



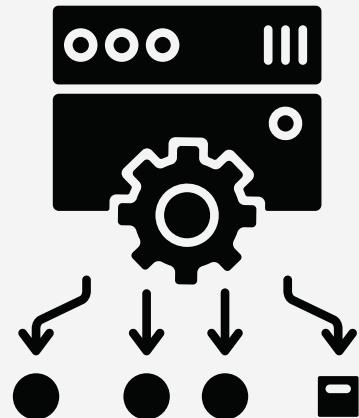
Rapport Projet

Administration Systèmes

Première Année Cycle d'Ingénieur : Sécurité IT et Confiance Numérique

Load

Balancer

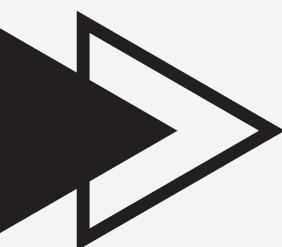


Réalisé par :

- BOUTALMAOUINE MOHAMED
- AHOUARI BELAID
- ER-RAMIQI MOHAMMED

Encadré par :

Pr AZIZI



CONTENU

- ▶ INTRODUCTION X
- ▶ Définition et Importance X
- ▶ Concepts de Base du Load Balancing X
- ▶ Algorithmes de Répartition de Charge X
- ▶ Mise en Œuvre Pratique X
- ▶ Tests & Application(sera au cours de la présentation) X
- ▶ Conclusion X

INTRODUCTION

Dans le cadre de notre module "Programmation et administration système", nous avons entrepris un mini projet visant à simuler un load balancer. Ce projet a pour objectif principal de comprendre et de mettre en pratique les concepts fondamentaux de la répartition de charge dans un environnement informatique. En effet, la répartition de charge, ou load balancing, est une technique cruciale utilisée pour améliorer la réactivité et la disponibilité des applications en distribuant les charges de travail de manière équitable entre plusieurs serveurs.

Dans cette simulation, nous allons implémenter un algorithme de load balancing et examiner son fonctionnement à travers divers scénarios de charge. Notre étude portera sur les différents algorithmes de répartition de charge, tels que le round-robin, le least connections, et le hash-based. Chaque méthode sera évaluée en termes de performance, de scalabilité, et de robustesse.

Ce projet permettra également d'explorer les aspects pratiques de l'administration système, tels que la configuration des serveurs, la surveillance des performances, et la gestion des pannes. En combinant théorie et pratique, nous espérons acquérir une compréhension approfondie des mécanismes de répartition de charge et de leur importance dans l'optimisation des systèmes informatiques modernes.

À travers ce mini projet, nous allons non seulement développer nos compétences en programmation, mais aussi renforcer notre capacité à gérer des systèmes complexes, ce qui est essentiel pour toute carrière dans le domaine des technologies de l'information et de la communication.

DÉFINITION ET IMPORTANCE



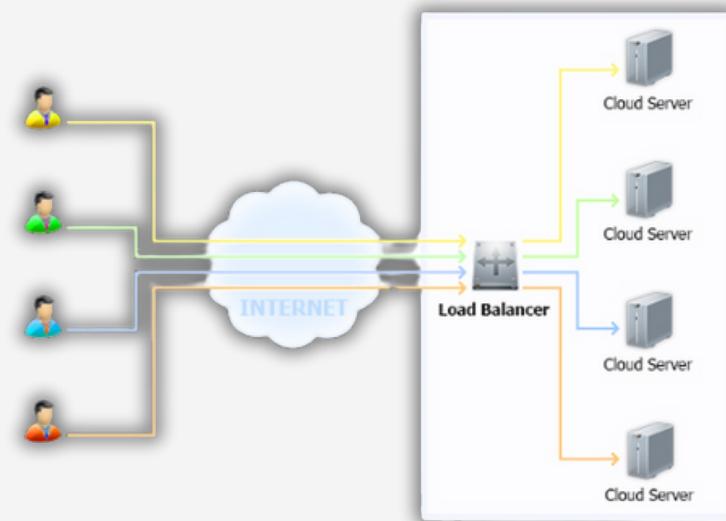
► Qu'est-ce qu'un load balancer ?

Un load balancer a pour tâche de répartir la charge de travail (d'un serveur Web / email / base de données par exemple) entre un ou plusieurs serveurs (backends). Pour que ce service de répartition de charge soit opérationnel, il faudra naturellement faire appel à des serveurs à la configuration et aux données contenues parfaitement identiques.

En utilisant un serveur de load balancing connecté en amont à un ensemble de serveurs Cloud, vous obtenez par la réPLICATION du contenu, une forte augmentation du niveau de la fiabilité et de la pérennité des services concernés.

Via un service de répartition de charge, vous accroissez le niveau de rendement des ressources déployées au sein de votre infrastructure tout en maximisant leur capacité de production en réduisant les temps d'accès.

Sans oublier le point-clé recherché par la mise en place d'une politique de load balancing : éviter l'indisponibilité des services critiques fortement sollicités.



Ainsi surviendrait l'arrêt inopiné d'un des serveurs Cloud en load balancing , le load blancer prendra automatiquement la décision de ne plus travailler avec ledit serveur, l'exclura de son pool de loadbalancing et n'utilisera plus que les X serveurs du pool restants mis à sa disposition pour répartir la charge.

Suite à cette action, le système de monitoring associé au service de laod balancing enverra un email et/ou SMS de notification au contact renseigné (lors du déploiement du service).

DÉFINITION ET IMPORTANCE

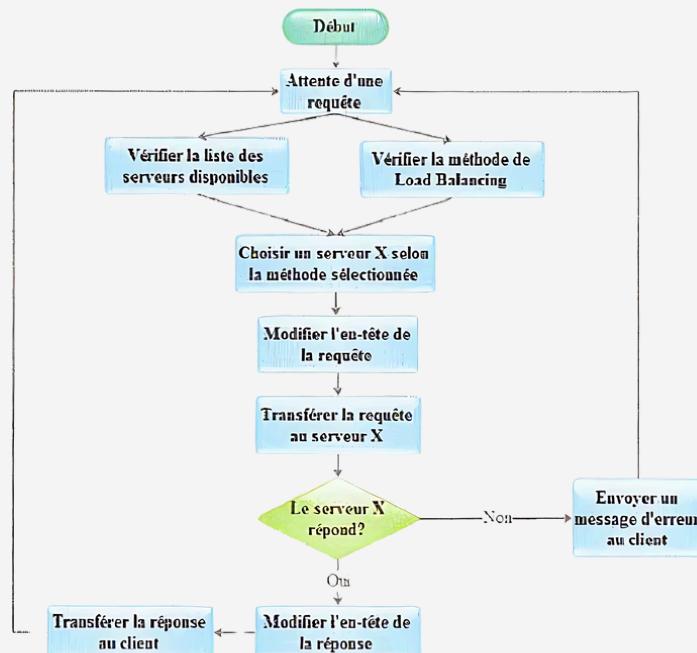


▶ Pourquoi est-il crucial pour les infrastructures modernes ?

