- Rapport Jmeter (Tests de charge) :

- ***Nom et Prénom :*** **BOUKHRIS** Mohamed Lyazid
- ***Encadré par :*** **QUAFAFOU** Mohamed

Table des matières

1. Tests de charge avec Apache JMeter sur l'application conteneurisée :

1.1 Objectif des tests :

L'objectif est de mesurer les performances de l'application web conteneurisée (backend Flask + frontend Nginx) en simulant une charge via Apache JMeter, et d'observer la réponse du système sous différents niveaux de stress.

1.2 Environnement de test :

- Pour le backend :
  - ***Outil de test :*** Apache JMeter 5.6.3
  - ***Nombre de threads (utilisateurs virtuels) :*** 100
  - ***Période de montée en charge (Ramp-up) :*** 1 seconde
  - ***Nombre de boucles :*** 10 itérations par utilisateur
  - ***Méthode de test :*** Requête HTTP POST vers une API REST
  - ***Conteneurs utilisés :*** Application déployée dans des conteneurs Docker (Flask pour le backend, Nginx pour le frontend)
  - ***Adresse de l'API testée :*** http://localhost:5000/api/check_name
  - ***Données envoyées :*** {"name": "Achraf"}
  - ***Type de contenu :*** application/json

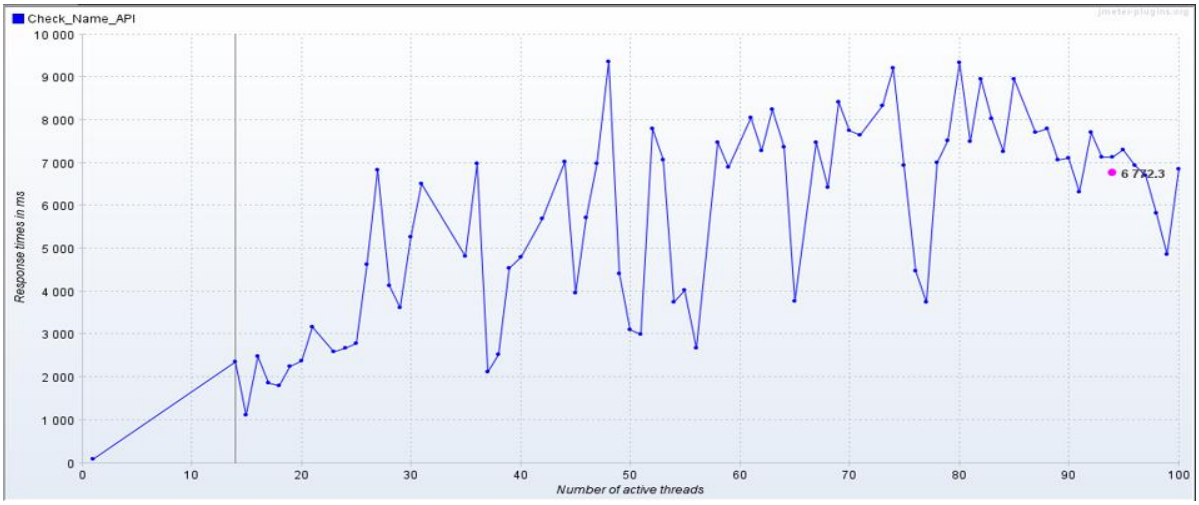
- Pour le frontend :
  - ***Outil de test*** : Apache JMeter 5.6.3
  - ***Nombre de threads (utilisateurs virtuels)*** : 100
  - ***Période de montée en charge (Ramp-up)*** : 1 seconde

- ***Nombre de boucles :*** 10 itérations par utilisateur
- ***Méthode de test :*** Requête HTTP GET vers l'interface web statique
- ***Conteneurs utilisés :*** Application déployée dans un conteneur Docker avec Nginx
- ***Adresse de l'interface testée :*** http://localhost:8080/
- ***Données envoyées :*** Aucune (requête GET simple sans paramètres)
- ***Type de contenu attendu :*** text/html

