Infrastructure de Supervision – OpenTelemetry / Prometheus / Grafana / Loki

Architecture

Deux serveurs supervisés (p01, p02), chacun avec un **OTel Collector Contrib** local. Les outils de centralisation (Prometheus, Grafana, Loki) tournent en **Docker Compose** sur p02.

```
p01/p02  →  OTel Collector  →  Prometheus (métriques)  →  Grafana
                            →  Loki (logs)             →  Grafana
```

Configuration OTel Collector

Receivers

`journald` – collecte les logs systemd des unités configurées (sshd, nginx, docker, jupyterhub, kafka) à partir du niveau `info`.

`hostmetrics` – collecte toutes les 30s les métriques système : CPU, mémoire, charge, pagination, et filesystem sur les points de montage listés en correspondance stricte.

Processors

Processeur	Rôle
`resourcedetection/system`	Ajoute le hostname (source : OS) comme attribut à toutes les données
`memory_limiter`	Rejette les données si la RAM dépasse 75 % (vérification chaque seconde, marge de 15 %)
`batch`	Regroupe les données en lots de 100 éléments ou toutes les 10s avant export

> **`memory_limiter` doit toujours être placé en premier dans la liste des processors.**

Exporters

`prometheus` – expose les métriques sur `0.0.0.0:8810` au format Prometheus/OpenMetrics. Un namespace et un label constant (hostname) sont ajoutés à toutes les métriques. Les attributs de ressource sont convertis en labels. Expiration des métriques après 180 min d'inactivité.

`otlphttp/loki` – envoie les logs vers l'endpoint OTLP de Loki (`port 3100`) en HTTP.

Pipelines

```
metrics: hostmetrics → [resourcedetection, memory_limiter, batch]
→ prometheus
logs:    journald    → [resourcedetection, memory_limiter, batch]
→ otlphttp/loki
```

> **`telemetry.metrics: level: none` désactive les métriques internes du Collector.**

Configuration Prometheus

Ajouter un job de scraping par serveur ciblant le port `8810`. Aligner l'intervalle de scraping sur le `collection_interval` du Collector (30s).

> **Les targets doivent être accessibles depuis le réseau du conteneur Docker de Prometheus.**

Configuration Loki

Activer le récepteur OTLP (`otlp_config`) pour recevoir les logs sur le port `3100/otlp`. Les attributs ajoutés par `resourcedetection` (hostname, etc.) sont automatiquement indexés comme labels, exploitables en LogQL.

Configuration Grafana

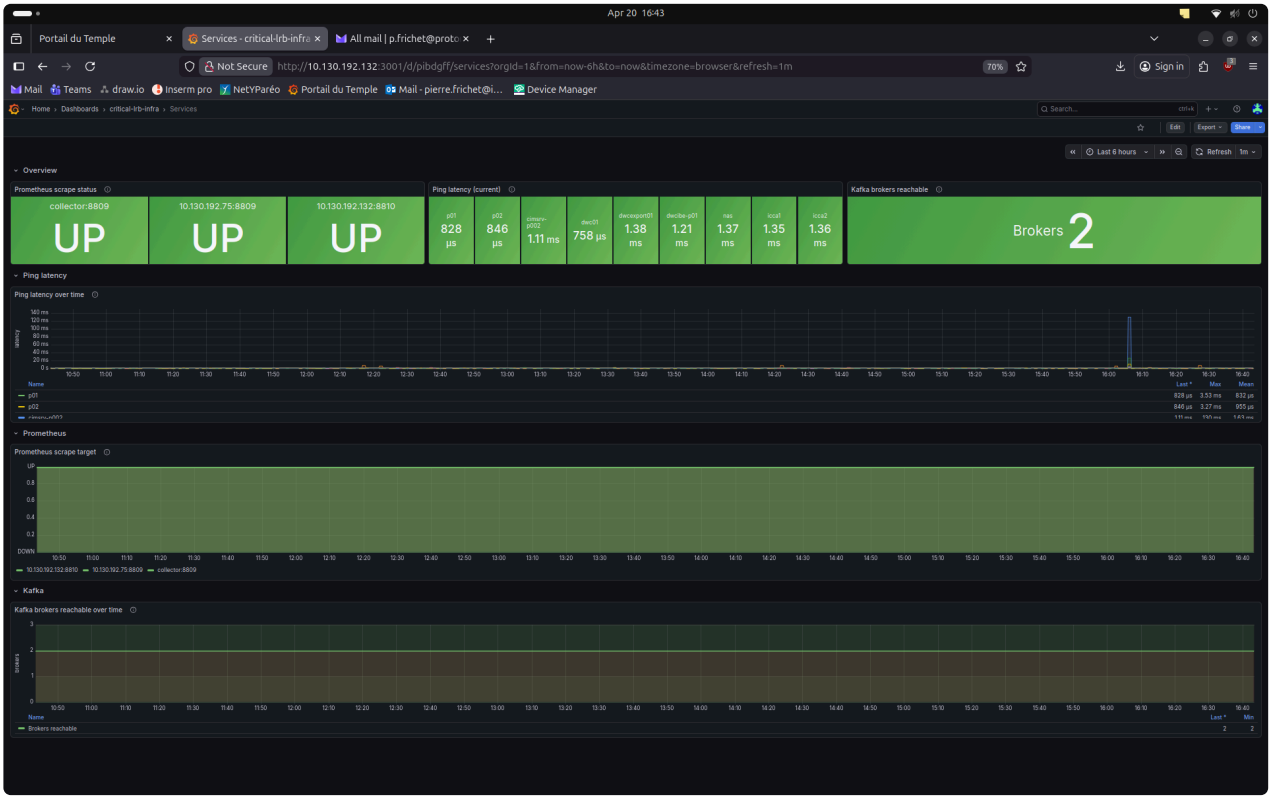
Ajouter deux datasources :