Les load balancers sont cruciaux pour les infrastructures modernes car ils garantissent la haute disponibilité, la scalabilité et l'optimisation des performances des services en ligne. En répartissant intelligemment le trafic entre plusieurs serveurs, ils évitent les surcharges, réduisent la latence et assurent une tolérance aux pannes. Cela permet aux applications de rester disponibles et réactives, même en cas de forte demande ou de défaillance de certains serveurs. De plus, ils améliorent la sécurité en distribuant les attaques potentielles et simplifient la gestion des ressources, rendant l'infrastructure plus fiable et efficace.

▶ La procédure générale de Load Balancing

La figure illustre le mode de fonctionnement d'un Load Balancer exécuté au niveau d'un contrôleur SDN. Au lancement du contrôleur, le Load Balancer se met en état d'attente. Lorsque le contrôleur reçoit une requête destinée au Load Balancer, celui-ci vérifie la liste des serveurs disponibles et sélectionne un serveur selon la méthode d'équilibrage de charge implémentée. Ensuite, il modifie l'en-tête de la requête en remplaçant l'adresse de destination par l'adresse (IP et MAC) du serveur choisi.



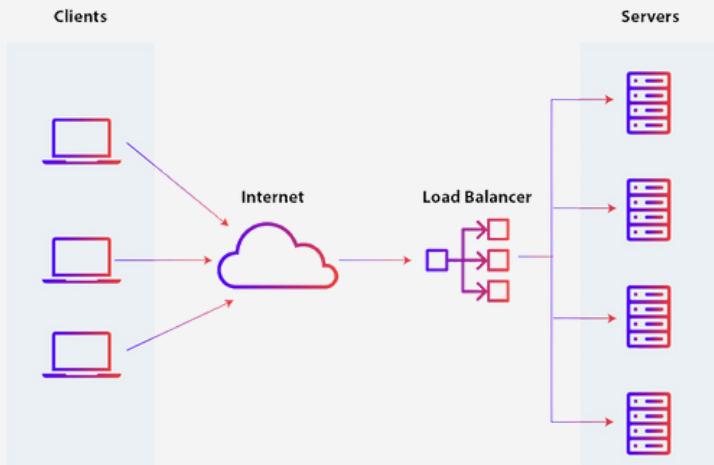
CONCEPTS DE BASE DU LOAD BALANCING



▶ Fonctionnalités Principales

1. Répartition de la Charge (Load Balancing)

La répartition de la charge est une technique essentielle pour optimiser les performances et garantir la disponibilité des services. Elle consiste à distribuer le trafic ou les demandes de travail sur plusieurs serveurs ou ressources. Les algorithmes de répartition, tels que le Round Robin ou le Least Connections, permettent de distribuer les requêtes de manière équilibrée. En surveillant en temps réel les performances des serveurs, cette fonctionnalité permet d'ajuster la distribution du trafic pour éviter toute surcharge et assurer la redondance. Ainsi, en cas de défaillance d'un serveur, le trafic est automatiquement redirigé vers les autres serveurs disponibles, assurant une continuité de service sans interruption.



2. Haute Disponibilité (High Availability)

La haute disponibilité vise à maintenir les services opérationnels en permanence, même lors de pannes ou de maintenances planifiées. Cela se réalise par la mise en place de composants matériels et logiciels redondants, éliminant les points de défaillance unique. Les systèmes de basculement automatique (failover) permettent de remplacer instantanément un composant défaillant par un autre, sans intervention manuelle. Cette fonctionnalité inclut également la possibilité d'effectuer des mises à jour et des maintenances sans interruption du service, grâce à une surveillance continue qui détecte et résout les problèmes avant qu'ils n'affectent la disponibilité.

CONCEPTS DE BASE DU LOAD BALANCING



3. Scalabilité (Scalability)

La scalabilité est la capacité d'un système à gérer une augmentation de la charge de travail en ajoutant des ressources supplémentaires. Elle se divise en scalabilité verticale (ajout de ressources à un seul serveur) et en scalabilité horizontale (ajout de nouveaux serveurs pour partager la charge). L'élasticité, qui permet d'ajuster dynamiquement les ressources en fonction de la demande, est une composante clé. Des tests de charge sont également réalisés pour garantir que le système peut évoluer efficacement sans compromettre ses performances.

4. Tolérance aux Pannes (Fault Tolerance)

La tolérance aux pannes assure que le système continue de fonctionner correctement même lorsqu'une ou plusieurs de ses composantes tombent en panne. Cela est possible grâce à l'utilisation de composants redondants qui éliminent les points de défaillance unique. Les systèmes de détection de pannes surveillent en temps réel et isolent les défaillances pour limiter leur impact. De plus, le recouvrement automatique permet au système de basculer vers des composants de secours sans intervention humaine, garantissant ainsi une résilience et une continuité de service accrues.

CONCEPTS DE BASE DU LOAD BALANCING



► Types de Load Balancers

1. Load Balancers Matériels

Ce sont des dispositifs physiques conçus pour gérer l'équilibrage de charge à grande échelle. Ils sont généralement déployés en data center et sont capables de gérer un grand volume de trafic. Ils sont également souvent dotés de fonctionnalités avancées et offrent une haute performance, mais peuvent en revanche se révéler coûteux et moins flexibles que leur pendant logiciel.



F5 Server Load Balancer, Size: 600 X 720 X 100 Mm

2. Load balancers logiciels

Ces équilibreurs de charge sont des applications installées sur un serveur. Généralement moins coûteux et plus flexibles que leurs homologues matériels, ils peuvent être facilement mis à jour ou modifiés. Cependant, leur performance peut dépendre des ressources du serveur sur lequel ils sont installés.

The screenshot shows the 'loadbalancer.org' interface with a red sidebar containing links like System Overview, Local Configuration, Cluster Configuration, Advanced, View Configuration, Reports, Logs, and Support. The main area displays 'SYSTEM OVERVIEW' with two sections: 'Web-Cluster-1' and 'Web-Cluster-2'. Each cluster has three entries: 'REAL SERVER', 'Web01', and 'Web02'. The 'Web-Cluster-2' section also includes an 'FTP-Cluster-1' entry. Below the overview are two graphs: 'Network Bandwidth' showing traffic over time, and 'System Load Average' showing CPU usage. At the bottom, there's a footer for 'Load Balancer Enterprise ADC'.

