Généralité :

Notre infrastructure sera composée de plusieurs base de données redis et un serveur backend (dans un premier temps) connecté à notre front end.
On fera des analyses grâce à l'application prometheus qui s'occupera de scrapper les données du back end et des bases redis.
Grafana nous affichera les metrics récupérés via prometheus.

Test sur le Backend:

1.Vertical Scaling Manuel :

Le Vertical Scaling (ou *mise à l'échelle verticale*) consiste à ajuster les ressources allouées à une même instance de pod (CPU/mémoire) pour répondre aux besoins de l'application, contrairement au scaling horizontal qui ajoute des instances supplémentaires. Le Vertical Scaling présente de nombreux avantages tel que :

- Simplicité de Gestion : Le scaling vertical ne nécessite pas de modifier l'architecture applicative : un seul pod est concerné.
- Performances Stables : il supprime les latences liées à la communication inter-pods et les problèmes de cohérence des données.
- Compatibilité avec les Applications Monolithiques : Idéale pour les applications non conçues pour le parallélisme.

Notre Implémentation :

- Méthode : Scaling vertical manuel (sans VPA automatisé)
- Configuration actuelle :
    - CPU :
        - Request : 100m (minimum garanti)
        - Limit : 500m (plafond)
    - Mémoire :
        - Request : 128Mi

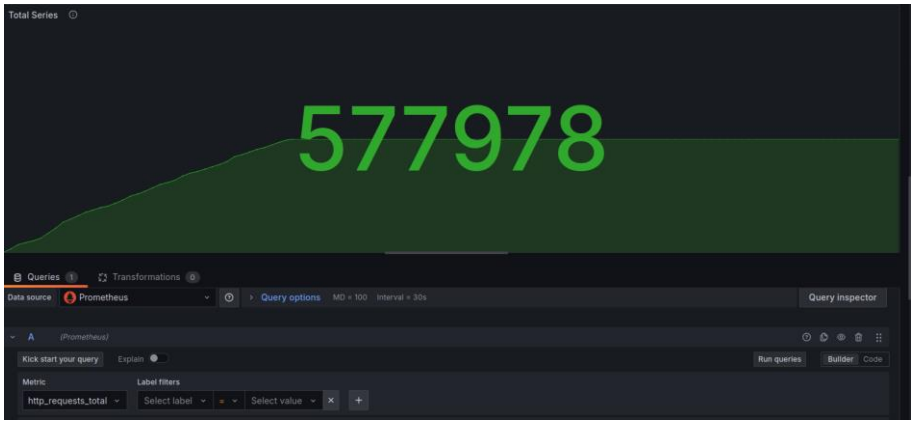
- Limit : 256Mi

Protocole appliqué :

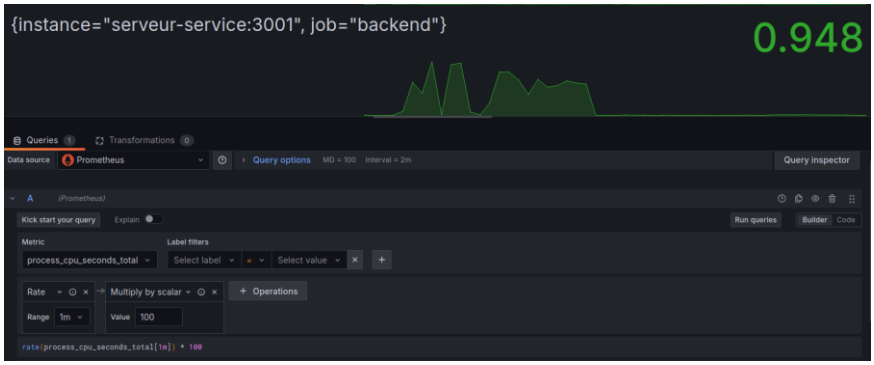
Nous avons évalué notre configuration en utilisant le script node fetchData.js pour générer différents types de charge sur notre application. Dans un premier temps, nous avons exécuté des requêtes HTTP simples pour tester les performances de base du serveur. Parallèlement, nous avons simulé des interactions plus complexes impliquant des opérations de données. Durant ces tests, nous avons systématiquement mesuré plusieurs indicateurs de performance, notamment la latence des requêtes, l'utilisation des ressources processeur, la consommation mémoire et la réactivité du système. Ces mesures nous ont permis d'analyser finement le comportement de l'application sous différentes conditions de charge

Graphes observés via Grafana :

