DE LA RÉUNION Institut Universitaire de Technologie	R502	TP 1 – Supervision des Réseaux	ALAMELOU - Rohan

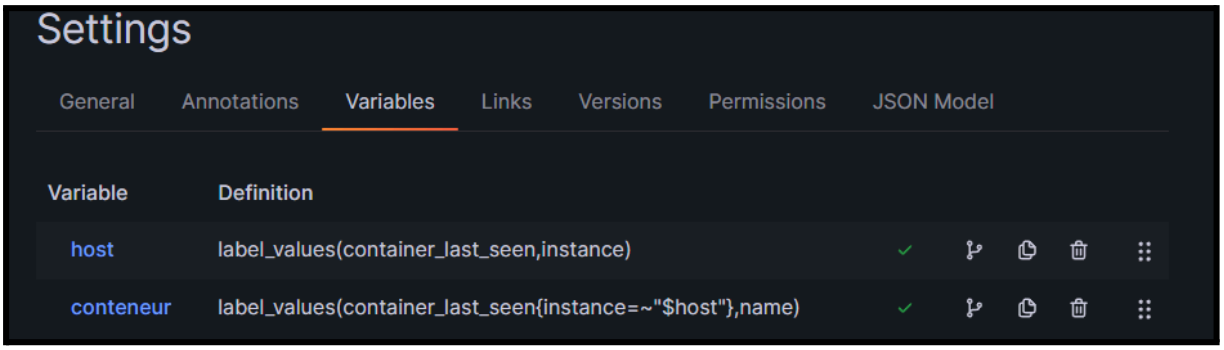
Grafana : Supervision des Conteneurs pour la Base TAAF

Promotion : BUT RT3 TP1

Date de rendu : 31/10/2025

Question 1 - Variables de base

Dans un premier temps voici les variables que j'ai créer :



et voici ce que contient la variable hosts que j'ai créer.

Ici dans host la métrique container_last_seen provient que du service cAdvisor, qui centralise les métriques de tous les conteneurs Docker.

Donc prometheus ne voit qu'une seule "instance" correspondant à exporter cadvisor.

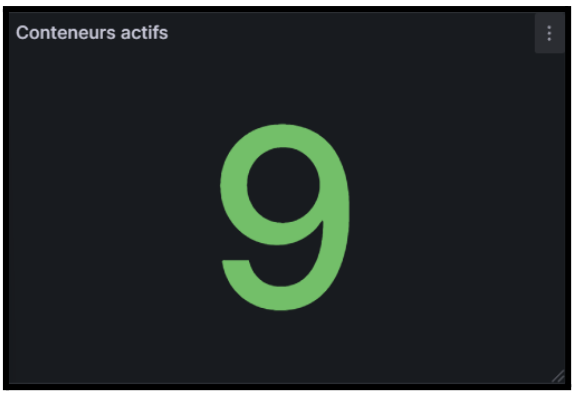
Question 2 - Métriques simples

Requête PromQL :

```
count(container_last_seen{name!="", instance=~"$host"})
```

Résultat : On a le total de conteneurs actifs qui est 9

Capture du panel fonctionnelle :



Question 3 - Filtrage par conteneur

Requête PromQL :

Dans un premier temps j'ai utilisé une requête pour afficher un avec un camembert ce qui facilite bien la vue de la consommation des 5 conteneurs.

```
topk(5, sum by
(name)(container_memory_usage_bytes{instance=~"$host",
name!=""}))
```

Explication :

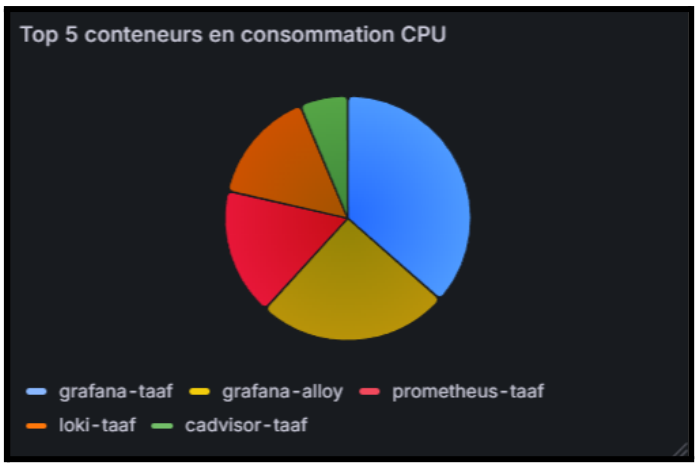
Cette requête utilise la métrique "container_memory_usage_bytes" pour mesurer la mémoire utilisée par chaque conteneur Docker.

Le filtre instance=~"$host" cible l'hôte sélectionné dans le dashboard, et name!="" exclut les conteneurs sans nom.

La fonction sum by (name) regroupe la consommation mémoire par conteneur,
et topk(5, ...) extrait les 5 conteneurs les plus consommateurs.

Cela permet d'identifier instantanément les services les plus gourmands en RAM.

Capture du panel fonctionnelle :



J'ai ensuite fait une requête promQL pour afficher sur le temps l'évolution de la consommation en CPU de chacun des 5 conteneurs :

```
topk(5, sum by (name) (rate(container_cpu_usage_seconds_total{name!="" ,
instance=~"$host"}[5m])) * 100)
```

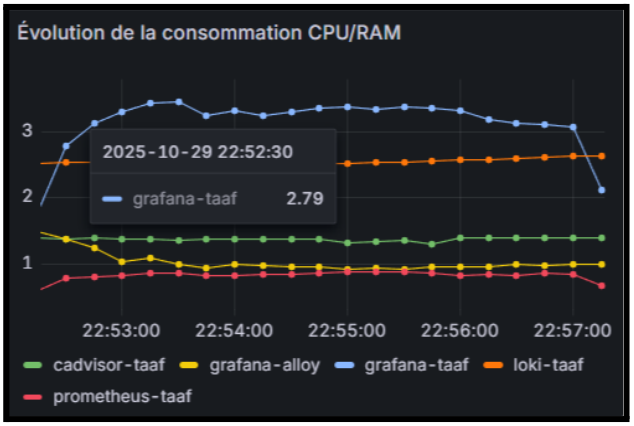
Explication :

Cette requête permet d'afficher les 5 conteneurs les plus consommateurs de CPU.

La fonction "rate()" mesure la variation moyenne du temps CPU sur 5 minutes, sum by (name) regroupe les valeurs par conteneur, et "topk(5, ...)" sélectionne les 5 plus élevés.

Le résultat est multiplié par 100 pour obtenir un pourcentage d'utilisation CPU.

Capture du panel fonctionnelle :



Question 4 - Calcul de pourcentage (graphiques)

Requête PromQL :

```
sum by (name)
(rate(container_cpu_usage_seconds_total{name="postgres-af",
instance=~"$host"}[5m])) * 100
```

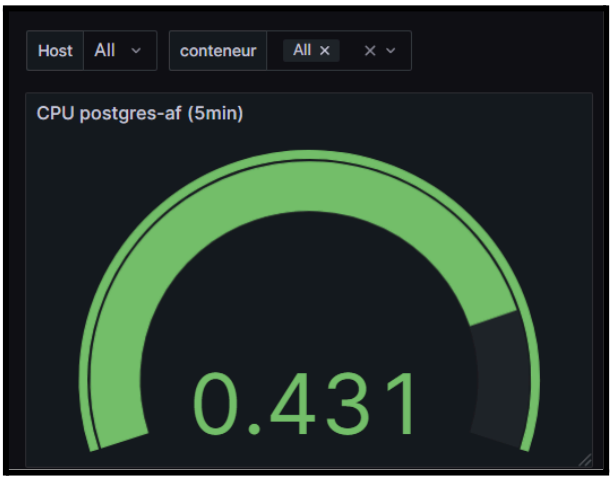
Explication :

Cette requête calcule le pourcentage d'utilisation CPU moyen du conteneur postgres-af sur les 5 dernières minutes.

La fonction "rate()" mesure la vitesse d'utilisation du CPU, "sum by" regroupe les valeurs par conteneur, et la multiplication par 100 permet d'obtenir le résultat en pourcentage.

Cela permet de surveiller la charge du serveur PostgreSQL et de détecter des surcharges CPU.

Capture du panel fonctionnelle :



Question 5 - Analyse temporelle

Requête PromQL :

```
container_memory_usage_bytes{name="prometheus-taaf",
instance=~"$host"}
```

Explication :

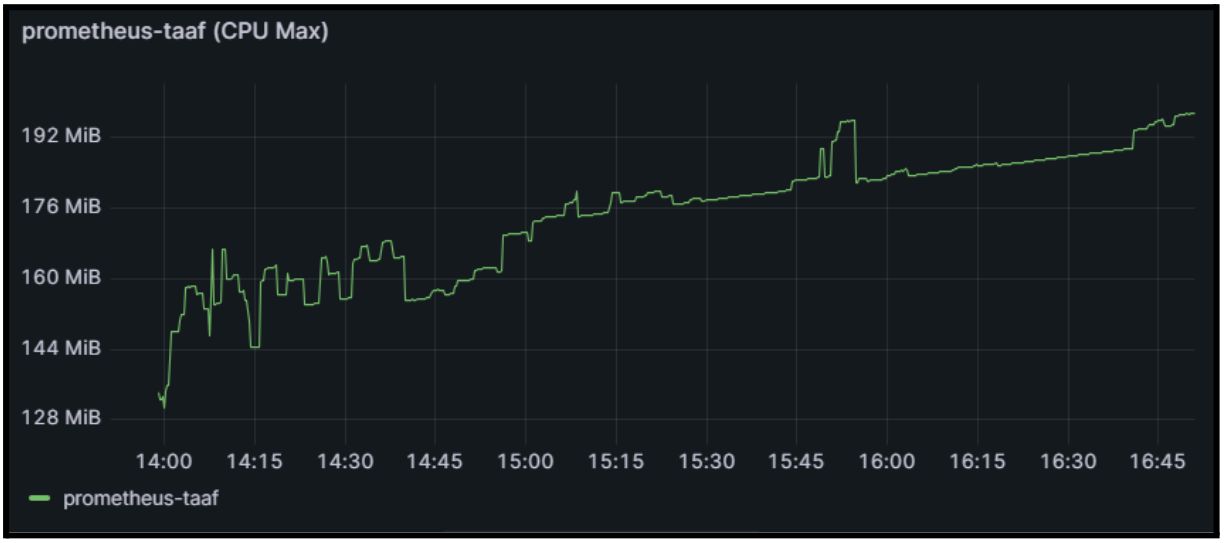
Cette requête affiche la quantité de mémoire utilisée par le conteneur prometheus-taaf.