Load Balancer Enterprise ADC

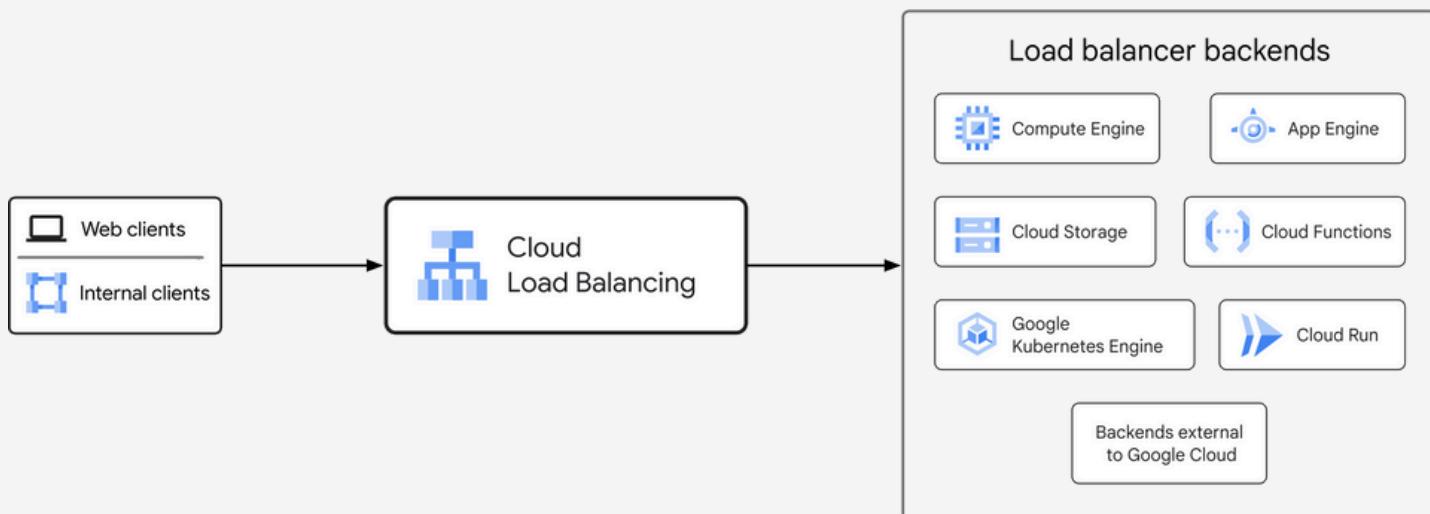
CONCEPTS DE BASE DU LOAD BALANCING



▶ Types de Load Balancers

3. Load balancers basés sur le cloud

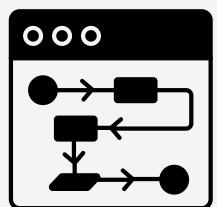
Les load balancers basés sur le cloud sont des outils essentiels pour la distribution de charges de travail sur plusieurs serveurs, assurant une répartition équilibrée du trafic entrant. Ils sont souvent déployés dans des environnements cloud pour garantir une disponibilité élevée, une évolutivité et une résilience accrues.



Présentation de Cloud Load Balancing(GOOGLE CLOUD)

Parmi les fournisseurs de cloud les plus populaires proposant des solutions de load balancing cloud, on peut citer Amazon Web Services (AWS) avec Elastic Load Balancing (ELB), Google Cloud Platform (GCP) avec Google Cloud Load Balancing et Microsoft Azure avec Azure Load Balancer. Chacun de ces services offre des fonctionnalités spécifiques et des intégrations avec d'autres services cloud.

ALGORITHMES DE RÉPARTITION DE CHARGE



➤ Algorithmes Statistiques

Les algorithmes statistiques de répartition de charge utilisent des méthodes mathématiques pour distribuer le trafic entre plusieurs serveurs. Ils assurent une distribution équilibrée des demandes, optimisant ainsi les performances et la disponibilité des services. Les principaux algorithmes utilisés sont :

1. Round Robin

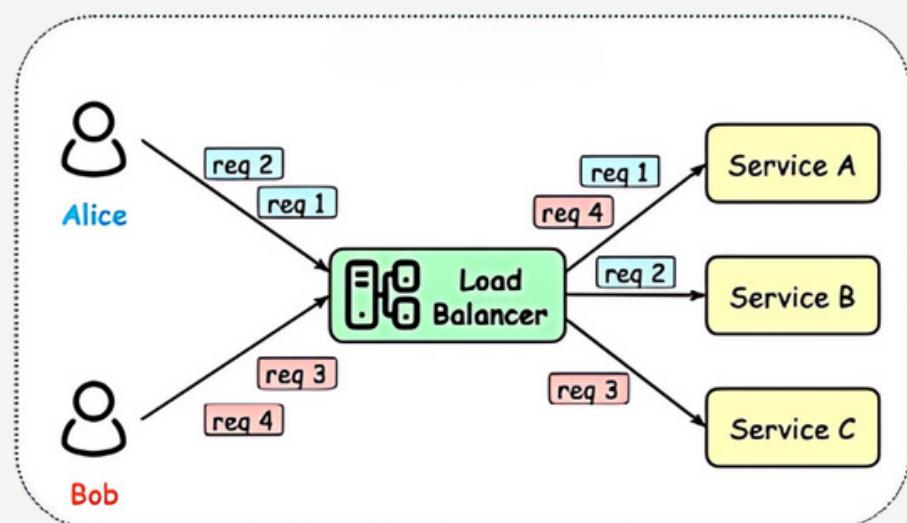
L'algorithme Round Robin distribue les demandes de manière séquentielle à travers une liste de serveurs. Chaque demande successive est envoyée au serveur suivant dans la liste, et une fois arrivé à la fin de la liste, l'algorithme recommence au début.

Caractéristiques :

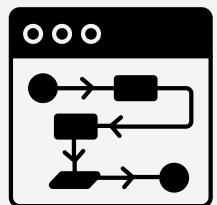
- Simplicité : Facile à implémenter et à comprendre.
- Équité : Assure une distribution égale des requêtes sur tous les serveurs, indépendamment de leur charge actuelle.
- Adaptabilité : Convient bien aux environnements où les demandes sont relativement égales en termes de charge de traitement.

Limites :

- Inégalité potentielle de charge : Ne tient pas compte de la charge réelle des serveurs, ce qui peut entraîner une surcharge de certains serveurs si les requêtes ont des exigences de traitement variées.



