Optimisation du serveur

Durant cette période, j'ai travaillé sur l'optimisation du serveur pour améliorer ses performances. Pour ce faire, j'ai décidé d'utiliser la clusterisation de Node.js, qui permet de créer plusieurs processus enfants (*workers*) qui s'exécutent simultanément sur plusieurs cœurs de CPU. Cela permet d'améliorer les performances de l'application en répartissant les tâches sur plusieurs cœurs de CPU.

Si le processus en cours d'exécution est le processus principal (master), il crée un processus enfant (*worker*) pour chaque cœur de CPU disponible sur la machine. Si un processus enfant (*worker*) se termine, le processus principal en crée un nouveau pour maintenir le nombre de processus enfants constant. Si le processus en cours d'exécution est un processus enfant (*worker*), il démarre l'application en écoutant le port `PORT` et affiche un message indiquant que le serveur a démarré.

Le principal fichier où se déroulent toutes ces opérations est `server.ts`.

Problèmes rencontrés avec la génération d'un graphique de flamme

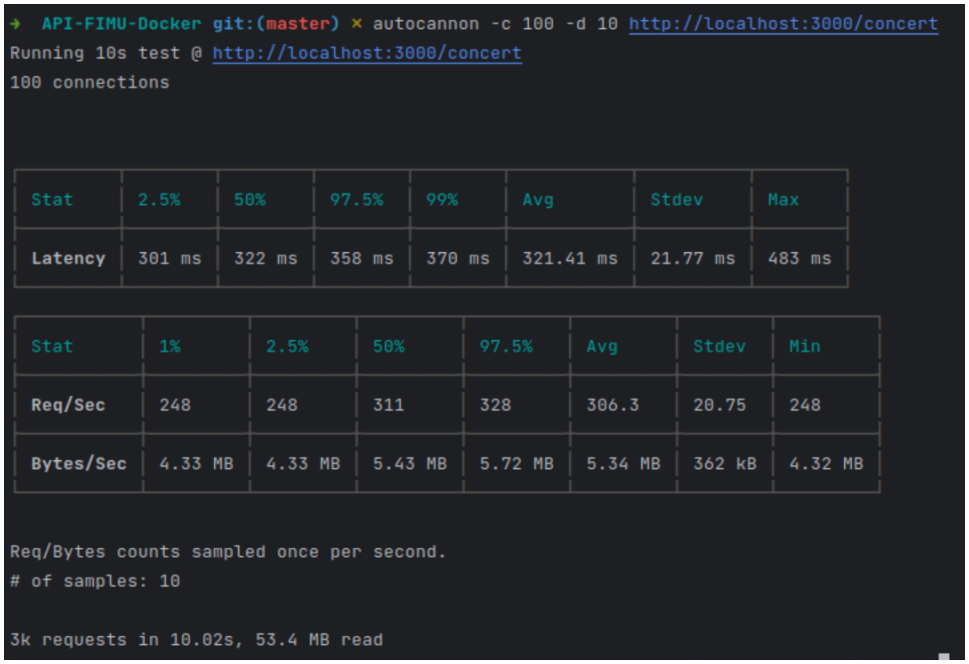
Cependant, lors de mes tentatives pour générer un graphique de flamme, j'ai rencontré plusieurs problèmes. En effet, l'API est dans un docker, ce qui a compliqué la tâche. Les fichiers que me créais `perf` étaient illisibles pour `perf script`.

Évaluation des performances avec Autocannon

Afin d'évaluer les performances du serveur, j'ai utilisé l'outil `Autocannon` pour tester les temps de réponse de plusieurs routes. Après plusieurs tests, j'ai choisi de retenir la route `concert` car elle contient une quantité importante de données et est constituée de différents `INNER JOIN`. Les résultats ont montré que la clusterisation a permis de diviser par deux le temps de réponse du serveur pour cette route en particulier. Ces résultats sont très encourageants et montrent que l'optimisation des performances peut avoir un impact significatif sur l'expérience utilisateur.

