

DÉVELOPPEMENT D'UN SERVICE DE LOAD BALANCER POUR UNE CALCULATRICE DISTRIBUÉE

Rapport de Projet

4ème année Ingénierie Informatique et Réseaux

Sous le thème :Virtualisation d'un service de Load Balancer en architecture client-serveur

Réalisé par :

Majjid Ayoub

 LINKEDIN

 WEBSITE

 NEWSLETTER

 PAYPAL

Encadré par :

Tuteur de l'école : Mostapha Zbak

DÉDICACE

Je dédie ce travail à ma famille, pour leur soutien inconditionnel, leur amour et leurs encouragements constants. Leur présence a été ma force tout au long de ce parcours académique. À mes parents, qui ont toujours cru en moi et m'ont poussé à poursuivre mes rêves, je vous suis infiniment reconnaissant.

À mes amis et collègues, qui ont partagé les hauts et les bas de cette aventure, merci pour votre amitié et votre soutien. Vos encouragements ont rendu ce projet plus enrichissant.

Enfin, à mes professeurs et mentors, dont les conseils et l'expertise ont guidé mes pas, je vous exprime ma profonde gratitude. Ce travail est le fruit de vos enseignements et de votre confiance.

REMERCIEMENTS

Nous tenons à exprimer notre gratitude à toutes les personnes qui ont contribué à la réalisation de ce projet.

Nos parents pour leur soutien inconditionnel et leur patience.

Nos camarades pour leurs idées et leur collaboration.

Notre encadrant pour ses conseils avisés et son accompagnement rigoureux.

Merci à tous pour votre aide précieuse.

RÉSUMÉ

Introduction :

Ce projet a pour objectif de concevoir un service de Load Balancer distribué pour une calculatrice en architecture client-serveur. Il implique la virtualisation de machines Ubuntu sous Hyper-V, la configuration réseau, et le développement d'APIs Flask pour la communication entre les nœuds.

Contexte :

Les architectures distribuées sont essentielles pour optimiser les performances et la disponibilité des services. Ce projet explore la répartition de charge entre un master et des slaves pour le calcul d'expressions mathématiques.

Objectifs :

- Virtualiser 3 machines (1 master, 2 slaves) sous Hyper-V.
- Configurer un réseau interne avec IPs statiques.
- Développer des APIs Flask pour la gestion des tâches.
- Tester le système avec Postman et valider la répartition de charge.

Résultats :

Le projet a abouti à une plateforme fonctionnelle où le master répartit les tâches de calcul entre les slaves. Les tests confirment la robustesse et l'efficacité du système.

Perspectives :

Améliorer la sécurité, ajouter des fonctionnalités de monitoring, et étendre le système à un plus grand nombre de slaves.

ABSTRACT

Introduction:

This project aims to develop a distributed Load Balancer service for a client-server calculator. It involves virtualization of Ubuntu machines under Hyper-V, network configuration, and Flask APIs for node communication.

Context:

Distributed architectures are key to optimizing performance and service availability. This project explores load balancing between a master and slaves for mathematical computations.

Objectives:

- Virtualize 3 machines (1 master, 2 slaves) using Hyper-V.
- Configure an internal network with static IPs.
- Develop Flask APIs for task management.

- Test the system with Postman and validate load distribution.

Results:

The project resulted in a functional platform where the master distributes tasks to slaves. Tests confirm system robustness and efficiency.

Future Work:

Enhance security, add monitoring features, and scale the system to more slaves.

TABLE DES MATIÈRES

1. [Introduction](#)
 2. [Objectifs du Projet](#)
 3. [Environnement et Outils](#)
 4. [Méthodologie](#)
 - [Création des Machines Virtuelles](#)
 - [Configuration Réseau](#)
 - [Développement des APIs Flask](#)
 - [Tests avec Postman](#)
 5. [Résultats et Analyse](#)
 6. [Conclusion](#)
 7. [Annexes](#)
-

LISTE DES ABRÉVIATIONS

- VM : Machine Virtuelle
 - API : Application Programming Interface
 - UML : Unified Modeling Language
 - HTTP : HyperText Transfer Protocol
-

INTRODUCTION

L'évolution des architectures distribuées a révolutionné la gestion des ressources informatiques. Ce projet explore la virtualisation et la répartition de charge à travers un service de calcul distribué.