La métrique container_memory_usage_bytes correspond à la mémoire totale consommée par le conteneur,

et les filtres name="prometheus-taaf" et instance=~"$host" permettent de cibler uniquement ce conteneur sur l'hôte sélectionné.

Ce graphique permet d'observer les pics de consommation mémoire et de détecter les phases de forte activité, comme la compaction des données dans Prometheus.

Capture du panel fonctionnelle :



Le pic vient du travail interne de Prometheus pour gérer ses données, ce n'est pas une erreur, mais une activité normale du service.

Question 6 - Métriques réseau

Requête PromQL :

```
topk(1, sum by (name) (
  increase(container_network_transmit_bytes_total{name!="" ,
instance=~"$host"}[1h])))
```

Explication :

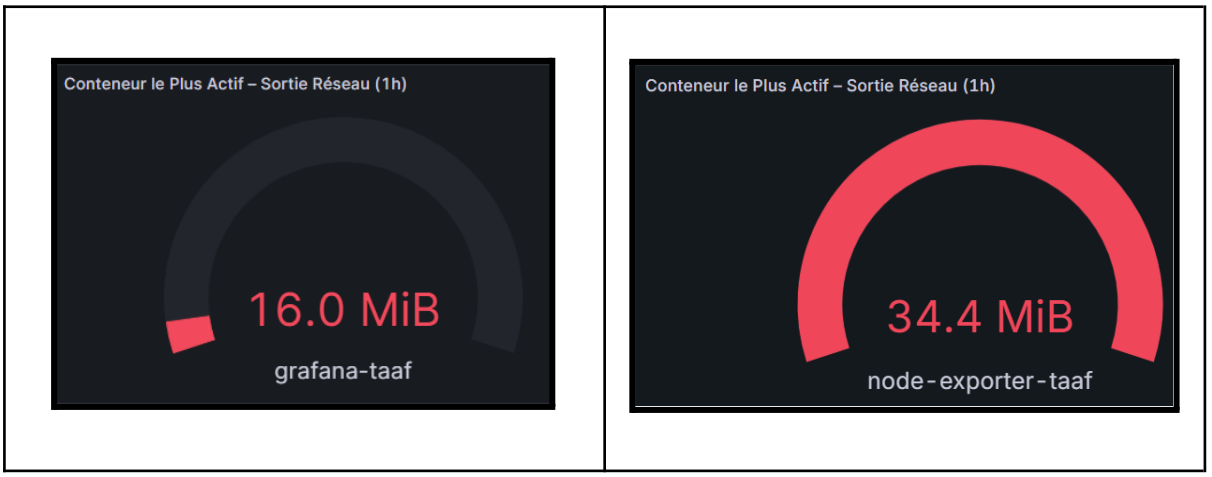
Cette requête affiche le conteneur ayant généré le plus de trafic réseau sortant sur la dernière heure.

La métrique container_network_transmit_bytes_total mesuré le volume d'octets transmis, et la fonction "increase(...[1h])" va calculer la quantité envoyée pendant l'heure écoulée.

Ensuite "sum by" regroupe les valeurs par conteneur et enfin "topk" affiche celui qui a transmis le plus de données.

Dans mon cas, c'est le conteneur grafana-taaf, car il exporte en continu des métriques vers Grafana.

Capture du panel fonctionnelle :



Question 7 - Corrélation métrique

Requêtes PromQL :

```
sum by (name)
(rate(container_cpu_usage_seconds_total{name="postgres-af",
instance=~"$host"}[5m])) * 100
```

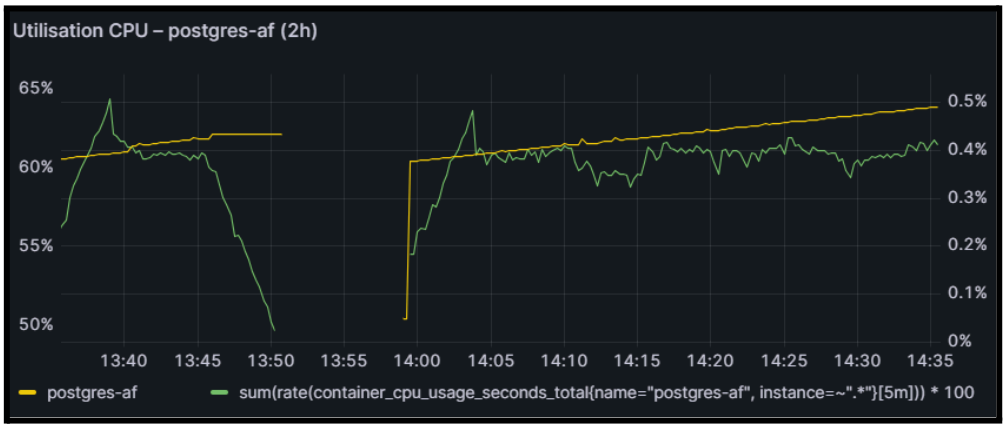
```
container_memory_usage_bytes{name="postgres-af",
instance=~"$host"}
```

Explication :

Ces deux requêtes permettent de suivre simultanément la charge CPU et la consommation mémoire du conteneur postgres-af.

La première calcule le pourcentage moyen de CPU utilisé sur 5 minutes (rate + * 100), et la seconde affiche la mémoire totale occupée (container_memory_usage_bytes).

Il a donc fallu faire deux requêtes pour pouvoir superposer les deux courbes deux demandes et faire la corrélation.



La corrélation ici est forte, l'activité du CPU influence directement la mémoire consommée par le conteneur PostgreSQL

Postgres exécute plusieurs requêtes SQL ou écrit beaucoup de données donc ça utilise du CPU.

Question 8 - Alertes de seuil

Requêtes PromQL :

```
((container_memory_usage_bytes{name!="",job="cadvisor"}/(9.91*1024*1024*1024))*100)and((container_memory_usage_bytes{name!="",job="cadvisor"}/(9.91*1024*1024*1024))*100>80)
```

Explication :

Dans un premier temps je suis partie regarder combien de mémoire est allouée par conteneurs pour mener à bien le calcul :

```
debian@debian:~/monitoring$ sudo docker stats --no-stream
[sudo] Mot de passe de debian :
CONTAINER ID   NAME                   CPU %     MEM USAGE / LIMIT    MEM %     NET I/O           BLOCK I/O         PIDS
b7aafe159f58   prometheus-taaf        1.93%     156.5MiB / 9.91GiB   1.54%     25.2MB / 2MB      81MB / 22.1MB     10
29b7d0c7124f   adminer-taaf           0.01%     29.23MiB / 9.91GiB   0.29%     1.9kB / 126B      20.5MB / 0B       1
fa8553201020   postgres-exporter-af   3.16%     23.32MiB / 9.91GiB   0.23%     15.7MB / 7.57MB   12.5MB / 0B       7
ee754a462edb   grafana-alloy          0.63%     262.8MiB / 9.91GiB   2.59%     157kB / 2.13MB    209MB / 455kB     10
cf3b9e3f4ecc   grafana-taaf           0.74%     329.3MiB / 9.91GiB   3.25%     7.29MB / 20.4MB   265MB / 4.07MB    16
705a57813f01   cadvisor-taaf          6.05%     65.63MiB / 9.91GiB   0.65%     208kB / 5.67MB    31.9MB / 0B       12
2d95f2e8548a   node-exporter-taaf     0.00%     20.2MiB / 9.91GiB    0.20%     0B / 0B           13.8MB / 0B       6
99ede0b53959   postgres-af            5.51%     56.21MiB / 9.91GiB   0.55%     3.19MB / 15.4MB   34.4MB / 9.09MB   8
8edf33a5bf18   loki-taaf              2.14%     155.9MiB / 9.91GiB   1.54%     189kB / 4.66MB    95.1MB / 0B       9
```

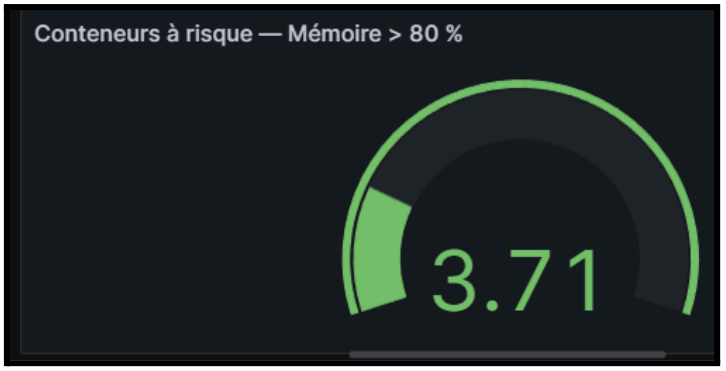
Ensuite container_memory_usage_bytes est la mémoire actuellement utilisée par chaque conteneur

On divise par la valeur limite mémoire (9.91 GiB) observée avec docker stats

On multiplie par 100 pour obtenir un pourcentage

Le and (... > 80) permet de ne garder que les conteneurs dépassant 80 %

Pendant le TP j'ai changé la valeur 80 temporairement et remplacée par 3 pour valider le bon fonctionnement du filtrage, on remarque que dans le panel il y a bien une valeurs d'un conteneurs qui est au dessus de 3 pour-cent.



J'ai ensuite baissé la valeur à 1 pourcent pour valider le fonctionnement de la requête et je remarque qu'il y a bien plusieurs conteneurs affichés.