Source	URL
Prometheus	`http://prometheus:9090`
Loki	`http://loki:3100`

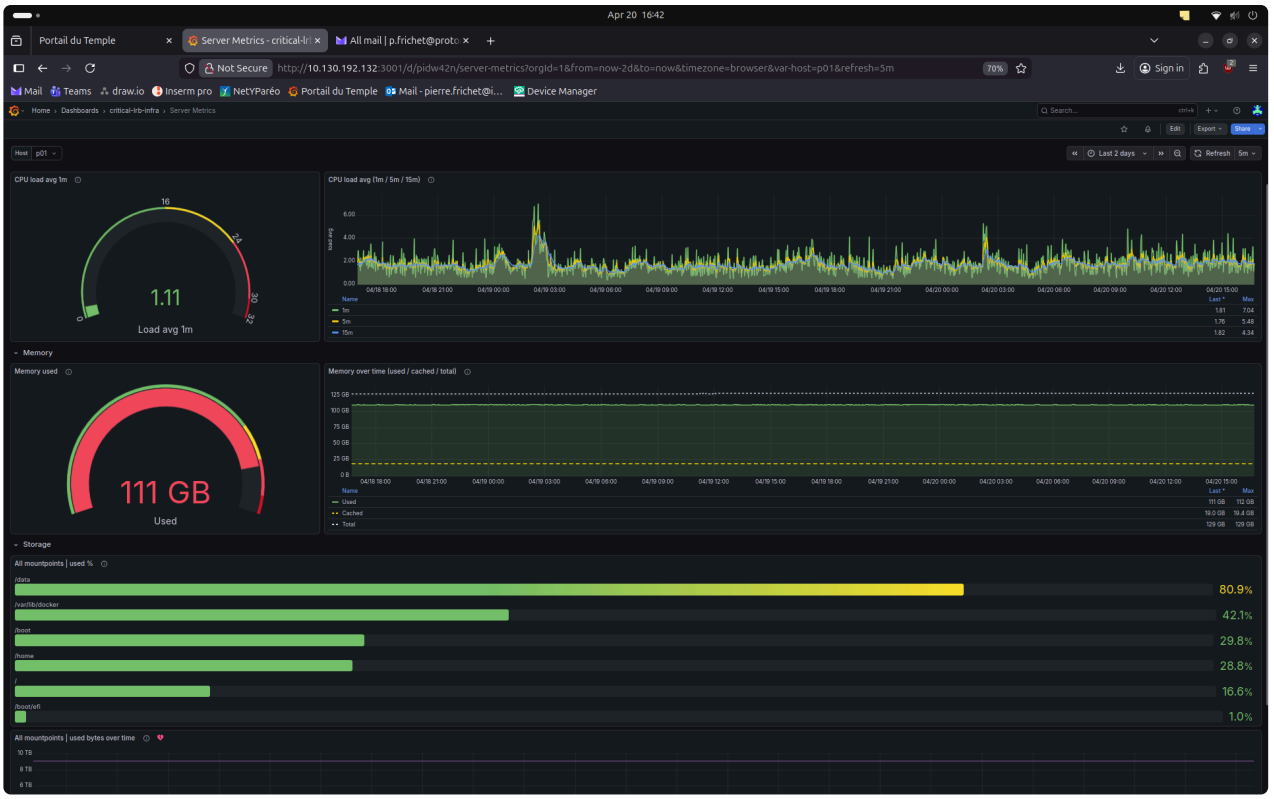
Les URLs utilisent les noms de services Docker Compose (résolution DNS interne). Les dashboards métriques se basent sur les données hostmetrics ; les logs s'explorent via **Explore > Loki** avec LogQL (ex : `{unit="nginx"}`).