Problématique :

Comment concevoir un système scalable et efficace pour répartir des tâches de calcul entre plusieurs nœuds ?

Organisation du Mémoire :

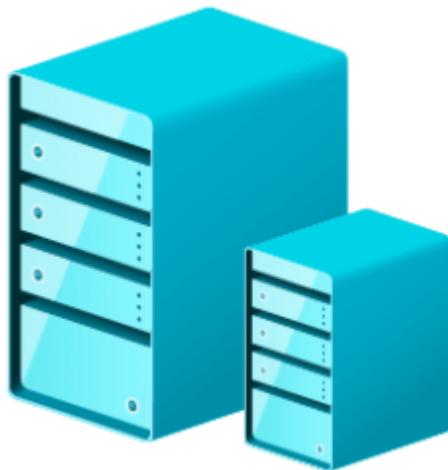
- Chapitre 1 : Présentation de l'environnement technique.
 - Chapitre 2 : Analyse et conception du système.
-

- Chapitre 3 : Réalisation et tests.
 - Chapitre 4 : Interfaces et résultats.
-

CHAPITRE 1 : PRÉSENTATION DE L'ENVIRONNEMENT

1.1 Outils Utilisés

1.1.1 Hyper-V



Solution de virtualisation utilisée pour créer les machines virtuelles master et slaves.

z

1.1.2 Ubuntu Server



Système d'exploitation choisi pour sa stabilité et son support des outils open-source.

1.1.3 Flask

Framework Python léger idéal pour le développement des microservices.