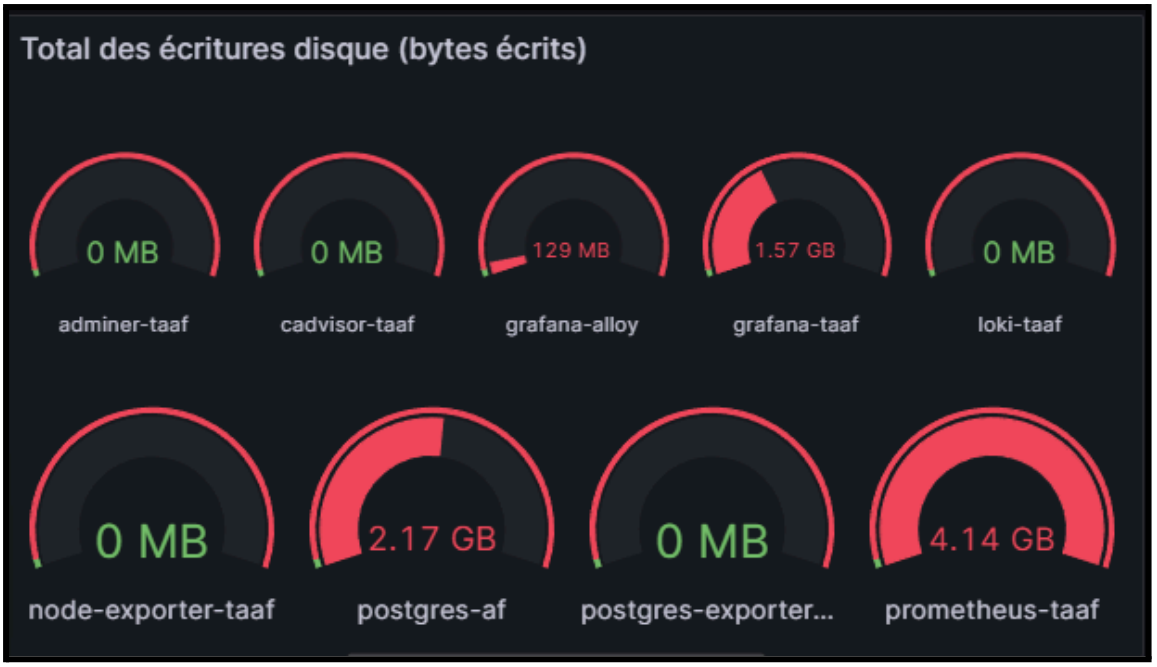
J'ai a la fin remis à 80 pourcent pour la suite du TP.

Question 9 - Performance système

Requête PromQL :

```
sum by (name) (
  rate(container_fs_writes_bytes_total{name!="" ,
job="cadvisor"}[5m]))
```

Capture du panel fonctionnelle :



Explication :

L'activité élevée est due aux transactions PostgreSQL.

Impact : forte charge I/O sur disque et donc ça provoque un ralentissement possible de Prometheus sur le même host.

Les jauges montrent que certains conteneurs génèrent plus d'activité I/O que d'autres, notamment Postgres-af, Prometheus-taaf, et Grafana-taaf.

Le conteneur Prometheus-taaf, avec 4.14 GB d'écritures disque, est celui qui impacte le plus les performances du host, suivi de postgres-af avec 2.17 GB d'écritures.

Cette activité élevée est liée à la collecte et à l'enregistrement des données de séries temporelles et des logs dans Prometheus et des transactions dans Postgres.

Question 10 - Optimisation infrastructure

Dans les panels Grafana on a pu voir que les conteneurs comme *Prometheus* et *PostgreSQL* consomment beaucoup de CPU et de RAM, ce qui peut ralentir les autres services.

Donc définir des limites de ressources dans le fichier docker-compose.yml pour chaque conteneur afin d'éviter qu'un service prenne toute la mémoire ou le CPU.

Seuils d'alertes possible :

- ➢ CPU > 90 % pendant 5 minutes → *alerte critique*
- ➢ RAM > 80 % pendant 5 minutes → *alerte warning*

Impact :

- ➢ Meilleure stabilité globale des services.
- ➢ Aucune surcharge du host

Faisabilité :

Il faut juste ajouter les directives `mem_limit` et `cpus` dans le fichier Docker Compose.

Ensuite, les graphiques ont montré que *Prometheus* et *PostgreSQL* génèrent de fortes écritures disque et cette activité I/O peut créer des ralentissements lorsqu'ils partagent le même disque.

Proposition :

- ➢ Placer *Prometheus* et *PostgreSQL* sur **des volumes séparés** ou **des disques différents**.

Seuils d'alerte recommandés :

- ➢ I/O disque > 70 % pendant 10 minutes → *alerte warning*
- ➢ I/O disque > 90 % → *alerte critique*

Impact :

- ➢ Réduction du risque de blocage disque.
- ➢ Meilleure performance du système de supervision.

Faisabilité :

Il suffit de créer des volumes dédiés dans le `docker-compose.yml` et surveillance via cAdvisor.

Certains pics de CPU et mémoire correspondent à des tâches internes comme la compaction ou les sauvegardes.

Proposition :

Planifier les backups, et autres tâches lourdes pendant les heures creuses.

Seuils d'alerte recommandés :

- ➢ Charge CPU > 85 % en dehors des créneaux planifiés → *alerte warning*

Impact :

- ➢ Réduction de la charge du système pendant les heures d'activité.
- ➢ Supervision plus fluide et réactive

Faisabilité :

Il faut ajouter un cron ou un script planifié dans Prometheus et PostgreSQL pour lancer les tâches à heure fixe.

DE LA RÉUNION Institut Universitaire de Technologie	R502	TP 2 – Supervision des Réseaux	ALAMELOU - Rohan

TP 2 - Dashboards de Supervision TAAF

Promotion : BUT RT3 TP1

Date de rendu : 31/10/2025

DE LA RÉUNION Institut Universitaire de Technologie	R502	TP 2 – Supervision des Réseaux	ALAMELOU - Rohan

Question 1 - Données de base (Facile)

Requête SQL :

```sql
SELECT COUNT(*) AS total_communications
FROM communications_satellite;
```

Explication :

La base contient 198 lignes, on remarque la bonne synchronisation entre Grafana et PostgreSQL. De plus j'ai utilisé La fonction `COUNT(*)` retourne le nombre total d'enregistrements de la table.

Capture du panel fonctionnelle :

Question 2 - Filtrage simple

Requête SQL :

```sql
SELECT COUNT(*) AS nb_reussies_7j
FROM communications_satellite
WHERE statut_transmission = 'Réussie'
  AND timestamp_comm >= CURRENT_DATE - INTERVAL '7 days';
```

Résultat : 17 communications réussies sur les 7 derniers jours.

Explication :

Cette requête filtre les enregistrements dont le statut est “Réussie” et dont la date se situe dans les 7 derniers jours grâce au filtre “WHERE” combiné à une condition temporelle.

Capture du dashboard fonctionnelle :

Question 3 - Personnel actuel

Requête SQL :

```
SELECT COUNT(*) AS nb_present
FROM personnel_base
WHERE statut = 'Présent'
  AND (date_depart_prevue IS NULL OR date_depart_prevue > NOW());
```

Résultat : 5 personnes présentes

Explication :

Cette requête compte le personnel dont le champ statut est « Présent ». La condition sur date_depart_prevue exclut les membres déjà partis.

Capture du panel fonctionnelle :

Question 4 - Calculs de moyenne

Requête SQL :

```sql
SELECT ROUND(AVG(qualite_signal), 2) AS qualite_moyenne
FROM communications_satellite
WHERE station_receptrice = 'Toulouse'
  AND timestamp_comm >= NOW() - INTERVAL '30 days';
```

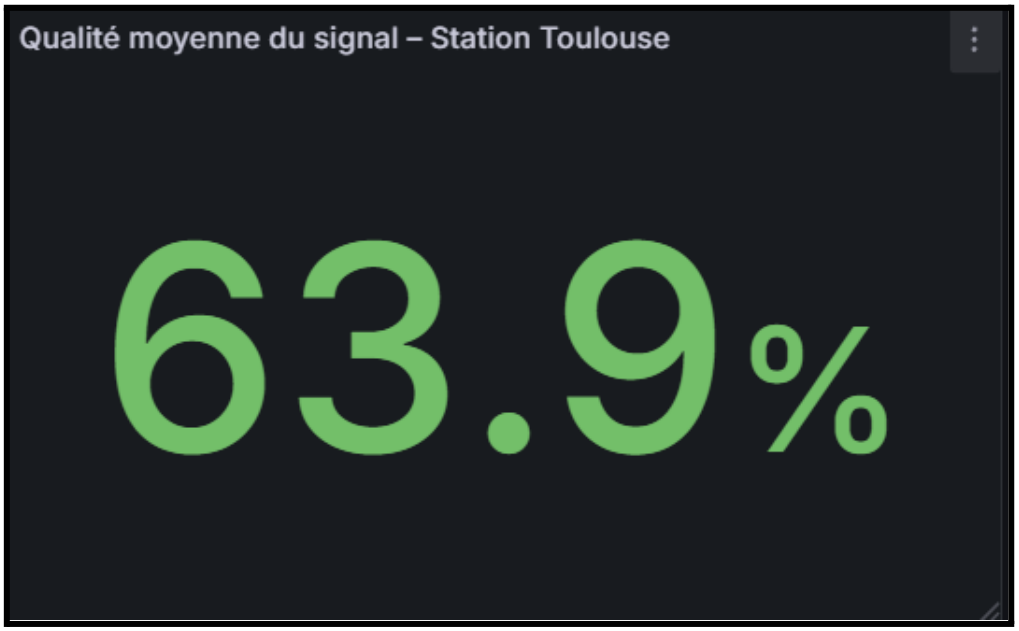
Résultat : 63,9 %.

Explication :

La fonction “AVG()” va calculer la moyenne de la colonne qualite_signal.

Le filtre station_receptrice = “Toulouse” cible la station concernée et la période de 30 jours est fixée par l’intervalle temporel.

Capture du panel fonctionnelle :

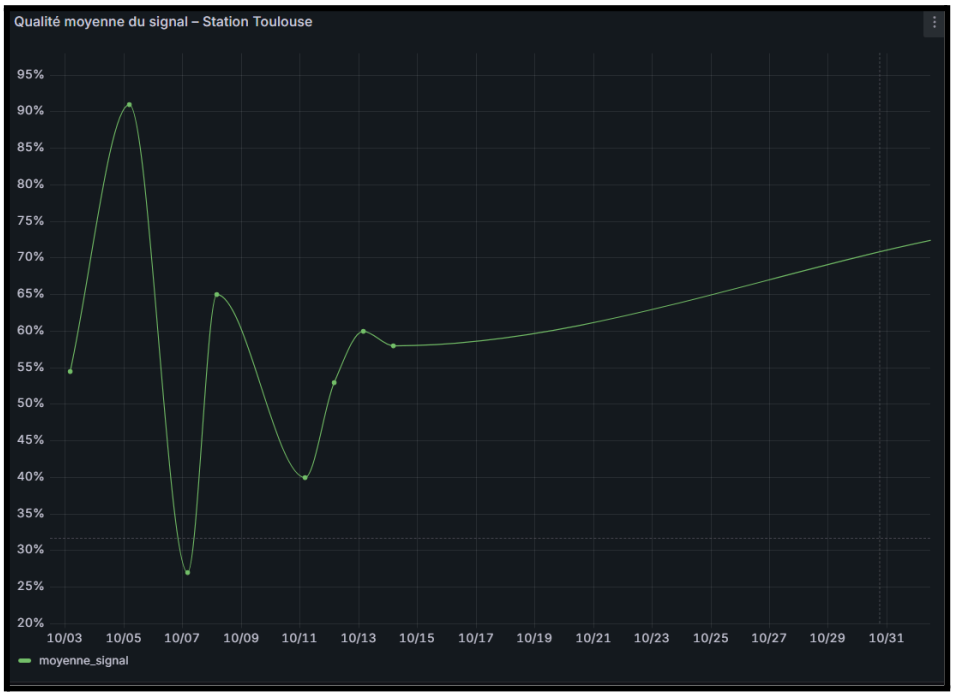


J’ai ensuite créé un panel où l'on peut voir la qualité moyenne du signal de Toulouse dans le temps.