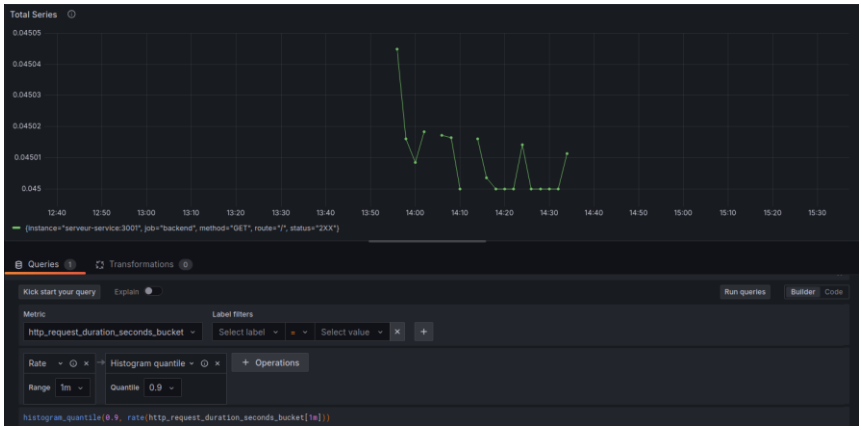
Nombre de requête http envoyé:



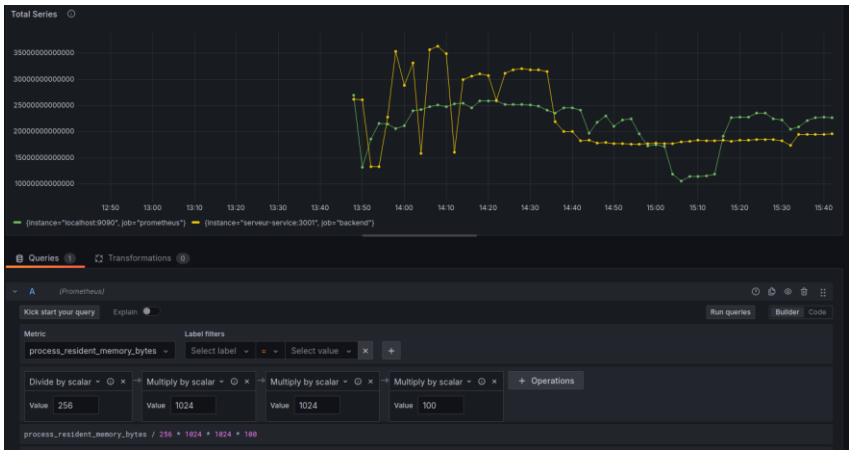
Utilisation CPU:



Latence HTTP:



Mémoire Utilisée :



Event Loop Lag:

Observation :

- On a utilisé 20% du CPU (500m).
- La mémoire utilisée est de 42.9 MiB sur 256Mi.
- La latence http observé est inférieur à 50ms.
- Le délai de la boucle d'événements est de 32.8ms.

Interprétation :

On observe que les valeurs observées sont largement inférieures au seuil critique, 80% pour l'utilisation du cpu, 90% pour la mémoire utilisée, une latence http inférieure à 100ms et enfin un event loop inférieur à 50ms. Les métriques observées démontrent que l'application est extrêmement fluide et réactive, offrant une excellente expérience utilisateur.

Conclusion:

On conclut que le pod est sous-utilisé dans la configuration actuelle, les ressources allouées (500m CPU / 256Mi mémoire) sont trop généreuses pour la charge réelle. Le pod pourrait fonctionner avec moins de ressources sans perte de performance. Dans la suite du rapport, afin de tester l'horizontal scaling, on va baisser les ressources accordées au pod.

2.Horizontal Pod:

Le scaling horizontal (ou *mise à l'échelle horizontale*) est un mécanisme fondamental dans Kubernetes permettant d'adapter dynamiquement le nombre d'instances (pods) d'une application en fonction de la charge, contrairement au scaling vertical qui modifie les ressources allouées à un pod existant. Cette approche offre plusieurs avantages :

- Meilleure résilience : La répartition de charge sur plusieurs pods réduit les risques de single point of failure
- Adaptation élastique : Capacité à absorber les pics de charge en ajoutant des instances
- Optimisation des coûts : Désallocation des ressources inutilisées lors des périodes creuses

Notre implémentation du HPA :

- Allocation de ressources par pod :
    - *Requests* (minimum garanti) : 100m CPU / 64Mi mémoire
    - *Limits* (maximum autorisé) : 200m CPU / 128Mi mémoire
- Paramètres de scaling :
    - Plage de réplicas : 1 (minimum) à 5 (maximum)
    - Métrique de déclenchement : Utilisation moyenne du CPU à 50%