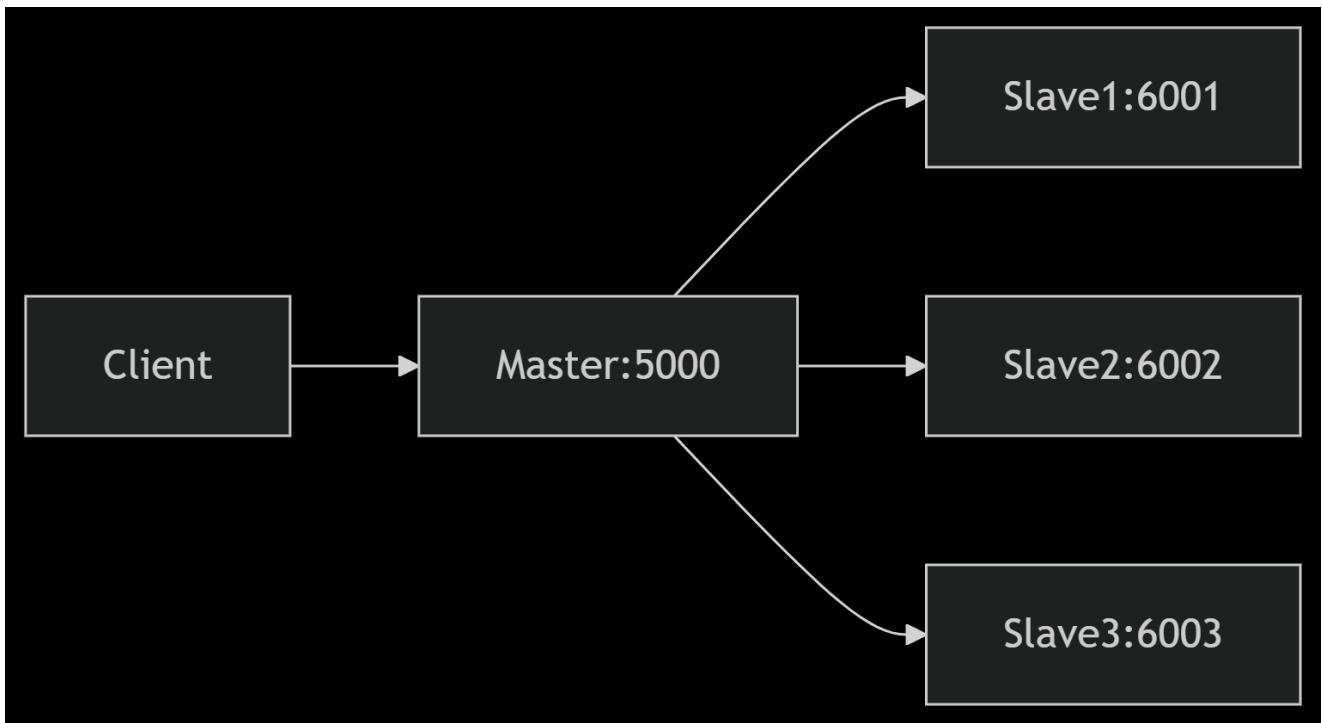
1.1.4 Postman

[API Documentation](#)



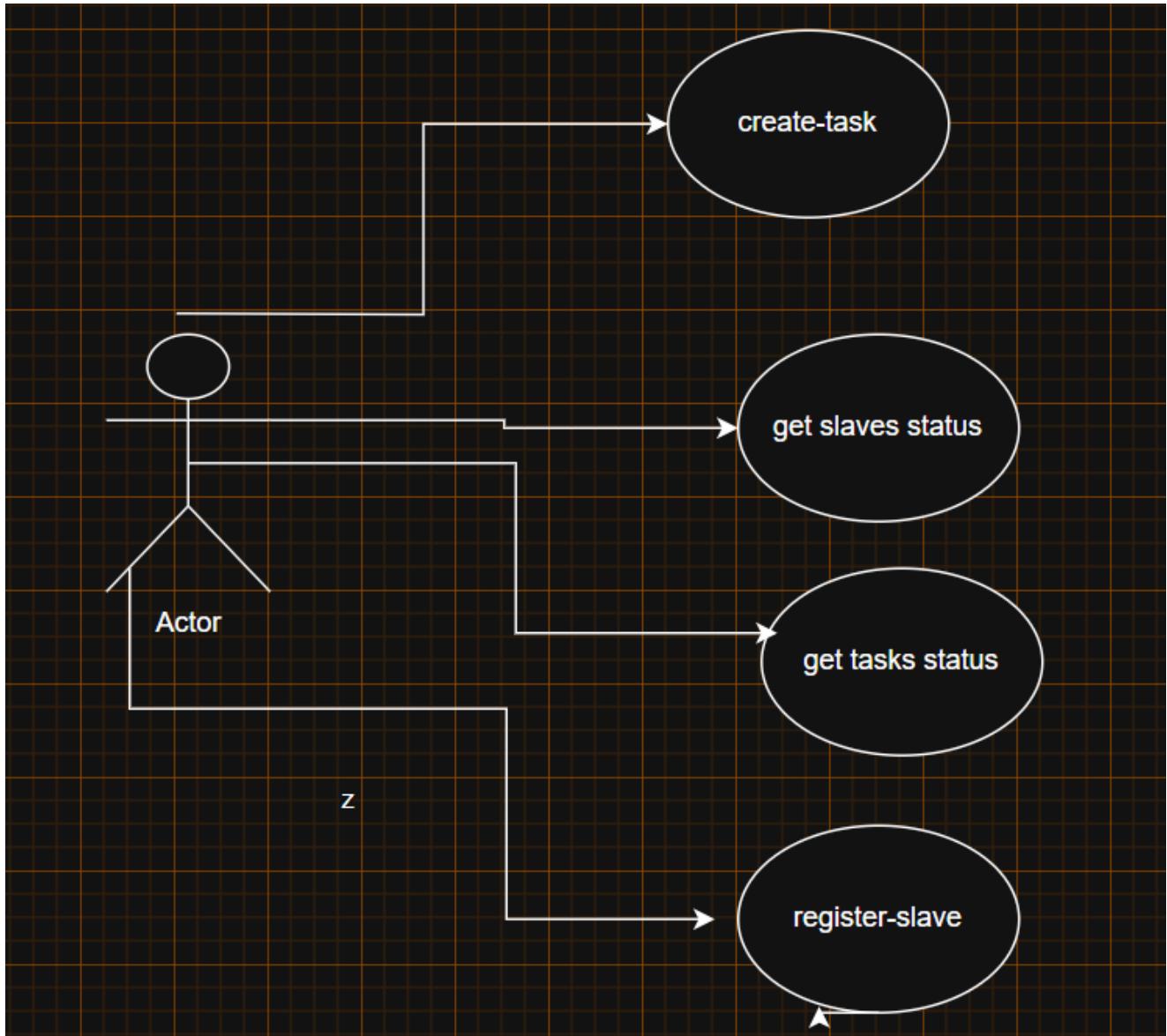
Outil indispensable pour valider les endpoints et documenter les APIs.