Creation des dashboards

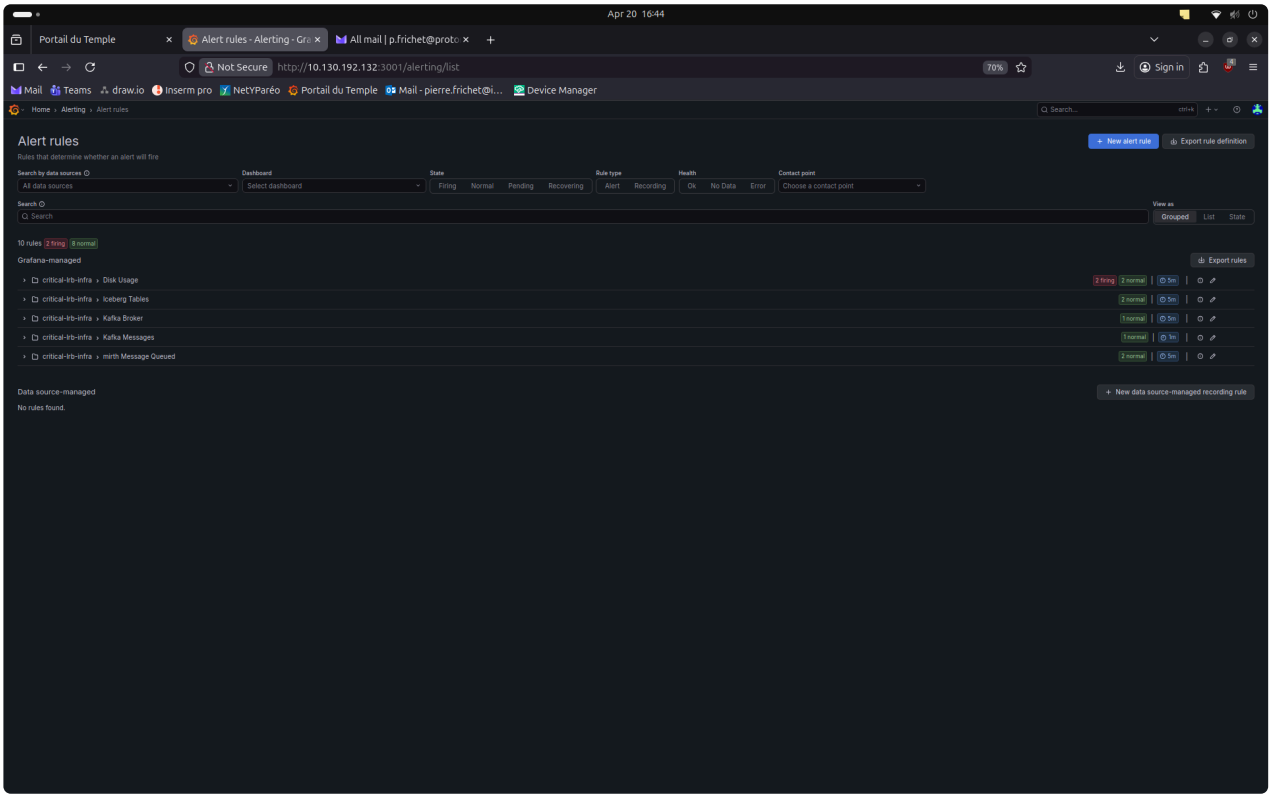
Services :



Serveurs :



Creation des alertes



Vérification

- [ ] Service OTel Collector actif sur p01 et p02
- [ ] Endpoint `:8810` accessible depuis Prometheus

- [ ] **Prometheus > Status > Targets** : p01 et p02 en `UP`
- [ ] **Grafana > Explore > Loki** : logs récents présents avec labels `hostname` et `unit`
- [ ] **Grafana > Data Sources** : connexion Prometheus et Loki réussies

> **En cas de problème Vérifier dans l'ordre : état du service Collector → accessibilité réseau (ports 8810/3100) → logs du Collector → targets Prometheus.**

BTS SIO SISR – E5 | Supervision système et réseau