Rapport de TP6

Stack Open-Source d'Observabilité
(ELK + Jaeger + Prometheus)

Par : Ambre GUIA

Sommaire

QUESTIONS PRÉLIMINAIRES

Section A : Compréhension théorique

Q1. Citez les 3 piliers de l'observabilité et donnez un exemple d'outil pour chacun.

- Pilier 1 : Métriques → Outil : Prometheus
- Pilier 2 : Logs → Outil : ELK (Elasticsearch, Logstash, Kibana)
- Pilier 3 : Traces → Outil : Jaeger

Q2. Quelle est la différence fondamentale entre monitoring et observabilité ?

Le monitoring consiste à surveiller des métriques prédéfinies et à déclencher des alertes lorsque des seuils sont dépassés. Il permet de détecter qu'un problème existe.
L'observabilité va plus loin : elle permet de comprendre pourquoi un problème se produit en analysant les métriques, les logs et les traces, même pour des scénarios non anticipés.

On peut le résumer par :
« Le monitoring dit QUOI ne va pas, l'observabilité explique POURQUOI. »

Q3. Associez chaque cas d'usage à l'outil approprié :

Cas d'usage	Outil
Analyser pourquoi une requête met 5 secondes	**Jaeger**
Voir le taux de requêtes HTTP par seconde	**Prometheus**
Chercher les erreurs 500 dans les logs	**ELK**
Identifier quel microservice cause la latence	**Jaeger**

Q4. Qu'est-ce qu'un Golden Signal en SRE ? Citez les 4 types.

1. Latency (Latence)
2. Traffic (Trafic)
3. Errors (Erreurs)
4. Saturation (Saturation des ressources)

Ces signaux permettent d'évaluer rapidement la santé d'un service.

Q5. Expliquez la différence entre push et pull pour la collecte de métriques.

En mode pull, le système de monitoring (Prometheus) interroge régulièrement les services pour récupérer leurs métriques via un endpoint (ex : /metrics).

En mode push, ce sont les applications qui envoient elles-mêmes leurs métriques vers un collecteur (ex : Pushgateway), ce qui est moins courant avec Prometheus.

Partie 1 : Déploiement de la stack

Étape 1.1 : Récupération du projet

La structure du projet est la suivante :

```
tp6-observability-stack/
│
├── docker-compose.yml
├── .env.example
├── .env
├── README.md
├── QUICKSTART.md
├── init.sql
│
├── frontend/
│   ├── Dockerfile
│   ├── package.json
│   ├── server.js
│   └── public/
│       └── index.html
│
├── backend/
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── app.py
│   ├── models.py
│   ├── config.py
│   └── init_db.py
│
├── prometheus/
│   └── prometheus.yml
│
├── logstash/
│   └── pipeline.conf
│
└── grafana/
    └── provisioning/
        ├── datasources/
        │   └── datasources.yml
        └── dashboards/
            └── dashboards.yml
```

Étape 1.2 : Configuration de l'environnement

Q6. Ces valeurs sont-elles adaptées pour un environnement de production ?

Votre réponse : ☐ Oui ☒ Non

Si non, pourquoi ?

Non, ces valeurs ne sont pas adaptées à un environnement de production car elles contiennent des identifiants par défaut (user/pass, admin/admin), ce qui représente un risque de sécurité important.

De plus, certaines options comme FLASK_DEBUG doivent être désactivées en production et les ressources mémoire allouées à Elasticsearch et Logstash sont trop faibles pour une charge réelle.

Étape 1.3 : Démarrage de la stack

Q7. Quelle commande permet de vérifier que tous les conteneurs sont bien démarrés et en état "healthy" ?

Commande : `docker compose ps`

Q8. Combien de conteneurs doivent être actifs au total ?