1.2 Architecture du Projet



CHAPITRE 2 : ANALYSE ET CONCEPTION

2.1 Diagrammes UML



2.2 Structure des Données

Le format JSON est utilisé pour envoyer et recevoir des informations entre les différents composants de l'architecture. Les requêtes envoyées au "master" contiennent des données sous forme de JSON qui spécifient les tâches à effectuer. Chaque tâche contient un identifiant unique, des informations sur l'opération à réaliser, ainsi que les données nécessaires pour effectuer le calcul.

Exemple de Requête JSON pour une Tâche :

```
{  
  "task_id": "12345",  
  "payload": {  
    "operation": "addition",  
    "operands": [5, 3]  
  }  
}
```

- **task_id** : Identifiant unique de la tâche.
- **operation** : Type d'opération à effectuer (par exemple, addition, soustraction, multiplication, etc.).
- **operands** : Liste des opérandes sur lesquels l'opération sera effectuée.

Exemple de Réponse JSON du "Slave" :

```
{  
  "task_id": "12345",  
  "result": 8,  
  "status": "completed"  
}
```

- **task_id** : Identifiant de la tâche correspondante.
- **result** : Résultat du calcul effectué par l'esclave.
- **status** : Statut de la tâche (par exemple, "completed" ou "failed").

Structure d'un Esclave (**slave**) :

```
{  
  "id": "slave-a1b2c3d4",  
  "webhook": "http://192.168.3.12:5000",  
  "status": "free"  
}
```

- **id** : Identifiant unique de l'esclave (UUID court).
- **webhook** : URL du point d'accès HTTP pour le traitement.
- **status** : Statut actuel de l'esclave (free ou busy).

CHAPITRE 3 : RÉALISATION

3.1 Développement des APIs

Endpoints :

- **/submit-task** : Soumission des expressions.
- **/register-slave** : Enregistrement des slaves.

3.2 Tests

Postman : Validation des fonctionnalités.

The screenshot shows the Postman interface with the following details:

- Collections:** master_slave_project
- Environments:** master
- POST /master/submit-task:** Request method is POST, URL is {{master-url}}/submit-task, Body is JSON with payload {"payload": "5x3-4"}, Response status is 201 CREATED with message "Task assigned to slave slave-4d1e44ad", taskID: "bd22d8f1-e8e9-404c-8534-843ed9d8e2f".

CHAPITRE 4 : INTERFACES ET MANUEL D'UTILISATION

4.1 Captures d'Écran

- **Master** : Interface de gestion des tâches.
- **Slaves** : Logs des calculs.

The screenshot shows the Proxmox VE interface with the following details:

- Virtual Machines:** A table listing four VMs: master, proxmox, slave1, and slave2, all in Off state.
- Checkpoints:** A section indicating "No virtual machine selected."

4.2 Manuel d'utilisation

Voici la documentation complète pour configurer votre projet de calculateur distribué Flask en utilisant un environnement virtuel Python ([venv](#)). Ce guide vous accompagne dans l'installation des dépendances, la configuration des environnements virtuels, l'exécution des services master et slaves, ainsi que les tests du système.