1.3 Configuration JMeter :

- **Thread Group** : 100 utilisateurs, ramp-up de 1 seconde, 10 itérations.
- **Header Manager** : pour inclure Content-Type: application/json
- **Plugins utilisés** :
  - View Results Tree
  - jp@gc – Response Times vs Threads

1.4 Résultats – Backend Dockerisé :



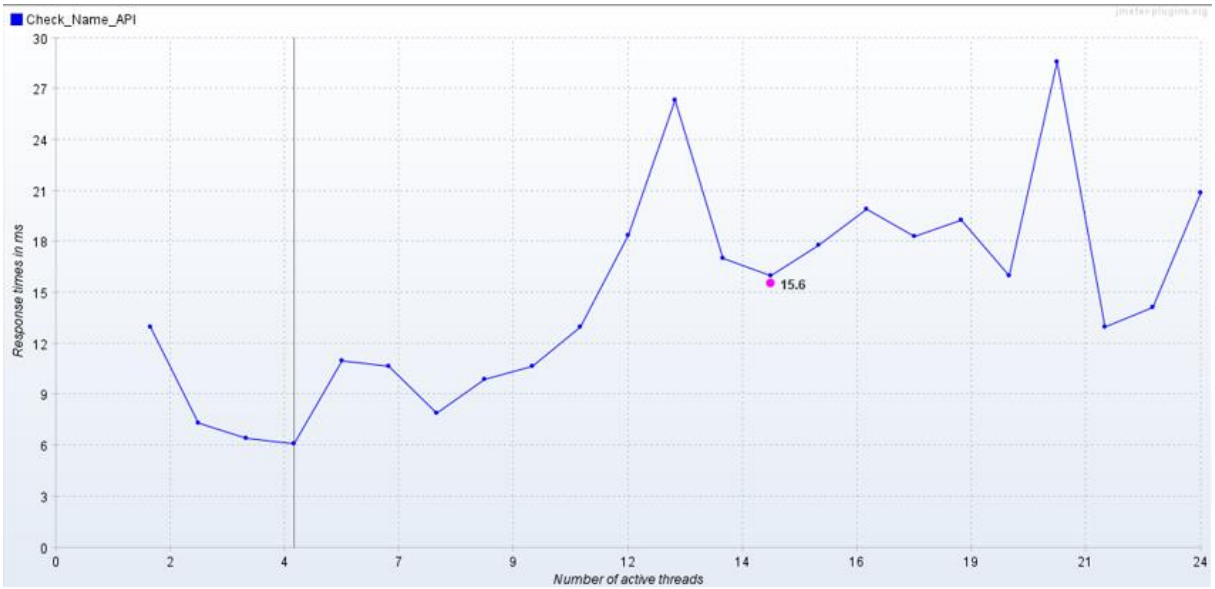
- ***Observation :***

- Les temps de réponse augmentent rapidement à partir de 20 threads.
- À 100 threads, la latence dépasse 6000 ms, avec des pics à plus de 9000 ms.
- Le backend semble moins stable sous forte charge.

- ***Analyse :***

Cela peut indiquer que le conteneur Flask en mode développement ne gère pas bien la concurrence. Il faudrait un **serveur WSGI** comme Gunicorn pour une meilleure scalabilité.

1.5 Résultats – Frontend Dockerisé (Nginx) :



- ***Observation :***

- Les temps de réponse restent faibles (en dessous de **40 ms**) jusqu'à 25 threads.
- Le front-end est **plus stable** que le back-end.

- ***Analyse :***

Nginx étant un serveur web optimisé pour la haute performance, il gère très bien la charge, mais il dépend toujours du backend pour générer les données dynamiques.

1.6 Comparaison frontend vs backend :

Métrique	Backend Dockerisé	Frontend Dockerisé
Temps moyen (100 threads)	6000ms	20ms
Stabilité sous charge	Faible	Bonne
Comportement à > 50 threads	Pics de latence	Réponses rapides

1.7 ***Axes d'amélioration :***

- Utiliser Gunicorn ou uWSGI avec Flask pour la prod.
- Ajouter du caching ou un reverse proxy Nginx devant Flask.

