus`, `grafana`, `cadvisor`, **`alertmanager`**

---

4) Dashboard Grafana — 4 panneaux de base (PromQL)

Remplace/ajuste `job="backend"` et les labels selon ton exposition.

1. **Request rate total (req/s)**

```
sum(rate(http_request_duration_seconds_count{job="backend"}[1m]))
```

2. **p95 latency par route (s)**

```
histogram_quantile(
  0.95,
  sum by (le, route)
(rate(http_request_duration_seconds_bucket{job="backend"}[5m]))
)
```

3. **Taux d’erreur 5xx (%)**

```
sum(rate(http_request_duration_seconds_count{job="backend",
status_code=~"5.."}[5m]))
/
sum(rate(http_request_duration_seconds_count{job="backend"}[5m]))
```

4. **CPU backend**

- **Process (prom-client)** → fraction de cœur :

```
rate(process_cpu_seconds_total{job="backend"}[2m])
```

- **Conteneur (cAdvisor)** → fraction de cœur :

```
rate(container_cpu_usage_seconds_total{container_label_com_docker_compose_service="backend"}[2m])
```

(+ bonus mémoire process)

```
process_resident_memory_bytes{job="backend"}
```

(+ health)

```
up{job="backend"}
```

---

5) Lire ces métriques (diagnostic rapide)

- p95 ↑ mais CPU normal → I/O (DB, réseau) probable.
- p95 ↑ + CPU ↑ → saturation backend.
- 5xx ↑ sur une route → bug/crash sur cette API.
- cAdvisor → repérer un conteneur qui “mange” RAM/CPU.

---

6) Alertes (recommandées pour le module)

Dans `rules.yml` (extraits) :

- **Backend DOWN**

```
- alert: BackendDown
  expr: up{job="backend"} == 0
  for: 1m
  labels: { severity: critical, service: backend }
```

- **Taux d’erreur > 5%**

```
- alert: HighErrorRate
  expr: (
    sum(rate(http_request_duration_seconds_count{job="backend",
status_code=~"5.."}[5m]))
    /

sum(rate(http_request_duration_seconds_count{job="backend"}[5m]))
  ) > 0.05
  for: 5m
  labels: { severity: warning, service: backend }
```

- **Latence p95 > 500ms**

```
- alert: HighLatencyP95
  expr: histogram_quantile(0.95,
         sum by (le, route)
(rate(http_request_duration_seconds_bucket{job="backend"}[5m]))
       ) > 0.5
  for: 5m
  labels: { severity: warning, service: backend }
```

(Tu peux ajouter CPU/Mem si tu veux, mais ces 3 couvrent l’essentiel du sujet.)

---

7) Rétention & sécurité

- Prometheus : `--storage.tsdb.retention.time=15d`
- Grafana (`.env`) :

```
GF_SECURITY_ADMIN_USER=admin
GF_SECURITY_ADMIN_PASSWORD=tonpassword
GF_AUTH_ANONYMOUS_ENABLED=false
GF_USERS_ALLOW_SIGN_UP=false
GF_SECURITY_SECRET_KEY=valeur_bien_longue_et_secrète
```

8) Lancer & ouvrir

Après `make` / `docker compose up -d` :

- **Prometheus** (targets, rules, alerts) → `http://localhost:9090`
    - Targets : `Status → Targets`
    - Alertes : `Alerts`
- **Grafana** (dashboards) → `http://localhost:3000`
    - Login = `.env`
- **Alertmanager** (alertes reçues) → `http://localhost:9093`
- **cAdvisor** (containers) → `http://localhost:8081`

9) Tests

9.1 Charge simple et observation dashboard

- **Via le proxy** :

```
for i in {1..200}; do curl -s -o /dev/null
http://localhost:8080/api/health; done
```

Observe dans Grafana : **Request rate**, **p95**, **Avg response**, **CPU/Mem**.

9.2 Déclencher “BackendDown”

```
docker compose stop backend
# Attendre ~1 minute (for: 1m)
```

- **Prometheus** → **Alerts** : BackendDown passe **Pending** puis **Firing**.
- **Alertmanager** → **Alerts** : groupe BackendDown visible.

```
docker compose start backend
```

- L’alerte passe **Resolved**.

10) Commandes utiles (cheat-sheet)

```
# UIs
xdg-open http://localhost:9090 || open http://localhost:9090
xdg-open http://localhost:3000 || open http://localhost:3000
xdg-open http://localhost:9093 || open http://localhost:9093
xdg-open http://localhost:8081 || open http://localhost:8081

# Charge simple (via proxy)
for i in {1..200}; do curl -s -o /dev/null
http://localhost:8080/<TA_ROUTE>; done

# Stop / start backend pour tester BackendDown
docker compose stop backend
docker compose start backend

# Rechargement Prometheus après modif de config/règles
curl -X POST http://localhost:9090/-/reload

# Sanity checks
docker compose ps
docker logs -n 50 prometheus
docker exec -it prometheus sh -lc 'wget -qO-
http://alertmanager:9093/-/healthy'
```