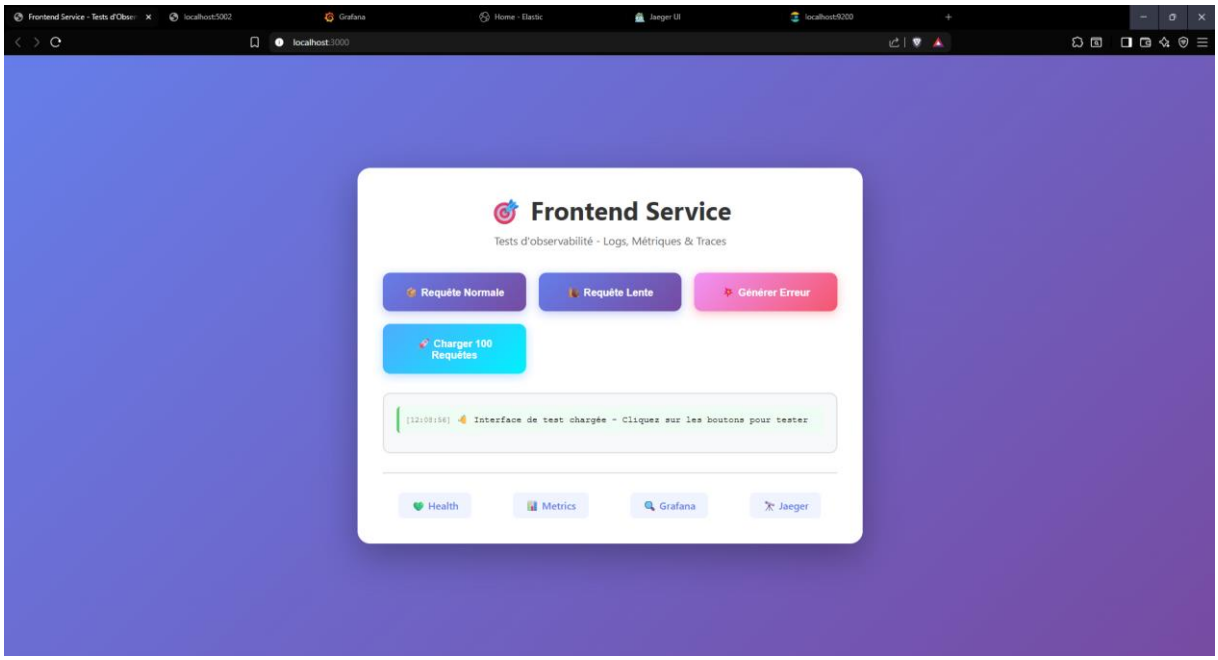
Réponse : 10 conteneurs

Étape 1.4 : Vérification des services

Complétez le tableau en accédant à chaque interface :

Service	**URL**	**Credentials**	**Status ✅/❌**
Frontend	http://localhost:3000	-	✅
Backend API	http://localhost:5002/health	-	✅
Prometheus	http://localhost:9091	-	✅
Grafana	http://localhost:3001	admin/admin	✅
Kibana	http://localhost:5601	-	✅
Jaeger UI	http://localhost:16686	-	✅

Q9. Prenez un screenshot de la page d'accueil du Frontend montrant les boutons de test.



Q10. Exécutez la commande suivante et analysez les logs :

La commande suivante est faite pour du linux comme je suis sous PowerShell et doit être modifié :

```
docker compose logs backend --tail=20 | grep "initialisé"
```

La bonne commande est :

```
docker compose logs backend --tail=20 | sls "initialis"
```

De plus, l'accent n'est pas reconnu.

La ligne indiquant que l'application fonctionne est :

backend_service | {"timestamp": "2025-12-17 11:07:28", "level": "INFO", "service": "backend", "message": "Application Flask initialis\u00e9e", "port": 5000, "database": "database:5432/products"}

Partie 2 : Métriques avec Prometheus & Grafana

Étape 2.1 : Exploration de Prometheus

*Q11. Allez dans **Status > Targets**. Combien de targets sont scrapées par Prometheus ? Listez-les.*

1. Target 1 : backend (http://backend:5000/metrics)
2. Target 2 : elasticsearch (http://elasticsearch-exporter:9114/metrics)
3. Target 3 : frontend (http://frontend:3000/metrics)
4. Target 4 : jaeger (http://jaeger:14269/metrics)
5. Target 5 : postgres (http://postgres-exporter:9187/metrics)
6. Target 6 : Prometheus (http://localhost:9090/metrics)