ALGORITHMES DE RÉPARTITION DE CHARGE



2. Sticky Round Robin

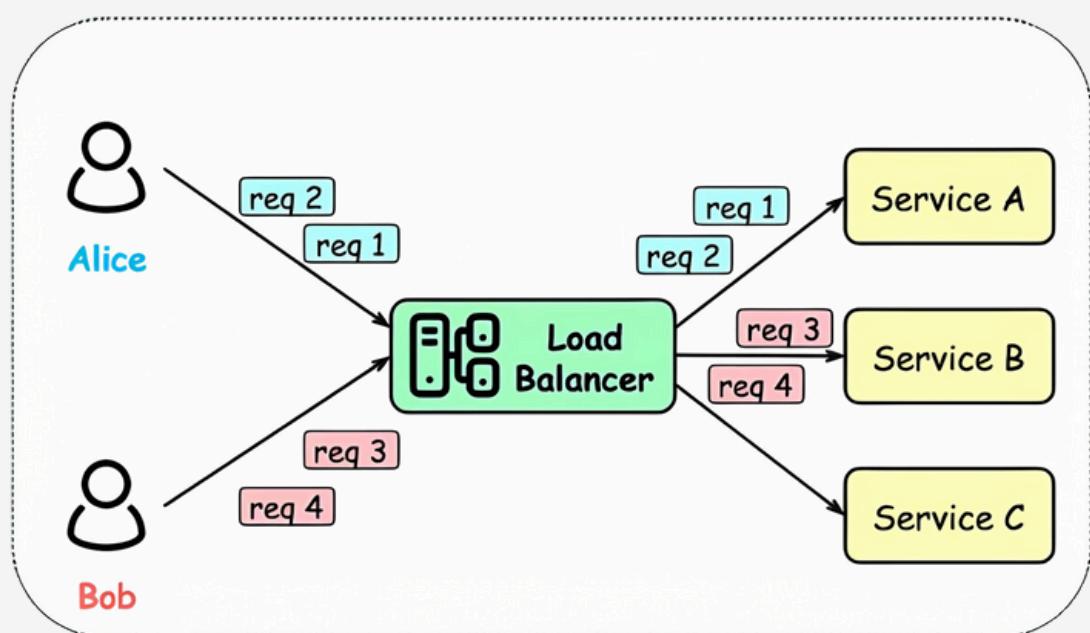
L'algorithme Sticky Round Robin améliore le Round Robin en ajoutant un mécanisme de persistance de session, de sorte que toutes les demandes d'un même client sont envoyées au même serveur.

Caractéristiques :

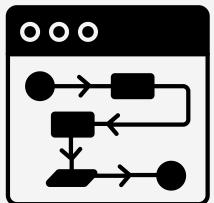
- Persistance de session : Garantit que les requêtes successives d'un même client vont au même serveur.
- Simplicité : Relativement facile à implémenter tout en ajoutant une logique de session.

Limites :

- Inégalité de charge : Peut entraîner une charge inégale si un serveur reçoit une proportion plus élevée de requêtes persistantes.
- Complexité supplémentaire : L'ajout de la gestion des sessions peut rendre l'algorithme plus complexe par rapport au Round Robin simple.



ALGORITHMES DE RÉPARTITION DE CHARGE



3. Weighted Round Robin

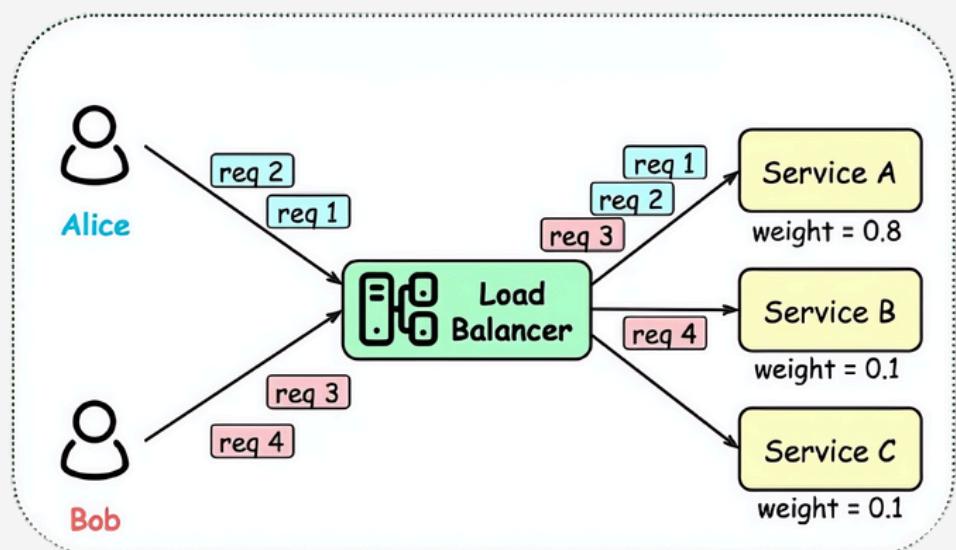
L'algorithme Weighted Round Robin attribue un poids à chaque serveur en fonction de sa capacité ou de ses performances. Les serveurs avec des poids plus élevés reçoivent plus de requêtes.

Caractéristiques :

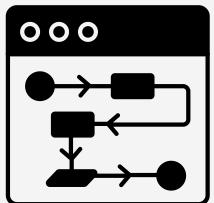
- Attribution pondérée des demandes : Chaque instance de service se voit attribuer un poids qui représente sa capacité ou sa performance relative. Les demandes sont ensuite distribuées en fonction de ces poids, permettant une utilisation équilibrée des ressources disponibles.
- Flexibilité dans la gestion des ressources : En assignant des poids différents à chaque instance de service, l'algorithme offre une flexibilité pour gérer les ressources de manière dynamique en fonction des besoins du système.

Limites :

- Complexité de configuration : La détermination des poids appropriés pour chaque instance de service peut être complexe et nécessiter une analyse approfondie des capacités et des performances de chaque serveur. Une mauvaise configuration des poids peut entraîner un déséquilibre de charge.
- Sensibilité aux changements de charge : Bien que l'algorithme Weighted Round Robin soit efficace pour répartir la charge lorsqu'il est correctement configuré, il peut être sensible aux fluctuations de charge. Les changements soudains de la charge peuvent nécessiter des ajustements fréquents des poids pour maintenir un équilibre optimal.



ALGORITHMES DE RÉPARTITION DE CHARGE



4. Hash

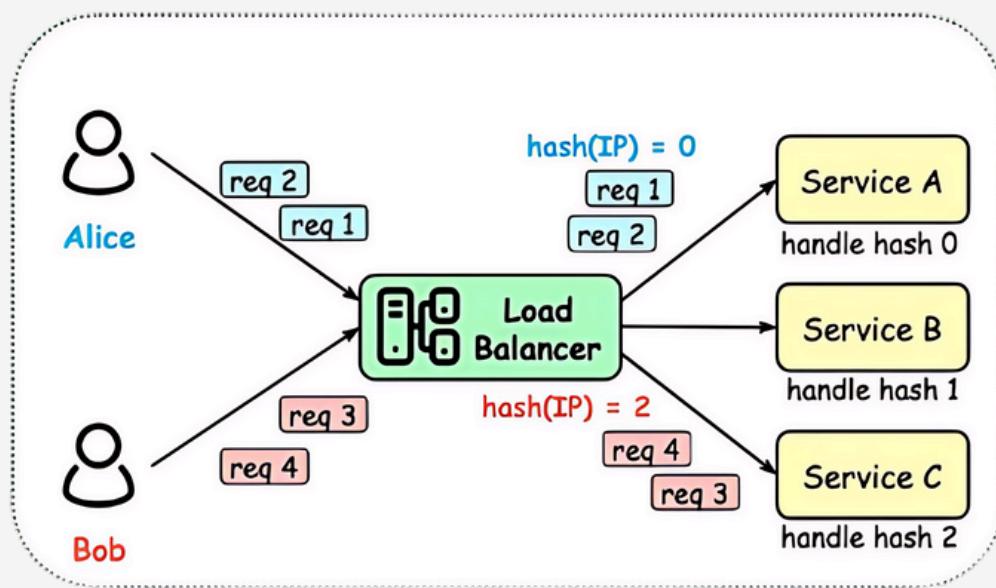
L'algorithme de hash consiste à appliquer une fonction de hachage aux adresses IP ou aux URL des demandes entrantes. Cette fonction de hachage transforme les adresses IP ou les URL en valeurs numériques (ou en une clé unique) de manière déterministe. Ensuite, les demandes sont routées vers les instances de service pertinentes en fonction du résultat de la fonction de hachage.