Structure du Projet

```
distributed_calculator/  
└── master.py
```

```
└── slave.py  
└── utils.py  
└── run_slaves.py  
└── requirements.txt  
└── README.md
```

📋 Prérequis

- Python 3.8 ou supérieur
 - pip
 - Connexion Internet pour installer les paquets
-

🔧 1. Créer et Activer l'Environnement Virtuel

Linux / MacOS :

```
python3 -m venv venv  
source venv/bin/activate
```

Windows (CMD) :

```
python -m venv venv  
venv\Scripts\activate
```

📦 2. Installer les Dépendances

Créer un fichier **requirements.txt** :

```
flask  
requests
```

Puis exécuter :

```
pip install -r requirements.txt
```

⚙️ 3. Configuration Importante Avant de Lancer les Slaves

Avant d'exécuter les slaves, vous devez **mettre à jour l'adresse IP du master** dans le fichier `slave.py` :

```
from threading import Thread

MASTER_URL = "http://<ip-de-votre-master>:5000" # Remplacez localhost par
l'adresse IP réelle du master
```

Assurez-vous que le master est en cours d'exécution et accessible à cette adresse.

⌚ 4. Lancer le Serveur Master

```
python master.py
```

```
(venv) slave1@master:~/client-server-architecture-project/code$ python3 master.py
* Serving Flask app 'master'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.3.10:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 576-917-555
192.168.3.11 -- [11/May/2025 21:36:44] "POST /register-slave HTTP/1.1" 200 -
192.168.3.13 -- [11/May/2025 21:38:48] "POST /register-slave HTTP/1.1" 200 -
192.168.3.4 -- [11/May/2025 21:39:15] "GET /tasks HTTP/1.1" 200 -
192.168.3.4 -- [11/May/2025 21:39:26] "GET /slaves HTTP/1.1" 200 -
192.168.3.4 -- [11/May/2025 21:45:42] "POST /submit-task HTTP/1.1" 201 -
```

Le serveur master est disponible à l'adresse `http://localhost:5000`.

📱 5. Lancer les Slaves

⊕ Option 1 : Lancer un seul Slave avec un port spécifique

```
python slave.py 6001
```

Dans ce cas, assurez-vous que `slave.py` accepte le port en argument :

```
import sys
WEBHOOK_PORT = int(sys.argv[1])
```

⚡ Option 2 : Lancer 3 slaves automatiquement (ports 6000, 6001, 6002)

Utilisez le script `run_slaves.py` pour démarrer trois instances simultanées :

```
python run_slaves.py
```

Cela lance trois slaves sur les ports 6000, 6001, et 6002 automatiquement, chacun s'enregistrant auprès du master défini.

✍ 6. Soumettre une Tâche

Utilisez Postman ou la commande `curl` suivante :

```
curl -X POST http://localhost:5000/submit-task \  
-H "Content-Type: application/json" \  
-d '{"payload": "4 * (2 + 3)"}'
```

☑ 7. Suivi et Supervision

- Consulter les tâches : <http://localhost:5000/tasks>
 - Voir les slaves enregistrés : <http://localhost:5000/slaves>
- ...
- [API Documentation](#)

✗ 8. Désactiver l'Environnement

```
deactivate
```

⌚ Temps de Réponse

Temps de réponse moyen : **~7 secondes** par tâche (selon la complexité de l'expression et la disponibilité des slaves).

CONCLUSION

Ce projet a permis de maîtriser :

- La virtualisation via Hyper-V
- L'architecture distribuée (master/slave)
- Le développement d'APIs Flask
- La gestion concurrente des tâches

Il constitue une base prometteuse pour évoluer vers des systèmes plus complexes et résilients.

ANNEXES

-  Code Source : Lien vers le dépôt GitHub
 -  Documentation API : Lien Postman
 -  Captures d'Écran : Interfaces Hyper-V, logs, réseau
-

WEBOGRAPHIE

- [Flask Documentation](#)
- [Microsoft Hyper-V](#)
- [Postman](#)