Requête SQL :

```sql
SELECT
  DATE_TRUNC('day', timestamp_comm) AS jour,
  ROUND(AVG(qualite_signal), 2) AS moyenne_signal
FROM communications_satellite
WHERE station_receptrice = 'Toulouse'
  AND timestamp_comm >= NOW() - INTERVAL '30 days'
GROUP BY jour
ORDER BY jour;
```

Capture du panel fonctionnelle :



Question 5 - Analyse par type

Requête SQL :

```sql
SELECT type_communication,
       SUM(volume_donnees_mb) AS total_volume_mb
FROM communications_satellite
GROUP BY type_communication
ORDER BY total_volume_mb DESC
LIMIT 3;
```

Résultat :

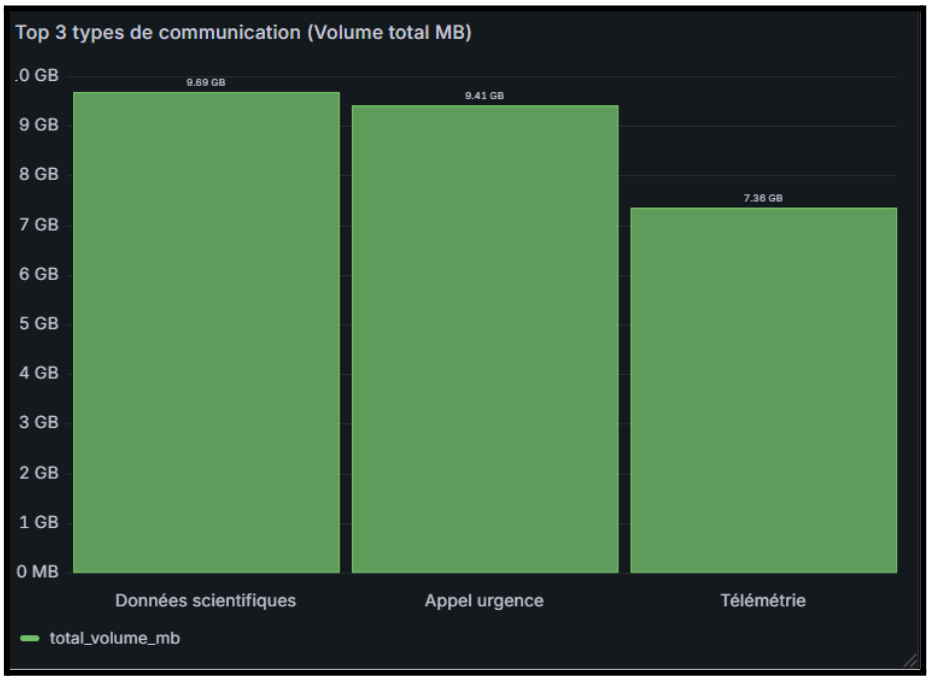
Type de communication	Volume total
Données scientifiques	9,69 GB
Appel d'urgence	9,41 GB
Télémétrie	7,36 GB

Explication :

"SUM()" additionne le volume de données transférées pour chaque type de communication.

"GROUP BY" regroupe les valeurs identiques et "ORDER BY DESC" classe les résultats du plus grand au plus petit volume.

Capture du panel fonctionnelle :



Question 6 - Analyse temporelle

Requête SQL :

```
SELECT EXTRACT(HOUR FROM timestamp_comm)::int AS heure,
       COUNT(*) AS nb_communications
FROM communications_satellite
GROUP BY heure
ORDER BY nb_communications DESC;
```

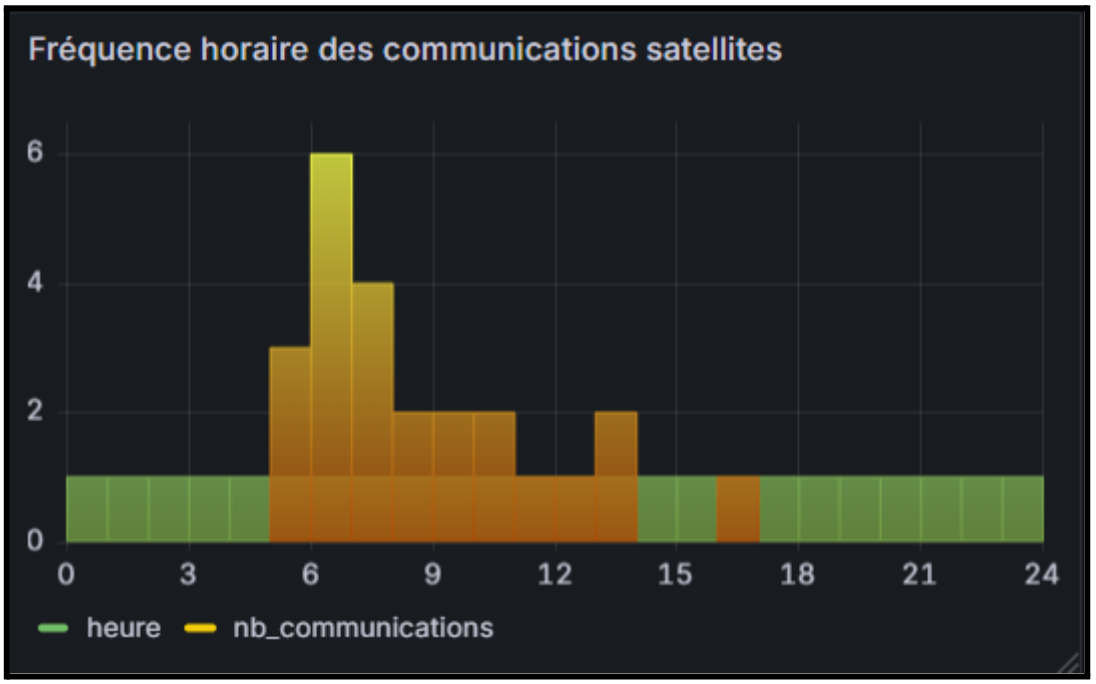
Résultat : On a le nombre de connexions par heure dans la journée.

Explication :

“EXTRACT(HOUR FROM timestamp_comm)” isole l’heure de chaque communication.

Le regroupement par heure (GROUP BY) permet d'obtenir la charge horaire.

Capture du panel fonctionnelle :



On a un pic d'activité entre 6h et 9h.

La majorité des transmissions sont effectuées en début de matinée, probablement lors de la fenêtre de synchronisation quotidienne avec la métropole.

Ce graphique peut aider à planifier les créneaux de maintenance ou de tests par exemple.

Question 7 - Analyse personnel longue mission

Requête SQL :

```
SELECT fonction, COUNT(*) AS nb_personnes
FROM personnel_base
WHERE (NOW()::date - date_arrivee::date) > 120
```

```
GROUP BY fonction
ORDER BY nb_personnes DESC;
```

Explication :

Le calcul "(NOW()::date - date_arrivee::date)" me donne la durée de mission en jours. Le filtre > 60 ou > 120 m'a permis de sélectionner les missions prolongées.

Résultat : On a tout le personnel listé dépasse 4 mois de mission.

Capture du panel fonctionnelle :

Personnel en mission prolongée(> 4 mois)

fonction	nb_personnes
Chef de district	2
Technicien télécoms	1
Géologue	1
Responsable logistique	1
Photographe scientifique	1
Technicien maintenance	1
Glaciologue	1
Vétérinaire faune	1

Ensuite , les longues périodes en mission peuvent engendrer des risques par exemple, la fatigue, l'isolement, la perte de motivation. On peut proposer une rotation toutes les 12 semaines.

Question 8 - Analyse des équipements critiques

Requête SQL :

```
SELECT nom_equipement, type_equipement, batiment, statut,
       NOW() - derniere_verification AS duree_depuis_verif
FROM equipements_critiques
WHERE statut = 'En panne'
  AND derniere_verification < NOW() - INTERVAL '48 hours'
ORDER BY duree_depuis_verif DESC;
```

Explication :
On sélectionne les équipements en panne et dont la dernière vérification date de plus de 48 heures.
La différence “NOW() - derniere_verification” fournit la durée écoulée.

Résultat : il y a deux radars météo en panne depuis la dernière vérification faite il y a 294 jours, on sait que les deux dans la tour météo.

Capture du panel fonctionnelle :

Équipements critiques en panne > 48 heures

nom_equipement	type_equipement	batiment	statut	duree_depuis_verif
Radar météo	Météorologie	Tour météo	En panne	294 days 20:01:52.91
Radar météo	Météorologie	Tour météo	En panne	294 days 20:01:52.91

Ces équipements sont importants pour la sécurité des communications.

Je peux proposer cette ordre de priorités sur les tâches qu’il y a faire :

Priorité 1 : réparation immédiate et urgente des radars météo.
Priorité 2 : contrôle global du parc afin d’éviter d’autres défaillances prolongées.

Question 9 - Dashboard d'alertes

Requête SQL :

```sql
SELECT 'Personnel mission > 90j' AS categorie_alerte, COUNT(*) AS nb
FROM personnel_base
WHERE (NOW()::date - date_arrivee::date) > 90
UNION ALL
SELECT 'Comms échouées > 24h', COUNT(*)
FROM communications_satellite
WHERE statut_transmission = 'Échec'
  AND timestamp_comm < NOW() - INTERVAL '24 hours'
UNION ALL
SELECT 'Équipements panne > 48h', COUNT(*)
FROM equipements_critiques
WHERE statut = 'En panne'
  AND derniere_verification < NOW() - INTERVAL '48 hours';
```

Résultat :

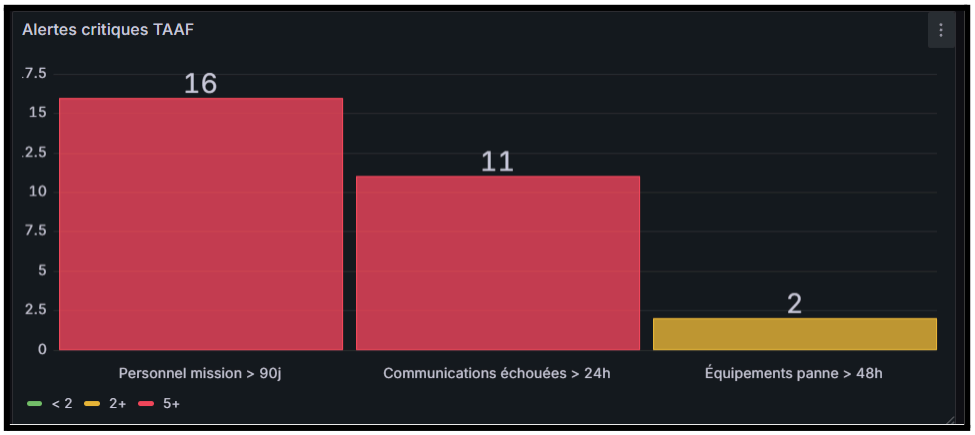
Catégorie	Nombre
Personnel > 90 j	16
Communications échouées > 24 h	11
Équipements en panne > 48 h	2

Explication :

Cette requête combine trois sous-requêtes grâce à “UNION ALL” : personnel, communications et équipements.

