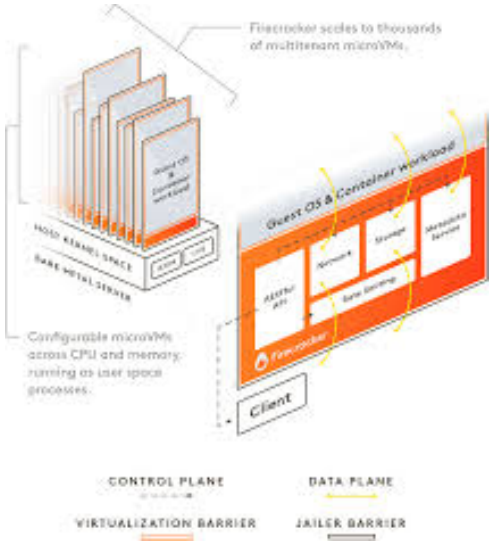
Rapport de Projet

Plateforme de Gestion d'Instances de Machines Virtuelles



Équipe de développement

ZOGO ABOUMA ZOZIME ACHAIRE	18N2824
Stephane Roylex NKOLO	21T2588
SOKOUDJOU CHENDJOU CHRISTIAN MANUEL	21T2396
MAHAMAT SALEH MAHAMAT	21T2423

25 juin 2025

Table des matières

Remerciements

Nous tenons à exprimer notre profonde gratitude à l'ensemble des personnes qui ont contribué de près ou de loin à la réalisation de ce projet.

Nos remerciements s'adressent tout particulièrement à nos encadrants pour leur précieux conseils, leur disponibilité et leur soutien constant tout au long de ce projet.

Nous remercions également l'ensemble des membres de l'équipe pour leur implication, leur travail acharné et leur esprit d'équipe qui ont été déterminants dans l'aboutissement de ce projet.

Chapitre 1

Introduction

1.1 Contexte du projet

Dans un monde de plus en plus numérique, la virtualisation est devenue un pilier fondamental de l'infrastructure informatique moderne. Les machines virtuelles (VMs) offrent une flexibilité inégalée pour le déploiement d'applications, le test de logiciels et l'optimisation des ressources matérielles. Ce projet s'inscrit dans cette dynamique en proposant une solution complète de gestion d'instances de machines virtuelles.

1.2 Objectifs du projet

L'objectif principal de ce projet est de développer une plateforme permettant :
— La création et la gestion simplifiée d'instances de machines virtuelles
— L'allocation dynamique des ressources
— La gestion des images système
— Le déploiement automatisé de configurations
— La surveillance et le suivi des performances

Chapitre 2

Architecture du système

2.1 Vue d'ensemble

Notre solution s'articule autour d'une architecture microservices, avec les composants principaux suivants :

- Service de gestion des clusters
- Service des images système
- Service des offres de machines virtuelles
- Service hôte des machines virtuelles
- Service de notification

2.2 Diagramme d'architecture

[Diagramme d'architecture à insérer]

Chapitre 3

Description des services

3.1 Service de gestion des clusters (service-cluster)

3.1.1 Responsabilités

— Gestion des clusters de machines hôtes
— Répartition de charge
— Surveillance de l'état des nœuds
— Gestion des utilisateurs et des permissions
— Orchestration des déploiements

3.1.2 Technologies utilisées

— Python 3.9+ avec FastAPI
— Base de données MySQL 8.0+
— Service Discovery avec Eureka
— Authentification JWT
— ORM SQLAlchemy

3.1.3 Points d'API Clés

```
# Créer un nouveau cluster
POST /api/service-cluster/clusters
{
    "name": "cluster-1",
    "description": "Cluster principal",
    "region": "eu-west-1"
}

# Lister les nœuds d'un cluster
GET /api/service-cluster/clusters/{cluster_id}/nodes

# Vérifier l'état du cluster
GET /api/service-cluster/health
```

3.2 Service des images système (service-system-image)

3.2.1 Responsabilités

— Gestion du catalogue d'images système
— Téléchargement et stockage des images
— Validation et vérification d'intégrité
— Gestion des versions d'images
— Conversion entre différents formats d'image

3.2.2 Technologies utilisées

— Python 3.9+ avec FastAPI
— Stockage objet (S3 compatible)
— Base de données PostgreSQL 13+
— Intégration avec Docker pour la gestion des conteneurs
— Système de cache Redis

3.2.3 Points d'API Clés

```
# Téléverser une nouvelle image
POST /api/service-system-image/images
Content-Type: multipart/form-data

# Lister les images disponibles
GET /api/service-system-image/images

# Télécharger une image spécifique
GET /api/service-system-image/images/{image_id}/download

# Vérifier l'intégrité d'une image
GET /api/service-system-image/images/{image_id}/verify
```