*Q12. Toutes les targets sont-elles en état **UP** (verte) ? Si une target est DOWN, quelle pourrait être la cause ?*

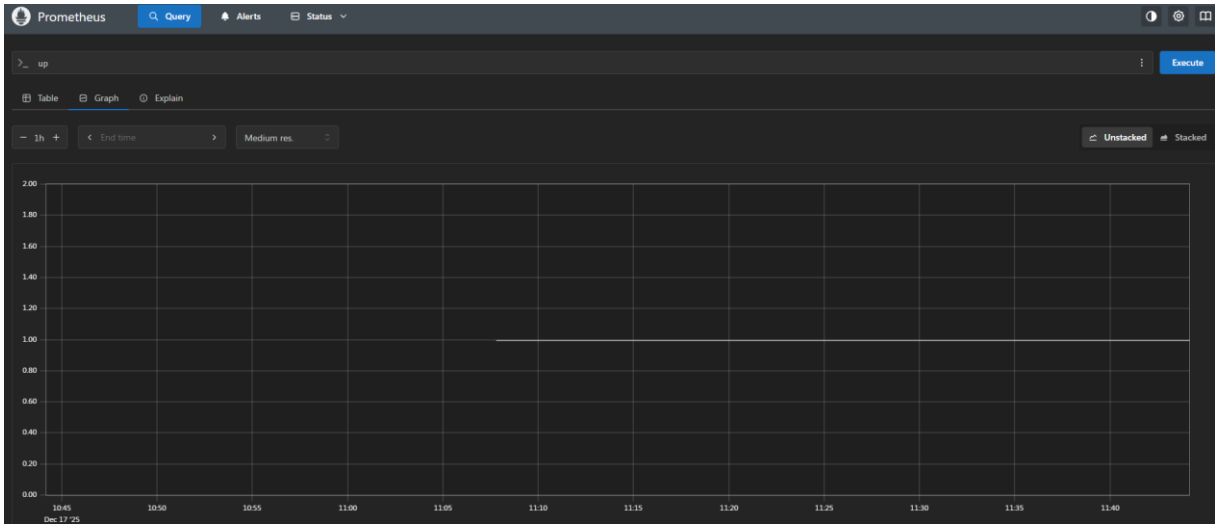
Oui, toutes les targets sont en état **UP** (vertes).

Si une target était **DOWN**, les causes possibles pourraient être :

- Le service correspondant n'est pas démarré ou a crashé.
- L'endpoint /metrics n'est pas exposé ou mal configuré.
- Un problème réseau empêche Prometheus d'atteindre le service.
- Le port ou le nom d'hôte est incorrect dans la configuration de Prometheus.

Étape 2.2 : Requêtes PromQL

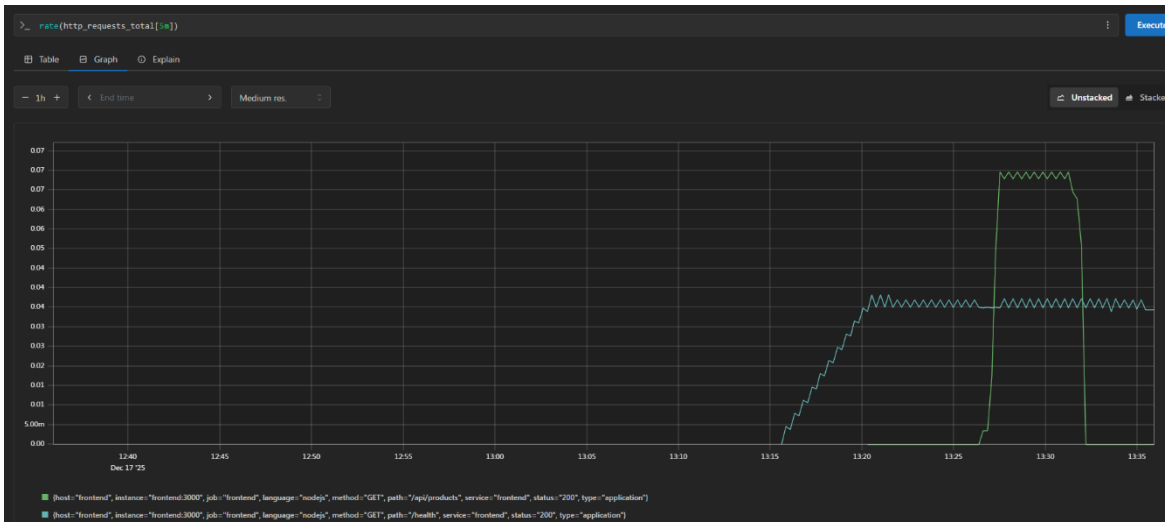
Q13. Dans l'onglet Graph, exécutez cette requête et notez le résultat :