Caractéristiques :

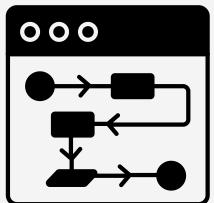
- Distribution équilibrée des demandes : L'algorithme de hash assure une répartition uniforme des demandes entre les différentes instances de service, optimisant ainsi l'utilisation des ressources.
- Prédicibilité du routage : Les demandes partageant des caractéristiques similaires, telles que des adresses IP ou des URL, sont routées de manière cohérente vers la même instance de service, offrant une prédictibilité dans le traitement des requêtes.

Limites :

- Manque d'évolutivité dynamique : L'ajout ou la suppression dynamique d'instances de service peut entraîner une redistribution des demandes, potentiellement causant des interruptions de service pendant cette transition.
- Déséquilibre avec des données non uniformément distribuées : Des données non uniformément distribuées, telles que des adresses IP générant plus de demandes que d'autres, peuvent entraîner un déséquilibre de charge entre les instances de service, compromettant l'efficacité des ressources et la performance du système.



ALGORITHMES DE RÉPARTITION DE CHARGE

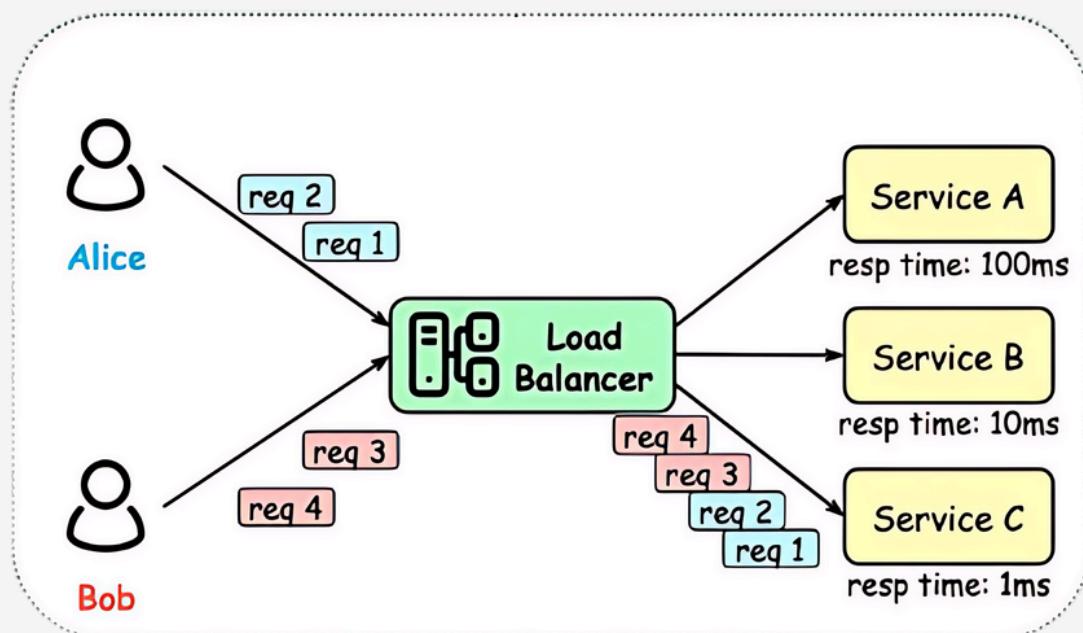


➤ Algorithmes Dynamiques

Les algorithmes dynamiques peuvent affecter de nouvelles ressources à des tâches en cours d'exécution. Concrètement, le système permet aux tâches en cours d'exécution d'être transférées d'une machine chargée à une machine moins chargée. Les algorithmes dynamiques sont particulièrement appropriés aux systèmes à mémoire partagée, en raison de la facilité procurée par ces systèmes pour migrer une tâche en cours d'exécution.

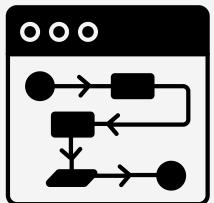
1. Least Response Time(Plus faible temps de réponse)

Cet algorithme attribue une nouvelle requête au serveur qui a le temps de réponse le plus faible à ce moment-là. Cela garantit que les requêtes sont dirigées vers les serveurs qui sont les moins occupés ou les plus rapides pour répondre. C'est une approche efficace pour minimiser le temps d'attente des clients.



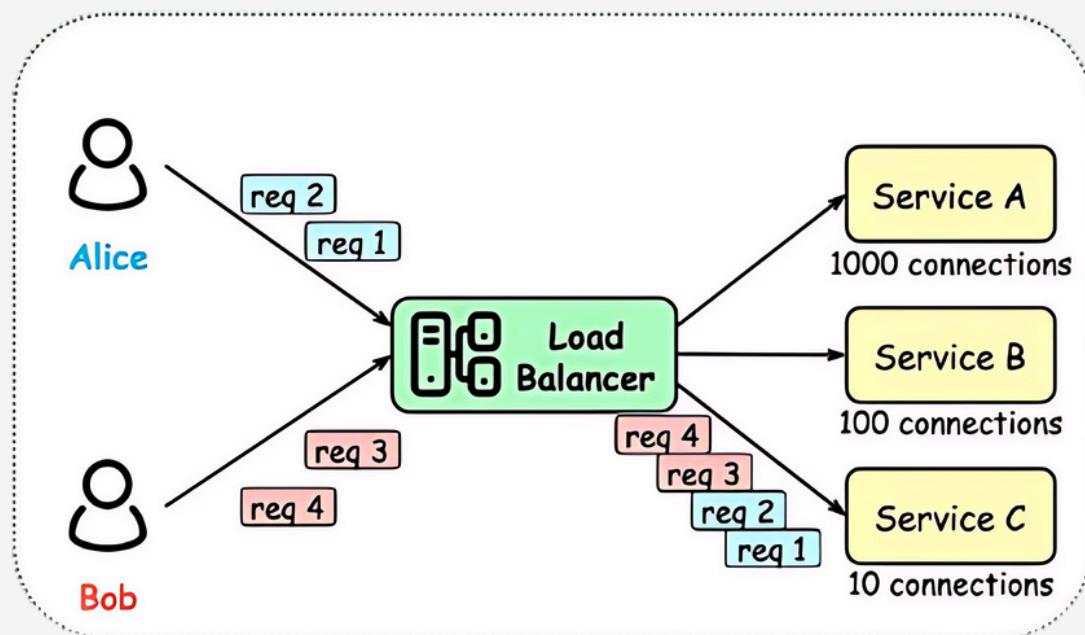
L'avantage de cet algorithme est qu'il permet de minimiser le temps d'attente des clients en choisissant les serveurs les plus réactifs. Cela peut contribuer à améliorer l'expérience utilisateur en garantissant que les demandes sont traitées rapidement et efficacement. Cependant, il est important de surveiller les performances des serveurs de manière proactive pour s'assurer qu'ils fonctionnent de manière optimale.