Chaque bloc retourne une catégorie d’alerte et un nombre d’occurrences.

Capture du panel fonctionnelle :



Le tableau d'alertes globales synthétise l'état opérationnel.

Voici les seuils que j'ai configurés pour les valeurs en vert il faut que les valeurs soient inférieures à 2, pour les valeurs orange 2–5, et pour les valeurs rouge > 5.

Avec ce panel on a une vue rapide pour les décisions de maintenance.

Question 10 - Optimisation opérationnelle

On va commencer par analyser les panels de supervision on pourra ensuite proposer une optimisation opérationnelle :

- **Communications satellites :** 198 transmissions recensées dont 17 réussies sur la dernière semaine. De plus, on sait que la qualité moyenne du signal à Toulouse s'élève à 63,9 %, sa montre une performance correcte mais pas parfaites

- **Rythme horaire :** la majorité des communications se concentrent entre 6h et 9h, ce qui traduit une surcharge horaire de la bande passante à certaines périodes.

- **Ressources humaines :** 5 personnels présents actuellement sur la base, dont plusieurs en mission prolongée (> 4 mois). On a pu voir que sa peut entraîner de la fatigue et de l'isolement.

- **Équipements critiques :** deux radars météorologiques sont en panne depuis plus de 295 jours, ce qui montre un manque de suivi des maintenances.

<u>On peut faire la corrélation entre la météo et qualité du signal :</u>

Le but est d'identifier l'influence des conditions météorologiques sur la dégradation du signal satellite pour anticiper les coupures.

Proposition technique : On peut relier la table conditions_meteo à communications_satellite dans une requête :

KPI proposés :

- Corrélation entre signal/météo
- Nombre moyen de coupures liées à la météo par semaine

Impact opérationnel :

- Anticipation des perturbations à cause de la météo
- Planification des transmissions en période correcte.

Faisabilité :

- On a les données météo déjà présentes dans la base.
- Il faut ajouter d'un panel "Signal et Météo" en courbes superposée

<u>On peut aussi faire la planification des rotations :</u>

Le but est de repérer les longues missions pour éviter les dépassements de durée et leurs effets sur la performance humaine.

KPI proposés :

- Durée moyenne des missions en jours
- Taux de dépassement des missions de plus de 90 jours
- Taux de rotation par mois du personnel

Impact opérationnel :

- Meilleure répartition des tâches avec le personnelle
- On évite les risques sur l'isolement et à la fatigue du personnelle

Faisabilité :

- Données déjà disponibles dans la table personnel base
- L'ajout direct d'un panel RH automatisé

<u>On peut mettre en place la surveillance des équipements critiques afin d'anticiper les pannes avant qu'elles ne deviennent bloquantes :</u>

L'objectif est de suivre la durée depuis la dernière maintenance ou vérification, afin d'organiser les interventions techniques en temps voulu.

KPI proposés :

- Temps de fonctionnement avant maintenance
- Nombre d'équipements en maintenance retardée avec plus de 90 jours sans maintenance.
- Taux d'équipements vérifiés dans les délais.

Impact opérationnel :

- Réduction significative du nombre de pannes prolongées
- Optimisation des déplacements techniques et de la planification logistique des maintenances.

Faisabilité :

- Les informations nécessaires (derniere_verification, prochaine_maintenance, statut) sont déjà présentes dans la table equipements_critiques.

- Un panel Grafana peut être créé pour afficher les équipements dépassant un seuil de 180 jours depuis leur dernière vérification.

DE LA RÉUNION Institut Universitaire de Technologie	R502	TP 3 – Supervision des Réseaux	ALAMELOU - Rohan

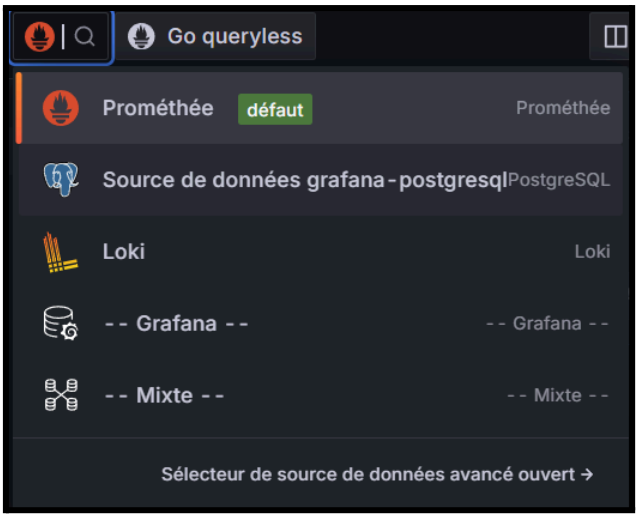
TP 3 - Loki Integration : Intégration Logs & Métriques TAAF

Promotion : BUT RT3 TP1

Date de rendu : 31/10/2025

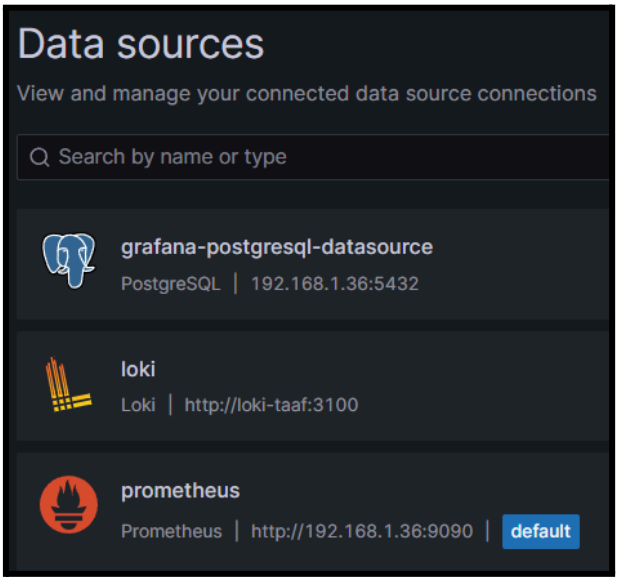
Question 1 - Configuration de base

Dans un premier temps, dans mon Grafana unifié on peut voir qu'il y a que 3 sources de données.



On peut voir dans mes data-sources avec leurs ip correspondantes :

- ➢ Postgres : 5432
- ➢ Loki : 3100
- ➢ Prometheus : 9090

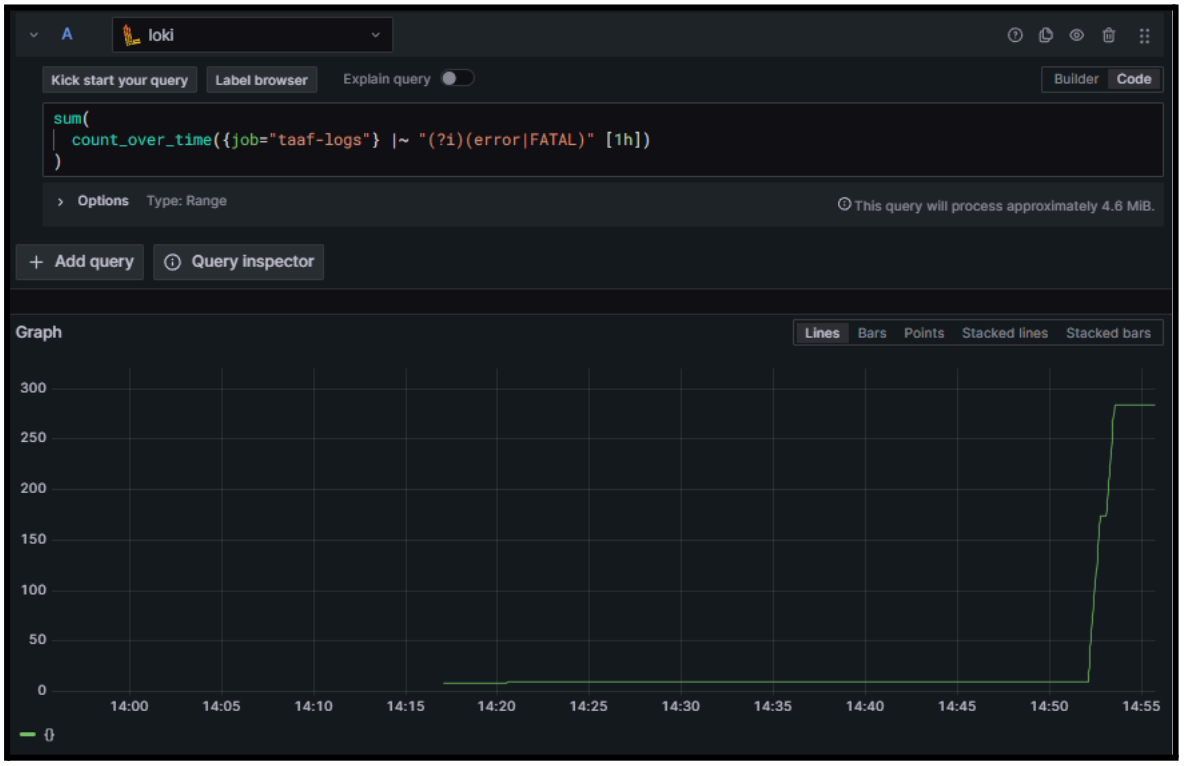


Question 2 - Requête LogQL simple

Requêtes LogQL :

```
sum(count_over_time({job="taaf-logs",
filename=~".*postgres-af.*"} |~ "(?i)(ERROR|FATAL)" [1h]))
```