Que signifie un résultat up{job="frontend"} = 1 ?

La métrique `up` indique l'état de disponibilité d'une target. `up{job="frontend"} = 1` signifie que le service frontend est actuellement accessible et que Prometheus peut scraper ses métriques avec succès.

Q14. Exécutez cette requête pour voir le taux de requêtes HTTP par seconde sur les 5 dernières minutes :

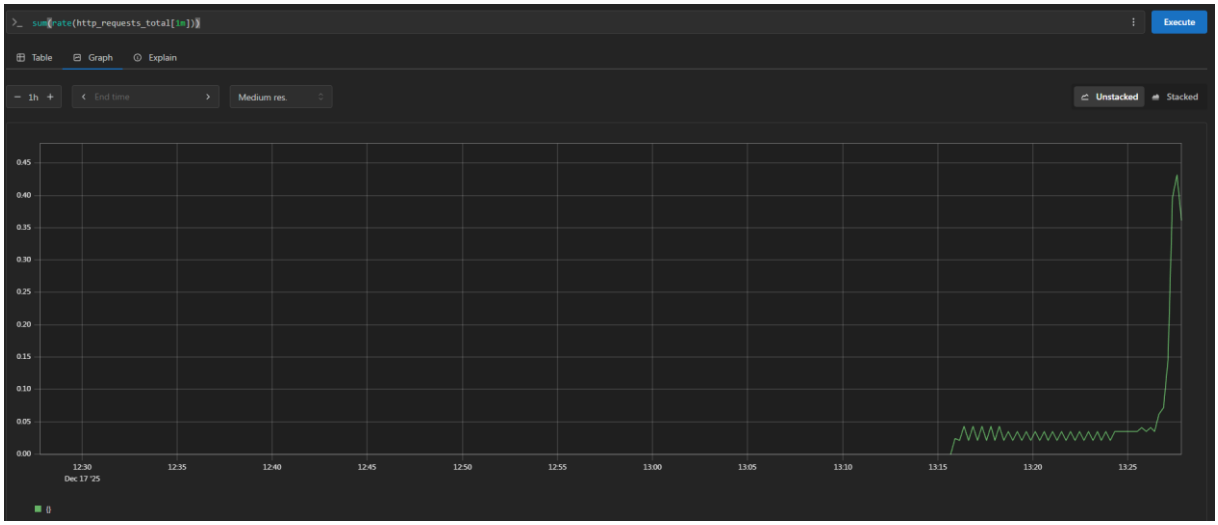


17 séries temporel sont retournés.

Q15. Filtrez pour ne voir que les requêtes vers le backend avec le code 200 :