1.8 ***Conclusion :***

Les tests JMeter ont permis de détecter que le backend Flask conteneurisé devient rapidement instable au-delà de 20 utilisateurs simultanés. Le frontend Nginx, quant à lui, reste performant même avec 100 threads.

2. Tests de charge avec Apache JMeter dans un environnement virtualisé :

2.1 Objectif des tests :

Le but est de mesurer les performances de l'application web installée dans une machine virtuelle (avec Oracle VirtualBox). Pour cela on utilise Apache JMeter afin de simuler un grand nombre d'utilisateurs. Les tests sont réalisés à deux niveaux :

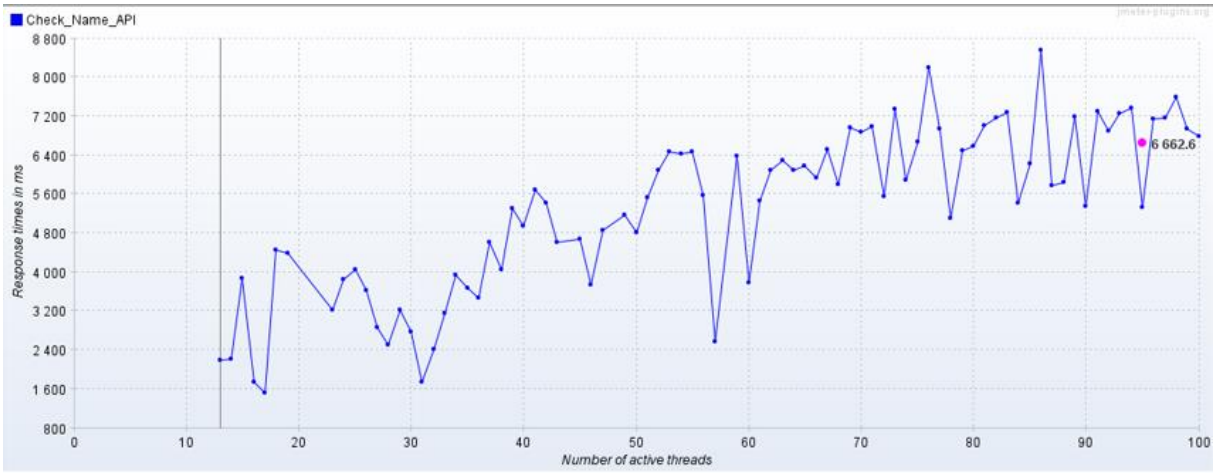
- sur le backend (une API développée avec Flask).
- sur le frontend (une interface web servie par Nginx).

L'objectif final est de comparer les résultats obtenus avec ceux des tests faits dans un environnement conteneurisé avec Docker afin de voir lequel est le plus performant quand l'application est soumise à une forte charge.

2.2 Environnement de test :

L'environnement de test est identique à celui utilisé pour les tests Docker, afin de garantir une comparaison équitable.