Capture du panel explorer fonctionnelle :

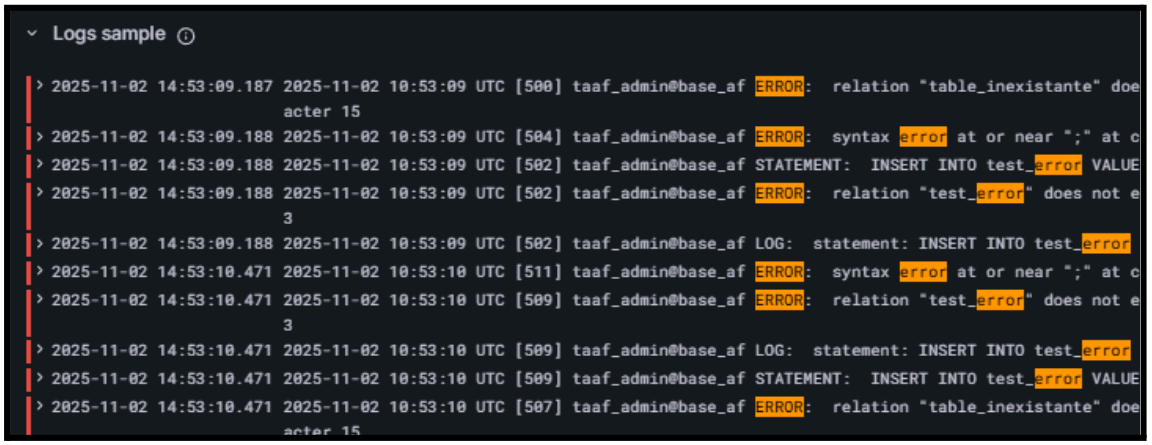


Afin de voir un nombre conséquent de logs d'erreurs

J'ai dans un premier temps simulé des erreurs sur la base postgres avec des mauvaises requêtes.

```
psql -U taaf_admin -d base_af -c "SELECT * FROM
table_inexistante;" >/dev/null 2>&1;
psql -U taaf_admin -d base_af -c "SELECT * FROM;" >/dev/null
2>&1;
```

J’ai pu ensuite voir les erreurs dans les logs :



Les logs “ERROR” proviennent des erreurs SQL simulées sur la base PostgreSQL.

J’ai pu valider la bonne intégration de Loki vers Alloy puis Alloy vers Docker.

J’ai donc en tout 283 logs d’erreur

Question 3 - Corrélation simple

Afin de pouvoir faire la corrélation entre un pic de CPU, j'ai créer deux dashboard dans Explorer afin de regarder les erreurs logs et le comportement du CPU au même moment

Requêtes utilisés pour voir CPU utilisés du conteneurs postgres (graphique vert) :

```
rate(container_cpu_usage_seconds_total{name="postgres-af"}[5m]) * 100
```

Requêtes utilisés pour voir les logs d'erreur au même moment que dans l'évolution du CPU (graphique rouge) :

```
{job="taaf-logs"} |~ "(?i)error"
```

Ce qui nous a permis d'avoir ce graphe corréler avec les log d'erreurs en dessous :

Explication :

Sur la courbe verte on voit plusieurs pics CPU, notamment aux vers de 02h00 et 10h00, atteignant environ 70 % d'utilisation.

Au même moment, la courbe rouge montre des pics d'erreurs dans les logs, avec 12 erreurs totales détectées sur la période.

La corrélation entre hausse du CPU et l'apparition d'erreurs indique une corrélation montre que le service PostgreSQL subit une surcharge lorsqu'il rencontre des erreurs d'exécution, comme des divisions par zéro ou des connexions interrompues.

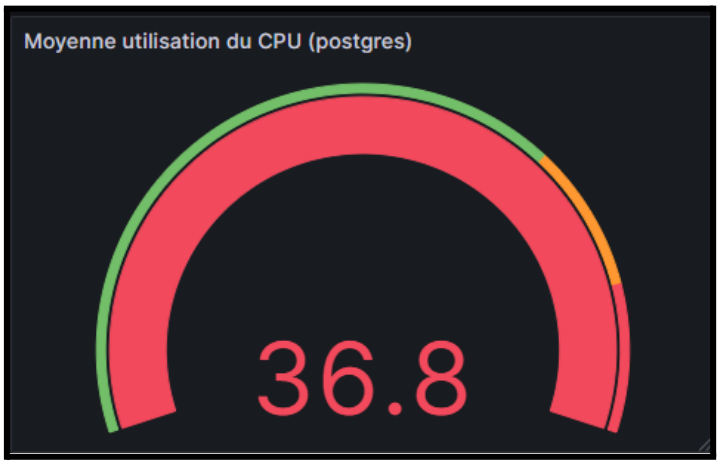
Question 4 - Requête Prometheus

Requêtes PromQL :

```
avg_over_time(container_memory_usage_bytes{name="postgres-af"}[30m]) / (1024*1024)
```

Cette requête va permettre de calculer la moyenne de l'utilisation du CPU sur les 30 dernières minute grâce à "avg_over_time"

Capture du panel fonctionnelle :



Le panel ci-dessus affiche la moyenne d'utilisation CPU du conteneur PostgreSQL qui est ici de 36.8 pourcent.

Question 5 - Analyse de patterns

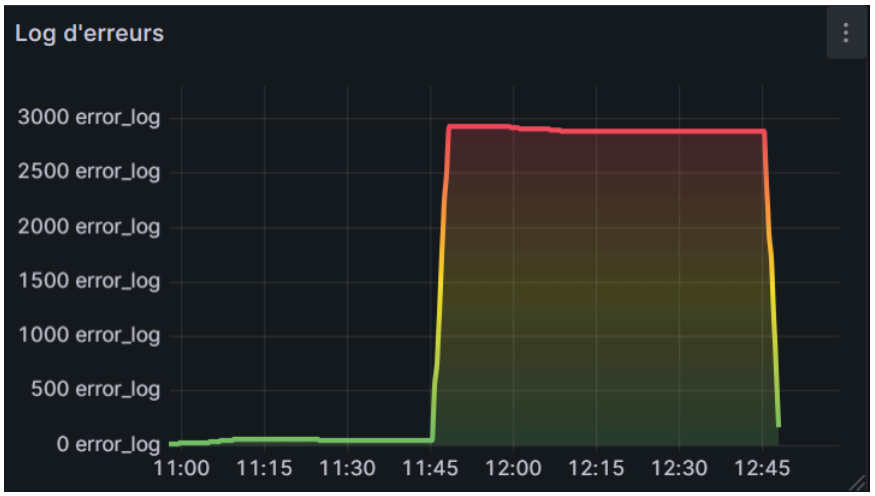
J'ai tout d'abord créé des logs d'erreurs de manière aléatoire afin de créer du mouvement sur la base de données.

Voici le scripts que j'ai utilisé pour simuler les erreurs de manière aléatoire

```
#!/bin/bash
LOG="/var/log/remote/TAAF-DOCKER-001/docker/postgres-af.log"

while true; do
  # Erreurs plus ou moins espacées
  ERRORS=$((RANDOM % 3 + 1))
  for i in $(seq 1 $ERRORS); do
    echo "$(date '+%Y-%m-%d %H:%M:%S') ERROR: requête échouée
$i - base_af" >> "$LOG"
  done
  # Pause aléatoire entre 1 et 5 minutes
  sleep $((RANDOM % 240 + 60))
done
```

Capture du panel fonctionnelle :



On remarque que le nombre de logs d'erreurs est croissant au début, en effet ça correspond au script que j'ai lancé, ensuite on remarque qu'il y a un gros pic, ce

gros pic correspond lorsque j'ai effectué mes tests sur le CPU de postgres, j'ai surcharger le conteneur postgres.

Question 6 - Dashboard hybride

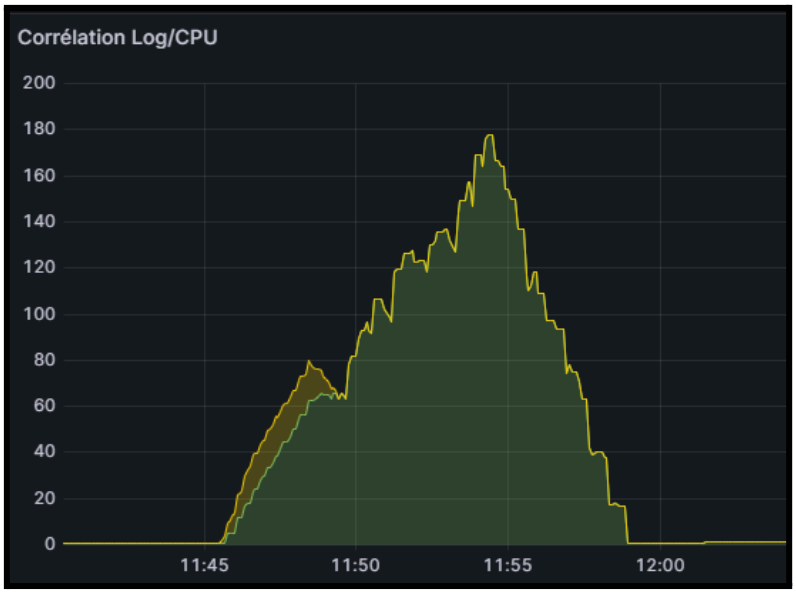
Requête PromQL pour le CPU (vert) :

```
sum by (name)
(rate(container_cpu_usage_seconds_total{name="postgres-af"}[5m]))
* 100
```

Requête LogQL pour les logs (jaune) :

```
sum(rate({job="taaf-logs"} |~ "(?i)error" [1m]))
```

Capture du panel fonctionnelle :



Explication :

On voit qu'entre 11h45 et 12h00, on voit une forte montée du CPU sur le conteneur PostgreSQL, atteignant environ 180 % et au même moment une augmentation simultanée du taux d'erreurs dans les logs .

On peut faire la corrélation directe sa montre qu'à chaque pic du CPU, PostgreSQL génère un grand nombre d'erreurs SQL.

Cela traduit une surcharge causée par des requêtes incorrectes ou répétitives, typiquement un test de division par zéro exécuté en boucle

Question 7 - Configuration d'alerte (Moyen)

J'ai dans un premier temps rechercher le fichier des alertes donc alert-rules.yml, puis j'ai modifier le fichier en ajoutant l'alerte qu'il me faut.