flask_http_request_total{job="backend", status="200"}

Étape 2.3 : Génération de trafic



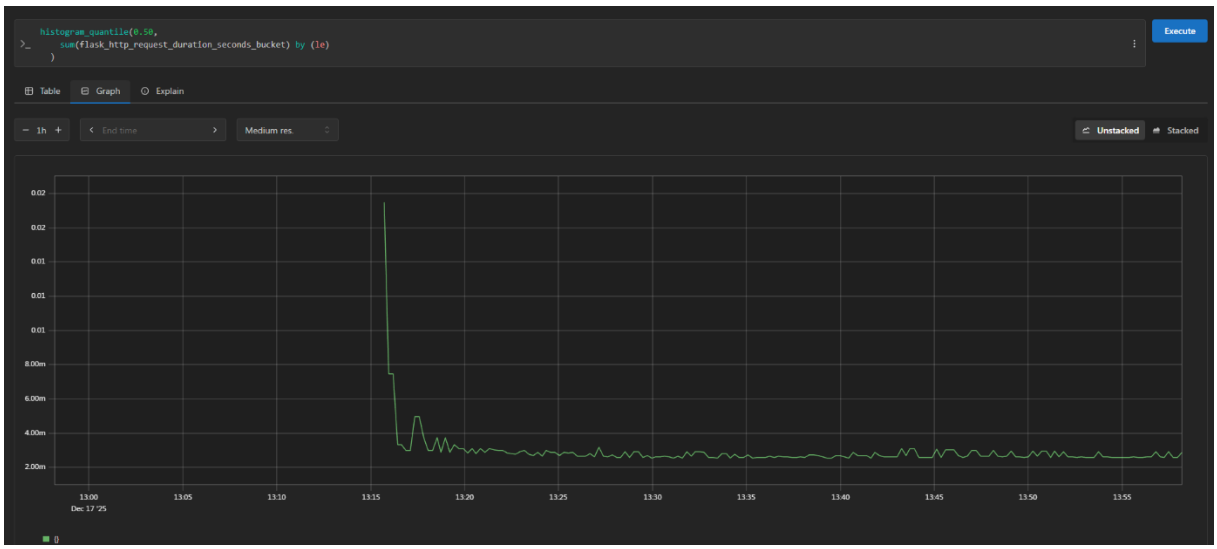
On obtient environ 0.432 req/s

Étape 2.4 : Analyse de latence

Q17. Le backend expose une métrique http_request_duration_seconds (histogram). Quelle requête PromQL permet de calculer la latence médiane (P50) sur 5 minutes ?

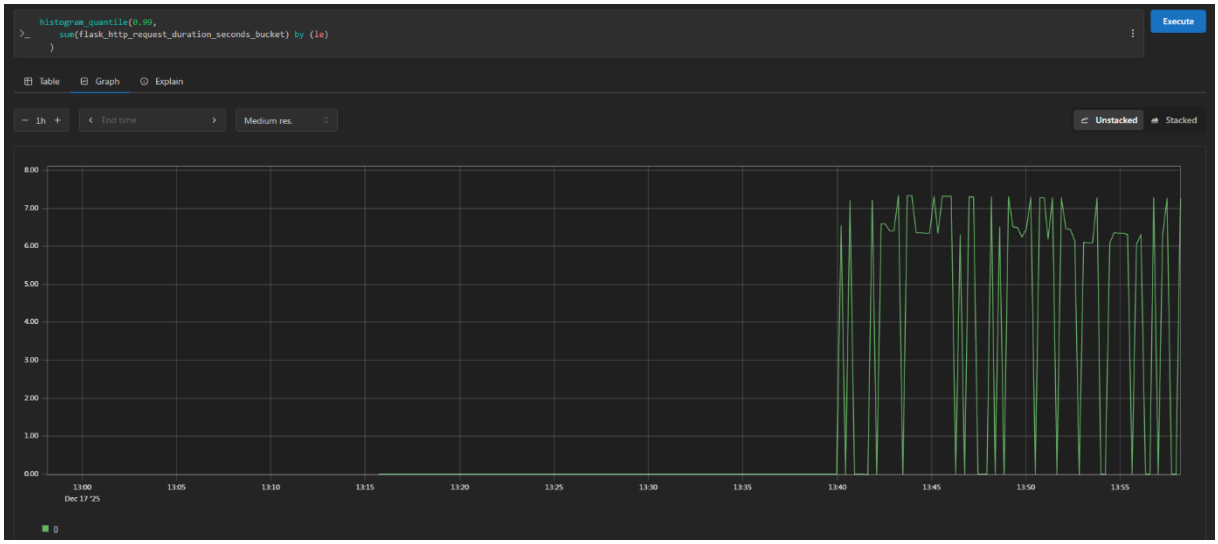
```
histogram_quantile(0.50,
  rate(flask_http_request_duration_seconds_bucket[5m])
)
```

Q18. Dans l'interface web du frontend (http://localhost:3000), cliquez 5 fois sur le bouton "Requête Lente". Quelle est la nouvelle latence P50 dans Prometheus ?