ALGORITHMES DE RÉPARTITION DE CHARGE



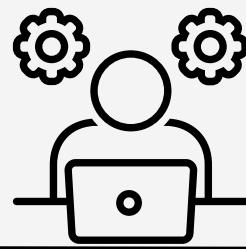
2. Least connections (Moins de connexions)

L'algorithme Least Connections (Moins de connexions) est un mécanisme de répartition de charge qui attribue chaque nouvelle demande au service ou à l'instance de serveur qui a actuellement le moins de connexions simultanées. L'idée principale derrière cet algorithme est de diriger les nouvelles requêtes vers les serveurs qui ont la capacité la plus disponible, ce qui peut contribuer à réduire les temps de réponse globaux et à éviter la surcharge des serveurs.



L'avantage de l'algorithme Least Connections est sa capacité à distribuer la charge de manière dynamique en fonction de la charge réelle de chaque serveur. Cela permet d'éviter la surcharge des serveurs tout en maximisant l'utilisation des ressources disponibles. Cependant, cela peut nécessiter une surveillance continue et des ajustements pour s'assurer que les serveurs sont équilibrés de manière appropriée.

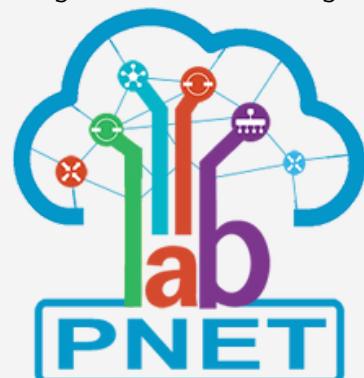
MISE EN ŒUVRE PRATIQUE



► Technologie de simulation([PNETLab](#)):

PNETLab est un environnement virtuel de réseau conçu pour la formation, la simulation et le test de réseaux informatiques. Cette plateforme offre un ensemble d'outils et de fonctionnalités permettant aux utilisateurs d'expérimenter avec diverses configurations réseau, topologies et technologies sans avoir besoin de matériel physique dédié.

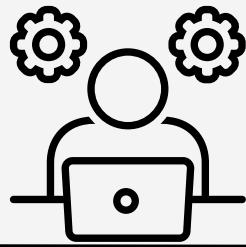
En utilisant PNETLab,
les utilisateurs peuvent créer des réseaux
virtuels complexes, simuler des scénarios réseau
réalistes, tester des configurations réseau,
et effectuer des diagnostics et des dépannages.
L'environnement fournit une interface conviviale
qui permet de déployer rapidement
des équipements réseau virtuels tels que des routeurs, des commutateurs, des
pare-feu, des serveurs, etc., et de les interconnecter selon les besoins.



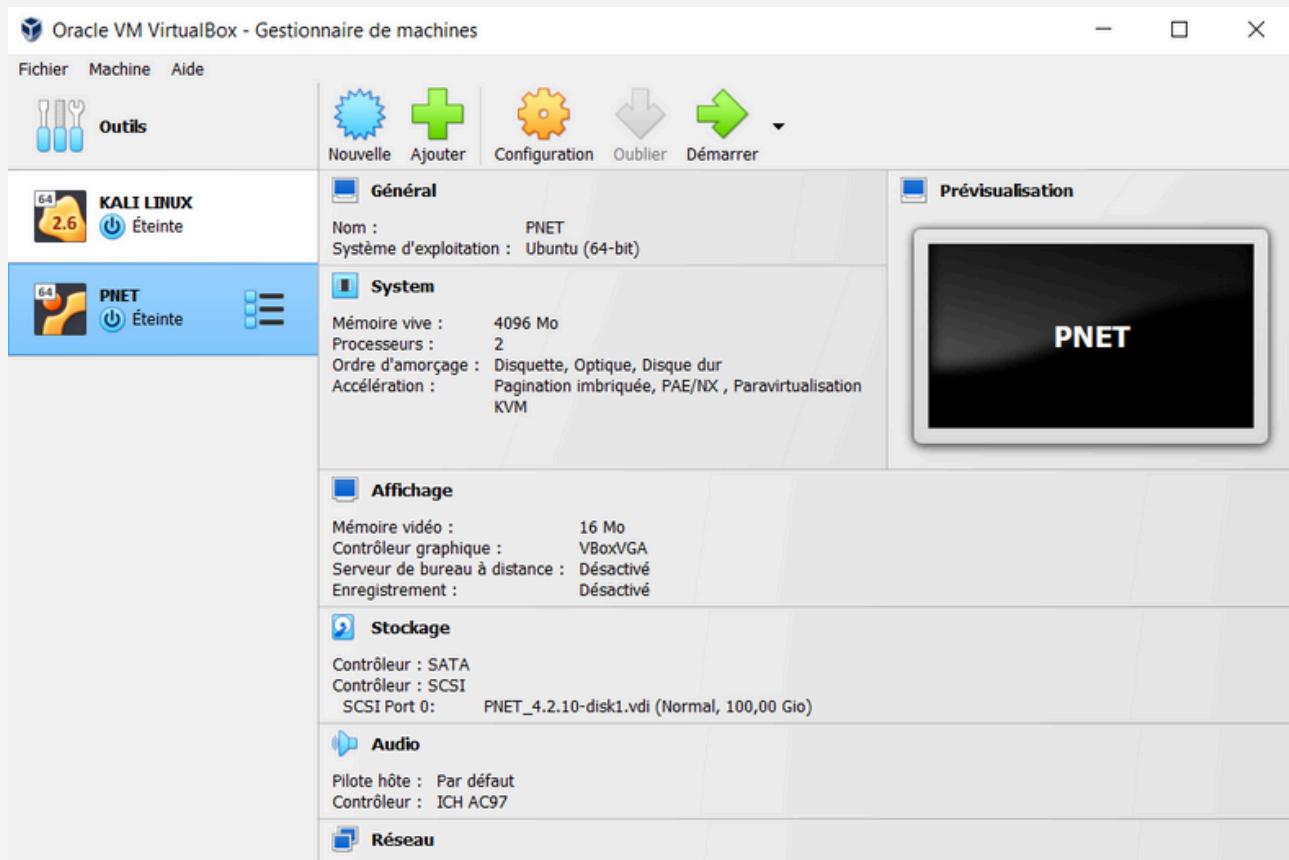
Grâce à sa flexibilité et à sa capacité à reproduire des environnements réseau variés, PNETLab est largement utilisé dans le domaine de l'éducation, de la formation en réseau, du développement de compétences pratiques en TI, ainsi que dans les environnements de test et de développement pour les professionnels des réseaux informatiques.