Voici le bloc que j'ai ajouter dans /home/debian/monitoring/config/prometheus/alert_rules.yml :

```
- alert: TAAFHighCPU
  expr: sum by (name)
(rate(container_cpu_usage_seconds_total{name="postgres-af"}[5m]))
* 100 > 80
  for: 2m
  labels:
    severity: warning
    mission: taaf
    base: alfred-faure
  annotations:
    summary: "CPU élevé sur {{ $labels.name }}"
    description: "Le conteneur {{ $labels.name }} dépasse 80% CPU
depuis 2 minutes."
```

J'ai ensuite surcharger mon CPU sur le conteneur PostgresSQL avec la commande suivante :

```
docker exec -it postgres-af bash -c "while true; do psql -U
taaf_admin -d base_af -c 'SELECT 1/0;' >/dev/null 2>&1; done"
```

J’ai ensuite patienter puis dans Grafana et dans alertes j’ai pu voir mon alerte en rouge avec “firing”, mon alarme fonctionne

Question 8 - Investigation d'incident

Requêtes PromQL pour l’usage du CPU (vert) :

```
sum by (name) (
  rate(container_cpu_usage_seconds_total{name="postgres-af"}[5m])
) * 100
```

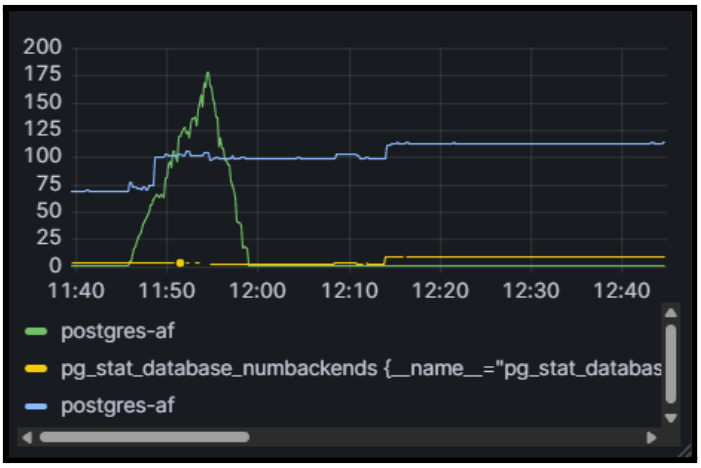
Requêtes PromQL nombre de connexion active à la base (bleu) :

```
pg_stat_database_numbackends{datname="base_af"}
```

Requêtes PromQL pour voir la mémoire utilisé par le conteneur postgres (jaune) :

```
container_memory_usage_bytes{name="postgres-af"} / 1024 / 102
```

Capture du panel fonctionnelle :



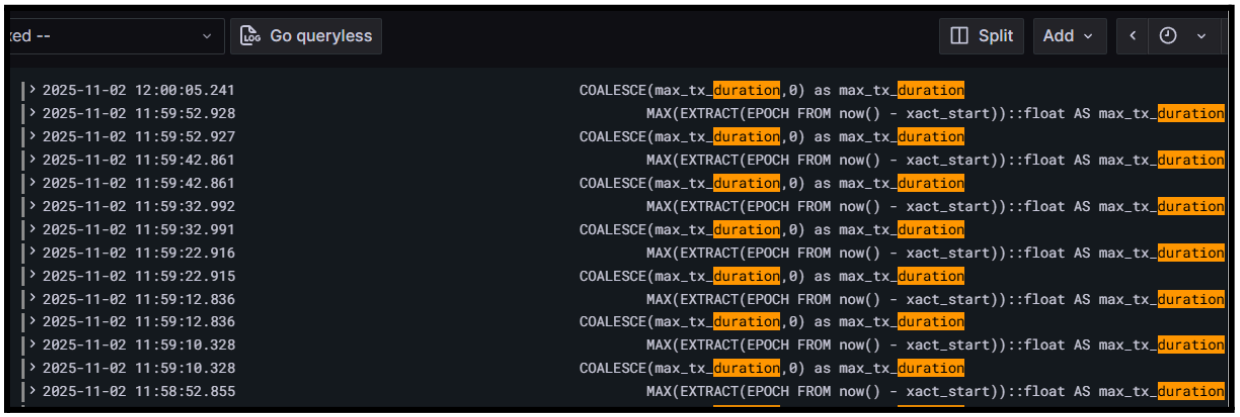
On remarque entre 11h45 et 12h00, on observe un pic CPU dépassant 170 %, avec une hausse des connexions actives.

On voit donc une saturation temporaire du conteneur postgreSQL

J'ai effectué a ce moment des tests aussi de ralentissement du service postgres.

J'ai donc fait beaucoup de requêtes avec **"SELECT pg_sleep(5);"** dans le conteneurs pour simuler une latence, un ralentissement.

On remarque au même moment dans les logs pendant mes tests qu'il y a des erreurs correspondant à un ralentissement du service.



On remarque qu'un comportement anormal a été observé entre 11h45 et 12h00 : plusieurs erreurs SQL affichaient une durée d'exécution élevée.

De plus dans le graphique on remarque un pic de CPU sur le conteneur postgres à 170 % puis une hausse des connexions actives et une augmentation de la durée des transactions (max_tx_duration) visible dans les logs.

J'ai ensuite regarder dans explore les logs d'erreur et mis en corrélation avec avec l'utilisation du CPU :

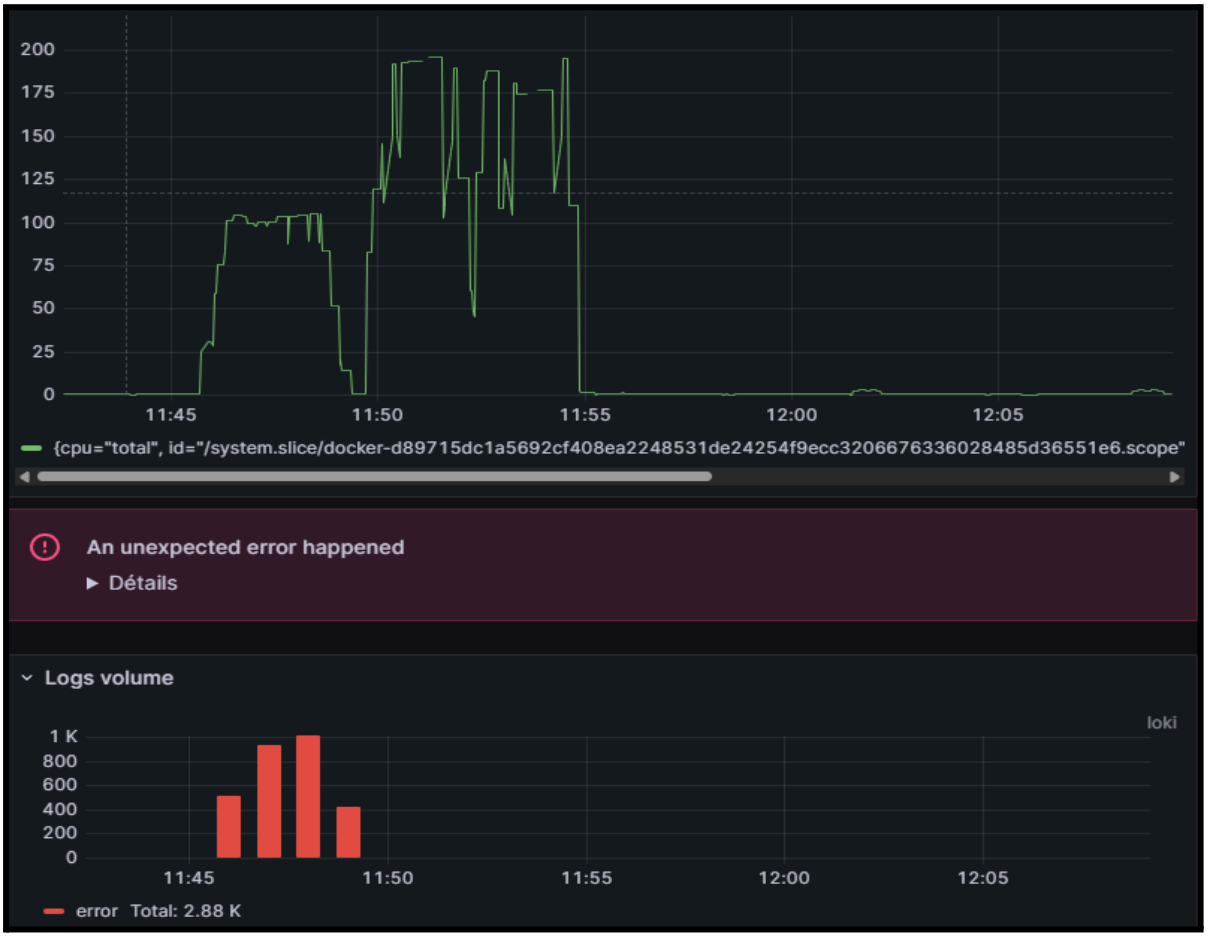
Requête PromQL pour l'utilisation du CPU (vert) :

```
rate(container_cpu_usage_seconds_total{name="postgres-af"}[1m])
* 100
```

Requête LogQL pour la vision des log (rouge) :

```
{job="taaf-logs", filename=~".*postgres-af.*"} |~ "(?i)ERROR"
|= "division"
```

Capture du dashboard fonctionnelle :



En observant l'image en croisant les métriques et des logs on voit que :

- ➢ Les requêtes erronées répétées ont provoqué une surcharge CPU.
- ➢ Cela a entraîné une latence élevée et des transactions incomplètes.

Question 9 - Debug et simulation

J'ai créé une panne en stoppant le conteneur postgres, cela simule bien une panne du conteneur postgres.

J'ai ensuite regardé les logs d'erreur pour voir ce qu'il se passe quand le service s'arrête, j'ai donc pu voir le type d'erreur que ça fait.

Requêtes LogQL :

```
{job="taaf-logs", filename=~".*postgres-af.*"} |= "terminating connection"
```

J'ai ensuite pu regarder les logs au moment ou j'ai effectué la panne c'est à dire à 14h17 et à 14h20.

Ensuite j'ai pu effectuer une requête Prometheus PromQL pour regarder comment les logs réagissent à ce moment c'est-à -dire au moment de la panne.

Requête PromQL :

```
pg_up{job="postgres-af"}
```

On remarque qu'en effet les logs baissent vers 14h17 lors de mon premier test, puis dans mon deuxième test, on remarque que les logs se coupent nettement ce qui traduit en effet la panne.

Question 10 - Synthèse et cas d'usage TAAF

Le monitoring traditionnel qui est basé que sur les métriques Prometheus, permet de mesurer l'état d'un service (CPU, mémoire, disponibilité).