2.3 Résultats – Backend Virtualisé :



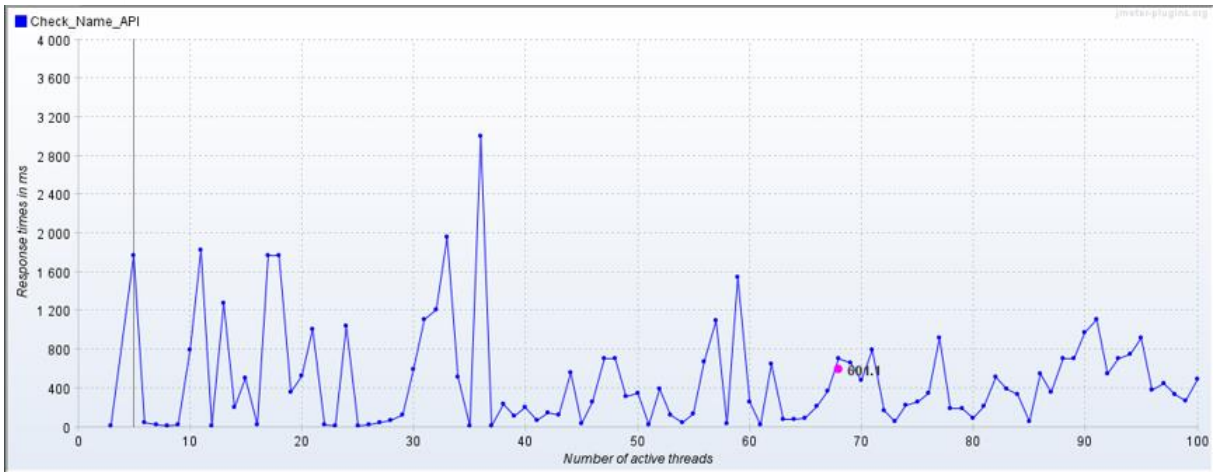
- ***Observation :***

- Les temps de réponse augmentent significativement à partir de 30 threads.
- Des pics sont observés jusqu'à 8600 ms à pleine charge.
- La courbe est instable, avec de nombreuses fluctuations.

- ***Analyse :***

Le backend lancé en mode développement via Flask ne semble pas adapté à des charges importantes. Le serveur ne peut traiter efficacement un grand nombre de requêtes simultanées dans cet environnement, ce qui provoque des délais importants.

2.4 Résultats – Frontend Virtualisé :



- ***Observation :***

- Les performances sont relativement stables jusqu'à 60 utilisateurs.
- Les temps de réponse varient entre 400 ms et 2500 ms.
- Les pics sont plus modérés qu'en backend, mais présents dès 30-40 threads.

- ***Analyse :***

Le frontend reste plus performant que le backend, mais la latence reste dépendante de ce dernier. L'environnement virtualisé et les ressources allouées à la VM peuvent également limiter les performances.

2.5 Conclusion :

Les tests réalisés dans un environnement virtualisé montrent que le backend développé avec Flask atteint rapidement ses limites dès qu'il dépasse 30 utilisateurs simultanés. On constate une montée importante de la latence, ce qui ralentit fortement les réponses de l'API. Le frontend, même s'il résiste un peu mieux grâce à Nginx, est aussi affecté car il dépend directement du backend pour obtenir les données. Pour améliorer les performances, il est recommandé d'utiliser un serveur plus adapté comme Gunicorn à la place du serveur de développement Flask. De plus, une meilleure configuration de la machine virtuelle (processeur, mémoire, réseau) pourrait aider à obtenir un comportement plus stable et plus proche de celui attendu en production.

3. Comparaison Docker vs Virtualisation – Performances de l'application :

Critère	Environnement Dockerisé	Environnement Virtualisé
Backend – Temps moyen	6000 ms à 100 threads	6600 ms à 100 threads
Backend – Pics max	9000 ms	8600 ms
Frontend – Temps moyen	20-40 ms (jusqu'à 25 threads)	400 à 2500 ms (dès 30 threads)
Stabilité frontend	Très bonne	Moyenne
Stabilité backend	Faible	Faible
Serveur Flask	Mode dev dans conteneur	Mode dev dans VM Ubuntu

4.1 Conclusion :

En conclusion, les tests montrent que l'environnement Docker donne de meilleures performances, surtout pour le frontend grâce à Nginx. L'environnement virtualisé est plus lent, surtout quand il y a beaucoup d'utilisateurs.

Dans les deux cas, le backend Flask est le point faible. Il ne supporte pas bien la charge car il tourne en mode développement. Pour améliorer cela, il faut :

- Utiliser uWSGI (serveur de prod) pour lancer Flask
- Ajouter un Nginx devant le backend

Docker est donc plus adapté si l'application est bien optimisée.