P50 = 0.0027417061844868347 secondes

Q19. Comparez avec la latence P99 :



P99 = 7.1547333920110185 secondes

Q20. Que signifie une différence importante entre P50 et P99 ?

Une différence importante entre P50 (0,0027 s) et P99 (7,15 s) signifie que la majorité des requêtes sont très rapides (P50 proche de zéro), mais qu'une petite fraction de requêtes subit une latence extrêmement élevée (P99).

Cela indique la présence de "outliers" ou de requêtes très lentes qui ne sont pas représentatives de la latence moyenne mais qui peuvent fortement impacter l'expérience utilisateur.

En pratique, cela suggère de rechercher la cause de ces requêtes lentes, par exemple :

- endpoints spécifiques lents (/api/slow)
- Blocages dans la base de données
- Problèmes de ressources ou contention

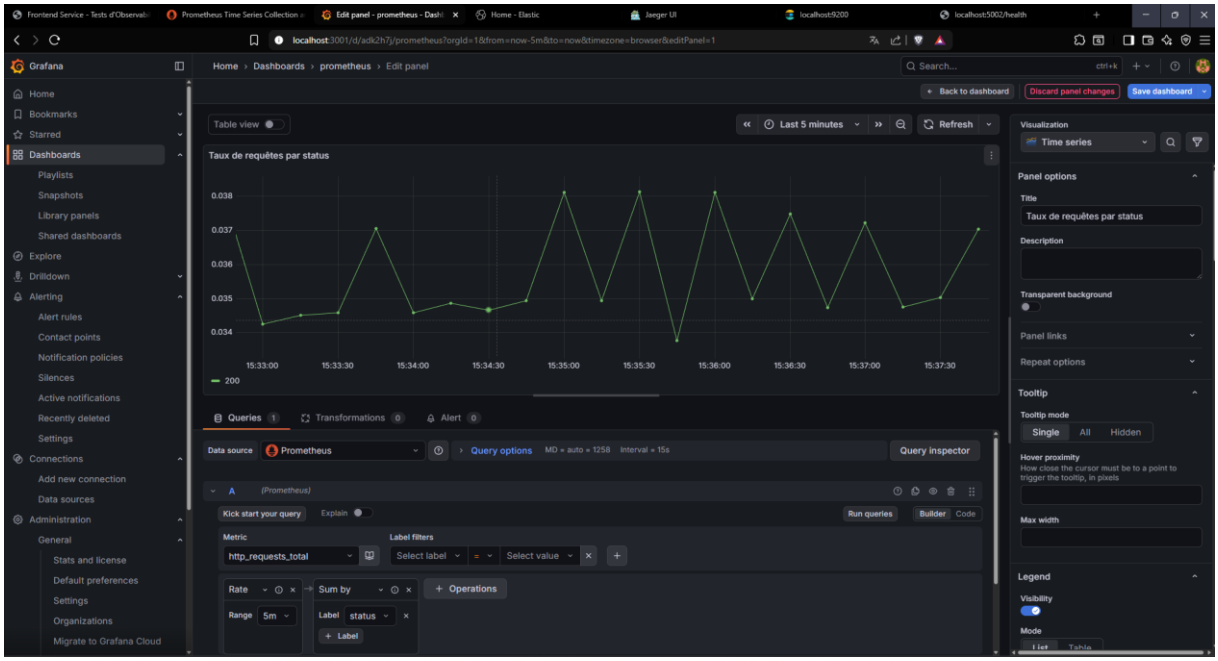
Étape 2.5 : Dashboard Grafana

Q21. Allez dans Configuration > Data Sources. Quel est le status de la datasource Prometheus ?

☒ Connected ✅

☐ Error ❌

Q22. Créez un nouveau dashboard sum(rate(http_requests_total[5m])) by (status) :



Q24. Sauvegardez votre dashboard et exportez-le au format JSON.

Chemin : **Dashboard settings >** *dashboard* **>** *prometheus.json*

Table des matières