Par contre, il ne donne pas la cause des problèmes que l'on va avoir : quand un service consomme trop ou tombe en panne on ne pourra pas savoir pourquoi sans regarder soi même manuellement

Alors que l'approche hybride avec métriques et logs, utilisée ici avec Prometheus, Loki et Grafana Alloy, permet de détecter les symptômes via les métriques, mais aussi d'identifier la cause grâce aux logs collectés en parallèle.

Donc on obtient une vision complète du comportement d'un service, c'est important et très utile.

Cas d'usage 1 – Surcharge de la base PostgreSQL Alfred Faure

Scénario :

Une série de requêtes SQL erronées provoque une montée CPU et un ralentissement global du service PostgreSQL.

On surveille les métriques qui sont :

```
rate(container_cpu_usage_seconds_total{name="postgres-af"}[5m]) *
100
```

```
pg_stat_database_numbackends
```

On analyse les logs avec :

```
{job="taaf-logs", filename=~".*postgres-af.*"} |~
"(?i)(error|fatal)"
```

Alerte associée :

```
- alert: HighCPUUsage
  expr:
rate(container_cpu_usage_seconds_total{name="postgres-af"}[5m]) >
0.8
  for: 2m
  labels:
    severity: warning
```

Résultat :

Les logs et les métriques montrent simultanément une surcharge CPU et des erreurs SQL.

On peut faire la corrélation permet d'identifier immédiatement la cause du ralentissement.

Cas d'usage 2 – Interruption de service (panne conteneur)

Scénario :
Le conteneur postgres-af s'arrête brutalement, simulant une panne de service.

Métriques surveillées :

```
up{job="cadvisor", name="postgres-af"}
```

Logs analysés :

```
{job="taaf-logs", filename=~".*postgres-af.*"} |= "terminating
connection"
```

Alerte associée :

```
- alert: ContainerDown
  expr: up == 0
  for: 30s
  labels:
    severity: critical
```

Résultat :

Prometheus détecte la panne et Loki montre la cause et grafana envoie l'alerte

Cas d'usage 3 – Espace disque saturé sur la base Alfred Faure

Scénario :

Au fil du temps, les fichiers journaux et les sauvegardes de la base PostgreSQL s'accumulent.

L'espace disque du serveur devient presque plein donc ça provoque des lenteurs et des erreurs lors de l'écriture dans la base.

Métriques surveillées par Prometheus :

```
node_filesystem_avail_bytes{instance=~".+",
mountpoint="/var/lib/docker/volumes/postgres_af_data/_data"} /
node_filesystem_size_bytes{instance=~".+",
mountpoint="/var/lib/docker/volumes/postgres_af_data/_data"} * 100
```

Mesure le pourcentage d'espace disque disponible sur le volume PostgreSQL.

Logs analysés par Loki :

```
{job="taaf-logs", filename=~".*postgres-af.*"} |~ "(?i)(disk
full|no space left|write failed)"
```

Ils repèrent les messages d'erreur dans les logs PostgreSQL signalant un manque d'espace.

Alerte associée :

```
- alert: LowDiskSpace
  expr: (node_filesystem_avail_bytes / node_filesystem_size_bytes)
< 0.1
  for: 5m
  labels:
    severity: warning
    mission: taaf
  annotations:
    summary: "Espace disque faible sur PostgreSQL"
    description: "Moins de 10 % d'espace libre sur le volume
PostgreSQL Alfred Faure."
```

Résultat :

L'alerte est déclenchée lorsque le volume tombe en dessous de 10 % d'espace libre. Les logs confirment ensuite des erreurs qui montrent qu'il n'y a plus de place. Cette corrélation permet de prévenir une panne avant qu'elle ne bloque complètement la base de données.

Capture du dashboard finale

DE LA RÉUNION Institut Universitaire de Technologie	**R502**	TP 4 – Supervision des Réseaux	**ALAMELOU - Rohan**

TP 4 - Alerting : Configuration d'alertes TAAF avec webhooks

Promotion : BUT RT3 TP1

Date de rendu : 31/10/2025

Question 1 - Configuration de base

J'ai effectué cette commande pour trouver le nombre d'alertes qu'il y a dans mon fichier alert-rules.yml, j'ai donc 8 alertes configurés :

```
 /home/d/monitoring  main !5 ?19
 grep -c "alert:" ./config/prometheus/alert_rules.yml
8
```

J'ai ensuite effectué cette commande pour lister les noms, voici les noms affichés :

```
 /home/d/monitoring  main !5 ?19
 grep  "name:" config/prometheus/alert_rules.yml
  - name: taaf_postgresql_alerts
  - name: taaf_infrastructure_alerts
```

Question 2 - Webhook simple

J'ai choisi discord pour configurer mon webhook, voici ma configuration discord :

J'ai donc pris le lien pour le rentrer dans prometheus.yml et dans docker-compose.yml.

J'ai dans un premier temps monter le conteneurs de alertmanager qui dois communiquer avec le conteneur de alertmanager-discord qui va envoyer les messages vers discord.

Voici les deux bloc qu'il a fallu rajouter dans docker-compose.yml :

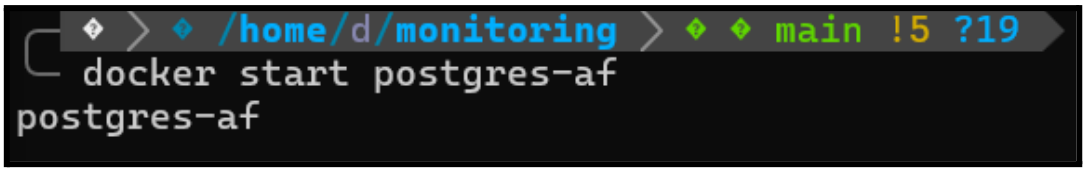
```
# ALERTMANAGER - TAAF
  alertmanager:
    image: prom/alertmanager:latest
```

```
    container_name: alertmanager
    restart: unless-stopped
    ports:
      - "9093:9093"
    volumes:
      -
./alertmanager/alertmanager.yml:/etc/alertmanager/alertmanager.yml
:ro
      - ./volumes/alertmanager_data:/alertmanager
    command:
      - '--config.file=/etc/alertmanager/alertmanager.yml'
      - '--storage.path=/alertmanager'
      - '--web.external-url=http://localhost:9093'
    networks:
      - taaf-monitoring


  alertmanager-discord:
    image: benjojo/alertmanager-discord
    container_name: alertmanager-discord
    environment:
      DISCORD_WEBHOOK:
"https://discord.com/api/webhooks/1434538898725404732/etc
      USERNAME: "TAAF Monitor"
      ICON_URL:
"https://upload.wikimedia.org/wikipedia/commons/1/10/Terres_a>
    ports:
      - "9094:9094"
    restart: unless-stopped
    networks:
      - taaf-monitoring
```

Question 3 - Test d'alerte

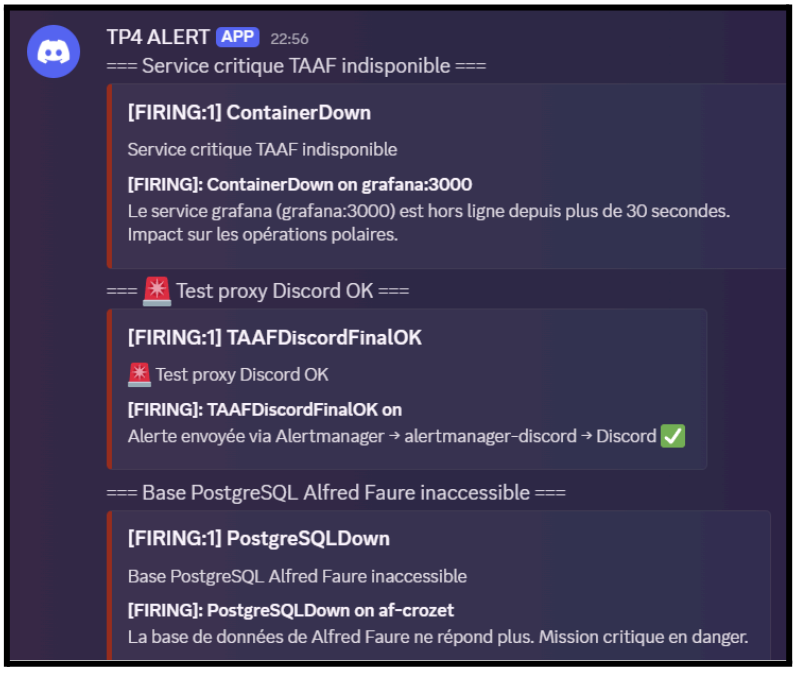
J'ai pu effectuer ensuite un test, j'ai arreter la base postgres, pour tester une alerte 'ContainerDown"



J'ai donc appelé cette commande en particulier, j'ai aussi arrêté grafana lors de mes tests et j'ai aussi envoyé un message manuels pour tester je les ai donc reçu.

```
- alert: PostgreSQLDown
        expr: pg_up == 0
        for: 1m
        labels:
          severity: critical
          mission: taaf
        annotations:
          summary: "Base PostgreSQL {{ $labels.base_taaf }}
inaccessible"
          description: "La base de données de {{ $labels.base_taaf }} ne
répond plus. Mission critique en danger."
```

Capture des alertes :



Question 4 - Seuils d'alerte

Le seuil de 85 % d'utilisation mémoire est choisi en se basant sur les métriques prometteuses observées que l'on a pu voir pendant les TP et sur les bonnes pratiques d'exploitation système.

On a souvent vu que la charge mémoire moyenne des conteneurs reste généralement comprise entre 60 % et 75 %, avec quelques pointes ponctuelles lors de l'exécution de requêtes SQL ou du démarrage des services.

Le choix de 85 % permet donc de laisser une marge de sécurité de 10 à 15 % avant une saturation réelle.

À 90 % ou plus, il y a un risque de OOM c'est-à- dire qu'il y a plus assez de mémoire pour tous les processus donc des processus peuvent être automatiquement tués.

Ensuite avec le seuil à 85 %, on peut anticiper cette situation tout en évitant des alertes fréquentes à cause des pics temporaires.

Conclusion

Lors de la réalisation de ce TP, j'ai rencontré plusieurs difficultés techniques, notamment au niveau de la configuration des conteneurs et de la communication entre Alertmanager et Discord. Ces problèmes ont empêché la finalisation complète du projet et ont nécessité beaucoup de temps de diagnostic.

De plus, certaines erreurs rencontrées lors du TP3 ont eu un impact direct sur ce TP4, notamment sur la remontée et la détection correcte des alertes. Malgré ces difficultés, ce travail m'a permis de mieux comprendre la logique de l'alerting dans Prometheus et le fonctionnement des webhooks avec Alertmanager.