MISE EN ŒUVRE

PRATIQUE



1. Installation de l'environnement PNETLabs sur VM virtualbox



2. Lancement de l'environnement

```

PNET [En fonction] - Oracle VM VirtualBox
Fichier Machine Écran Entrée Pérophériques Aide
onetlab login: Your service is working normally

PNETLab (default root password is 'pnet')
Use https or http://192.168.0.164/
WARNING: neither Intel VT-x or AMD-V found

onetlab login: root
Password:
Last login: Sun May 19 10:49:44 UTC 2024 from 192.168.0.170 on pts/0
Welcome to Ubuntu 18.04.5 LTS

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 System information as of Sun May 19 13:56:45 UTC 2024

 System load: 0.59      Users logged in:      0
 Usage of /:  9.0% of 96.94GB  IP address for pnet0:  192.168.0.164
 Memory usage: 12%
 Swap usage:  0%          IP address for pnet_nat: 10.0.137.1
 Processes:   210          IP address for docker0: 10.177.0.1

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
 just raised the bar for easy, resilient and secure K8s cluster deployment.
 https://ubuntu.com/engage/secure-kubernetes-at-the-edge

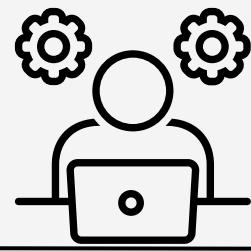
 * Canonical Livepatch is available for installation.
 - Reduce system reboots and improve kernel security. Activate at:
 https://ubuntu.com/livepatch

root@pnetlab:~# _

```

MISE EN ŒUVRE

PRATIQUE



3. Accéder à la page localhost avec l'IP adresse fourni par l'environnement

PNET [En fonction] - Oracle VM VirtualBox

Fichier Machine Écran Entrée Périphériques Aide

```
PNETLab (default root password is 'pnet')
Use https:// or http://192.168.0.164/

WARNING: neither Intel VT-x nor AMD-V found

pnetlab login: root
Password:
Last login: Sun May 19 18:42:03 UTC 2024 on tty1
Welcome to Ubuntu 18.04.5 LTS

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

 System information as of Sun May 19 19:45:39 UTC 2024

 System load: 0.25      Users logged in:      0
 Usage of /: 9.0% of 96.94GB  IP address for pnet0: 192.168.0.164
 Memory usage: 13%          IP address for pnet_nat: 10.0.137.1
 Swap usage: 0%            IP address for docker0: 10.177.0.1
 Processes: 213

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
 just raised the bar for easy, resilient and secure K8s cluster deployment.
 https://ubuntu.com/engage/secure-kubernetes-at-the-edge

 * Canonical Livepatch is available for installation.
 - Reduce system reboots and improve kernel security. Activate at:
 https://ubuntu.com/livepatch

root@pnetlab:~#
```

PNETLab | Lab is Simple

Non sécurisé 192.168.0.164/store/public/admin/main/view

Main 2 Running Labs Accounts System Download Labs Sell Your Labs Devices MOHAMMED2002

Workspace root / Your labs from PNETLab Store

Search Labs

Lab Preview

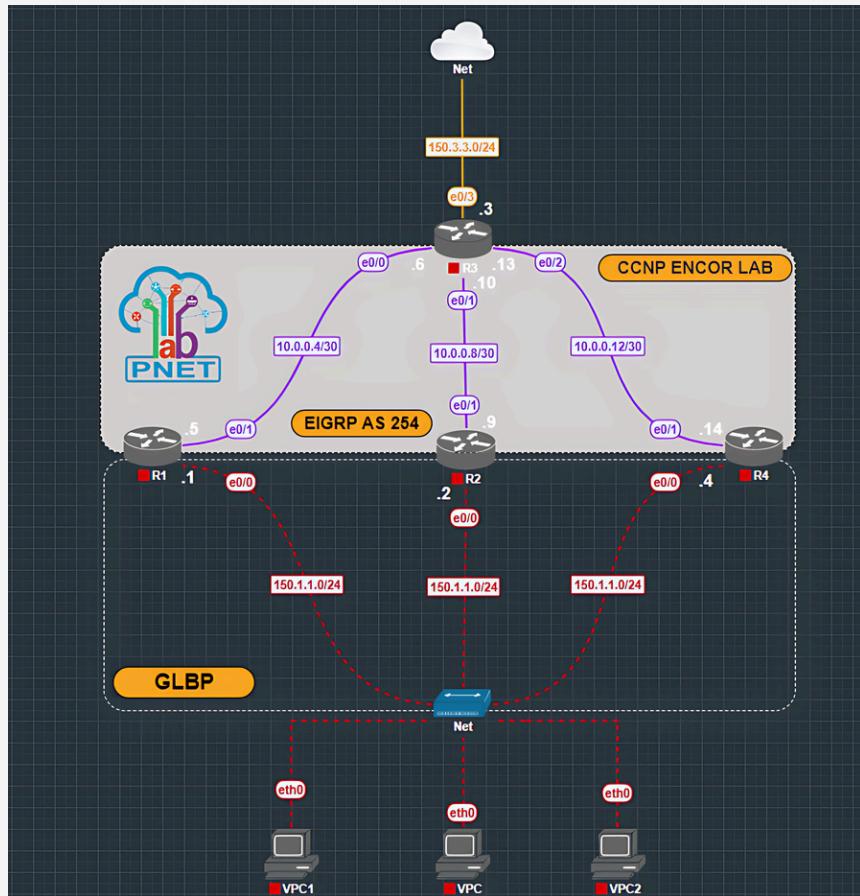
LOAD.uln 19 May 2024 13:09 Public

Gateway Load Balancing Protocol Ver_1.uln 19 May 2024 12:53

MISE EN ŒUVRE PRATIQUE



4. Assemblage de lab pour la simulation de load balancer



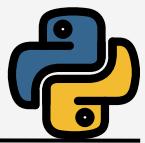
En raison de certaines configurations nécessaires que nous n'avons pas pu réaliser, nous avons changé d'approche pour simuler le Load Balancing. Plutôt que d'utiliser une infrastructure complexe, nous avons opté pour une simulation plus simple avec des scripts Python. Cette approche nous a permis de démontrer efficacement les principes fondamentaux du Load Balancing et d'explorer divers algorithmes de répartition de charge.

En utilisant des scripts Python, nous avons simulé différents serveurs et mis en œuvre un équilibrEUR de charge basique. Cette méthode a permis de distribuer les requêtes de manière cyclique ou en fonction des temps de réponse simulés, illustrant comment le Load Balancing peut optimiser l'utilisation des ressources et améliorer la performance globale du système.

Les tests effectués ont validé l'efficacité de notre solution, montrant qu'un équilibrEUR de charge, même simple, peut gérer le trafic réseau de manière efficace. Cette expérience nous a fourni une compréhension pratique des mécanismes de Load Balancing et de leur impact positif sur la disponibilité et la performance des services en ligne.