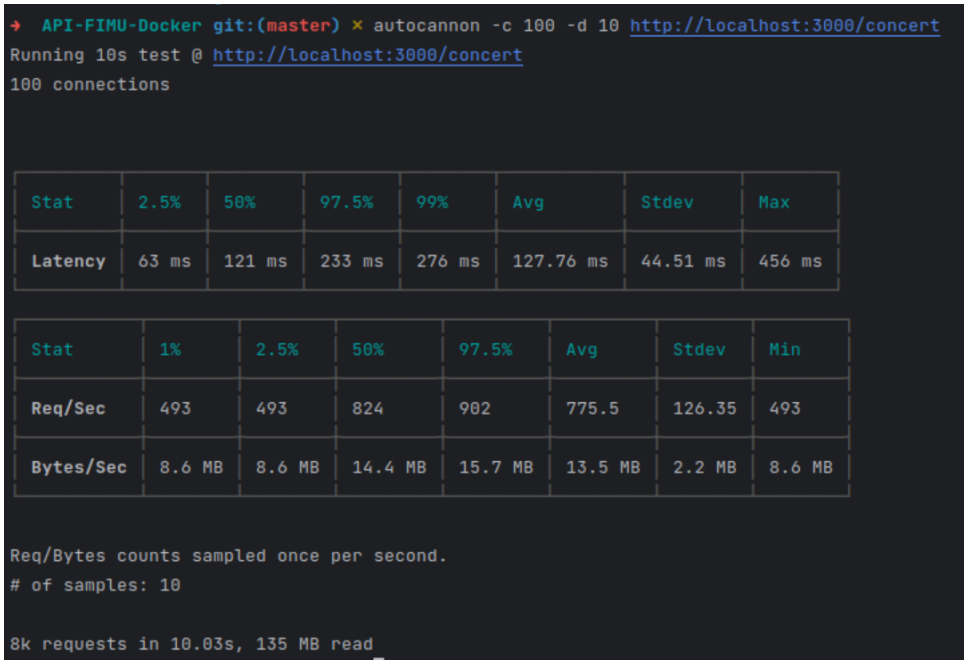
Captures d'écran

Sans Cluster



```
→  API-FIMU-Docker git:(master) ✗ autocannon -c 100 -d 10 http://localhost:3000/concert
Running 10s test @ http://localhost:3000/concert
100 connections


┌─────────┬──────┬──────┬───────┬──────┬──────────┬─────────┬────────┐
│ Stat    │ 2.5% │ 50%  │ 97.5% │ 99%  │ Avg      │ Stdev   │ Max    │
├─────────┼──────┼──────┼───────┼──────┼──────────┼─────────┼────────┤
│ Latency │ 301 ms │ 322 ms │ 358 ms │ 370 ms │ 321.41 ms │ 21.77 ms │ 483 ms │
└─────────┴──────┴──────┴───────┴──────┴──────────┴─────────┴────────┘

┌───────────┬─────────┬─────────┬─────────┬─────────┬─────────┬─────────┬─────────┐
│ Stat      │ 1%      │ 2.5%    │ 50%     │ 97.5%   │ Avg     │ Stdev   │ Min     │
├───────────┼─────────┼─────────┼─────────┼─────────┼─────────┼─────────┼─────────┤
│ Req/Sec   │ 248     │ 248     │ 311     │ 328     │ 306.3   │ 20.75   │ 248     │
├───────────┼─────────┼─────────┼─────────┼─────────┼─────────┼─────────┼─────────┤
│ Bytes/Sec │ 4.33 MB │ 4.33 MB │ 5.43 MB │ 5.72 MB │ 5.34 MB │ 362 kB  │ 4.32 MB │
└───────────┴─────────┴─────────┴─────────┴─────────┴─────────┴─────────┴─────────┘

Req/Bytes counts sampled once per second.
# of samples: 10

3k requests in 10.02s, 53.4 MB read
```

Avec Cluster



```
→  API-FIMU-Docker git:(master) ✗ autocannon -c 100 -d 10 http://localhost:3000/concert
Running 10s test @ http://localhost:3000/concert
100 connections


┌─────────┬──────┬──────┬───────┬──────┬──────────┬─────────┬────────┐
│ Stat    │ 2.5% │ 50%  │ 97.5% │ 99%  │ Avg      │ Stdev   │ Max    │
├─────────┼──────┼──────┼───────┼──────┼──────────┼─────────┼────────┤
│ Latency │ 63 ms │ 121 ms │ 233 ms │ 276 ms │ 127.76 ms │ 44.51 ms │ 456 ms │
└─────────┴──────┴──────┴───────┴──────┴──────────┴─────────┴────────┘

┌───────────┬─────────┬─────────┬─────────┬─────────┬─────────┬─────────┬─────────┐
│ Stat      │ 1%      │ 2.5%    │ 50%     │ 97.5%   │ Avg     │ Stdev   │ Min     │
├───────────┼─────────┼─────────┼─────────┼─────────┼─────────┼─────────┼─────────┤
│ Req/Sec   │ 493     │ 493     │ 824     │ 902     │ 775.5   │ 126.35  │ 493     │
├───────────┼─────────┼─────────┼─────────┼─────────┼─────────┼─────────┼─────────┤
│ Bytes/Sec │ 8.6 MB  │ 8.6 MB  │ 14.4 MB │ 15.7 MB │ 13.5 MB │ 2.2 MB  │ 8.6 MB  │
└───────────┴─────────┴─────────┴─────────┴─────────┴─────────┴─────────┴─────────┘

Req/Bytes counts sampled once per second.
# of samples: 10

8k requests in 10.03s, 135 MB read
```