MISE EN ŒUVRE PRATIQUE

AVEC PYTHON



Création des serveurs

Ce code Python met en place un serveur HTTP simple en utilisant les modules `http.server` et `socketserver`. Il définit une classe `MyServer` qui hérite de `BaseHTTPRequestHandler`, ce qui lui permet de gérer les requêtes HTTP entrantes.

La méthode `do_GET()` est spécifiée pour traiter les requêtes GET. Lorsqu'une requête GET est reçue, cette méthode envoie une réponse HTTP avec le code 200 (OK), indique le type de contenu comme `text/html`, et envoie une page HTML minimale contenant un titre `

Server 1</h1>`.

La fonction `run()` configure l'adresse et le port du serveur (dans ce cas, l'adresse est vide, ce qui signifie qu'il écoute sur toutes les interfaces disponibles, et le port est défini sur 8000). Ensuite, elle crée une instance de `HTTPServer` en utilisant la classe `MyServer` définie précédemment, puis lance le serveur en appelant `serve_forever()`.

Le bloc d'exécution conditionnelle `if __name__ == '__main__':` s'assure que le script est exécuté en tant que programme principal, et dans ce cas, il appelle la fonction `run()` pour démarrer le serveur.

En résumé, ce code établit un serveur HTTP minimaliste qui répond aux requêtes GET avec une simple page HTML contenant le texte "Server 1". Il fournit une base pour créer des serveurs web plus complexes en Python.

```

● ● ●
1  from http.server import BaseHTTPRequestHandler, HTTPServer
2
3  class MyServer(BaseHTTPRequestHandler):
4      def do_GET(self):
5          self.send_response(200)
6          self.send_header('Content-type', 'text/html')
7          self.end_headers()
8          self.wfile.write(b"<html><body><h1>Server 1</h1></body></html>")
9
10 def run():
11     server_address = ('', 8000)
12     httpd = HTTPServer(server_address, MyServer)
13     print('Server running at localhost:8000...')
14     httpd.serve_forever()
15
16 if __name__ == '__main__':
17     run()
18

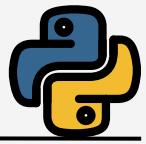
```



On a créé trois serveurs avec des ports différents

MISE EN ŒUVRE PRATIQUE

AVEC PYTHON



▶ [Création des Types différentes de load balance avec différents algorithmes](#)

1. Implimentation load balancer avec l'algorithme Least connections (Moins de connexions)

```

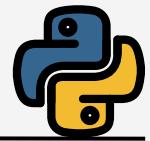
● ● ●

1  from flask import Flask, jsonify
2  import requests
3
4  app = Flask(__name__)
5
6  servers = {
7      "http://localhost:8000": 0,
8      "http://localhost:8001": 0,
9      "http://localhost:8002": 0
10 }
11
12 def get_server():
13     return min(servers, key=servers.get)
14
15 @app.route('/')
16 def index():
17     server = get_server()
18     try:
19         response = requests.get(server)
20         servers[server] += 1
21         return response.text
22     except requests.RequestException as e:
23         return jsonify({'error': str(e)}), 500
24
25 if __name__ == '__main__':
26     app.run(debug=True)
27

```

Ce code crée un serveur web avec Flask qui distribue les requêtes entre plusieurs serveurs disponibles en fonction de leur utilisation actuelle. Il utilise une fonction pour sélectionner le serveur le moins chargé, puis envoie les requêtes vers ce serveur. Si une erreur survient lors de la requête, elle est renvoyée au client avec un code d'état approprié.

MISE EN ŒUVRE PRATIQUE AVEC PYTHON



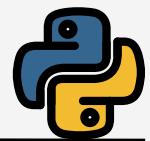
2. Implimentation load balancer avec l'algorithme Round Robin

```
● ● ●

1  from flask import Flask, jsonify
2  import requests
3
4  app = Flask(__name__)
5
6  servers = ["http://localhost:8000", "http://localhost:8001", "http://localhost:8002"]
7  current_server_index = 0
8
9  def get_server():
10     global current_server_index
11     server = servers[current_server_index]
12     current_server_index = (current_server_index + 1) % len(servers)
13     return server
14
15 @app.route('/')
16 def index():
17     server = get_server()
18     try:
19         response = requests.get(server)
20         return response.text
21     except requests.RequestException as e:
22         return jsonify({'error': str(e)}), 500
23
24 if __name__ == '__main__':
25     app.run(debug=True)
26
```

Ce code crée un serveur web avec Flask qui envoie les requêtes à une liste prédéfinie de serveurs de manière cyclique. Lorsqu'une requête est reçue, le serveur sélectionne le prochain serveur dans la liste pour y envoyer la requête. Si une erreur survient lors de la requête, elle est renvoyée au client avec un code d'état approprié.

MISE EN ŒUVRE PRATIQUE AVEC PYTHON



3. Implimentation load balancer avec l'algorithme Least Response Time

```
● ● ●
1  from flask import Flask, request
2  import time
3
4  app = Flask(__name__)
5
6  # Dictionnaire pour stocker les temps de réponse de chaque serveur
7  response_times = {
8      "Server 1": 1,
9      "Server 2": 0.8,
10     "Server 3": 1.2
11 }
12
13 @app.route('/')
14 def load_balancer():
15     # Trouver le serveur avec le temps de réponse le plus bas
16     min_response_time = min(response_times.values())
17     selected_server = [server for server, time in response_times.items() if time == min_response_time][0]
18     return f"La requête a été dirigée vers : {selected_server}"
19
20 if __name__ == '__main__':
21     app.run(debug=True)
22
```

Ce code crée un serveur web avec Flask qui sélectionne le serveur avec le temps de réponse le plus bas parmi une liste pré définie de serveurs simulés. Lorsqu'une requête est reçue, le serveur renvoie un message indiquant vers quel serveur la requête a été dirigée.

CONCLUSION



Ce projet a exploré le concept de Load Balancing et son importance dans la gestion efficace du trafic réseau et des ressources des serveurs. En examinant différents algorithmes de répartition de charge, nous avons compris comment ils peuvent être adaptés pour répondre aux besoins spécifiques d'un environnement informatique. En mettant en œuvre un équilibrEUR de charge simple avec Flask, nous avons visualisé le fonctionnement pratique de ces mécanismes et leur capacité à améliorer les performances globales du système. Les tests réalisés ont confirmé l'efficacité de l'équilibrEUR de charge dans la gestion du trafic et l'optimisation des ressources, contribuant ainsi à une meilleure expérience utilisateur et à une disponibilité accrue des services. En intégrant ces concepts dans la conception des infrastructures informatiques, nous pouvons garantir des performances optimales et une haute disponibilité des applications en ligne.

vous pouvez consulter le code dans notre repo **GitHub** en suivant ce lien :

⚠ https://github.com/MOHAMEDBOUTALMAOUINE/Load_Balancer