3.3 Service des offres de machines virtuelles (service-vm-offer)

3.3.1 Responsabilités

— Définition des configurations de VM
— Gestion des modèles de machines virtuelles
— Tarification et quotas
— Gestion des plans d'abonnement
— Suivi de l'utilisation des ressources

3.3.2 Technologies utilisées

— Python 3.9+ avec FastAPI

— Base de données PostgreSQL 13+
— Cache Redis pour les requêtes fréquentes
— Système de files d'attente pour le traitement asynchrone

3.3.3 Points d'API Clés

```
# Créer une nouvelle offre de VM
POST /api/service-vm-offer/offers
{
    "name": "small-vm",
    "cpu_cores": 2,
    "memory_mb": 4096,
    "disk_gb": 50,
    "price_per_hour": 0.05
}

# Lister les offres disponibles
GET /api/service-vm-offer/offers

# Mettre à jour les quotas utilisateur
PUT /api/service-vm-offer/users/{user_id}/quotas
```

3.4 Service hôte des machines virtuelles (service-vm-host)

3.4.1 Responsabilités

— Exécution des instances de VM via Firecracker
— Gestion complète du cycle de vie des VMs
— Surveillance en temps réel des ressources
— Gestion avancée du réseau et du stockage
— Isolation des ressources entre utilisateurs
— Gestion des snapshots et sauvegardes

3.4.2 Technologies clés

— Firecracker pour la virtualisation légère
— FastAPI pour l'API REST
— RabbitMQ pour la messagerie asynchrone
— Docker pour l'isolation des conteneurs
— Prometheus pour la surveillance
— Grafana pour la visualisation des métriques

3.4.3 Points d'API Clés

```
# Créer une nouvelle VM
POST /api/service-vm-host/vm/create
```

```
{
    "name": "ma-vm",
    "image_id": "ubuntu-22.04",
    "cpu_cores": 2,
    "memory_mb": 4096,
    "disk_gb": 50
}

# Démarrer/Arrêter une VM
POST /api/service-vm-host/vm/{vm_id}/start
POST /api/service-vm-host/vm/{vm_id}/stop

# Obtenir les métriques d'une VM
GET /api/service-vm-host/vm/{vm_id}/metrics

# Prendre un snapshot
POST /api/service-vm-host/vm/{vm_id}/snapshot
```

Chapitre 4

Spécifications techniques

4.1 Prérequis système

4.1.1 Exigences matérielles minimales

— CPU : 4 cœurs (8+ recommandés)
— RAM : 8 Go (16+ Go recommandés)
— Stockage : 100 Go d'espace disque (SSD recommandé)
— Accès réseau : Interface Ethernet Gigabit

4.1.2 Exigences logicielles

— Système d'exploitation : Linux (Ubuntu 20.04+ ou équivalent)
— Docker 20.10+ et Docker Compose 1.29+
— Python 3.9+
— MySQL 8.0+ ou PostgreSQL 13+
— Accès root pour la configuration réseau et système
— Privilèges de virtualisation activés (KVM)

4.2 Architecture du déploiement

4.2.1 Composants principaux

— **Load Balancer** : Nginx/Traefik
— **API Gateway** : FastAPI avec authentification
— **Base de données** : MySQL/PostgreSQL
— **Message Broker** : RabbitMQ
— **Cache** : Redis
— **Stockage** : Stockage objet S3-compatible
— **Surveillance** : Prometheus + Grafana
— **Logging** : ELK Stack (Elasticsearch, Logstash, Kibana)

4.3 Installation et déploiement

4.3.1 Configuration requise

— 4 Go de RAM minimum (8 Go recommandés)
— 20 Go d'espace disque disponible
— Accès à Internet pour le téléchargement des dépendances

4.3.2 Procédure d'installation

1. Cloner le dépôt du projet 2. Configurer les variables d'environnement 3. Lancer les conteneurs avec Docker Compose 4. Vérifier l'état des services

Chapitre 5

Manuel d'utilisation

5.1 Création d'une machine virtuelle

1. Sélectionner une image système 2. Choisir une configuration 3. Configurer le réseau 4. Lancer l'instance

5.2 Gestion des ressources

— Surveillance de l'utilisation CPU/Mémoire
— Gestion des disques
— Configuration réseau

Chapitre 6

Conclusion et perspectives

6.1 Bilan du projet

Ce projet a permis de mettre en œuvre une solution complète de gestion de machines virtuelles, en mettant l'accent sur la performance, la sécurité et la facilité d'utilisation. L'architecture microservices offre une grande évolutivité et maintenabilité.

6.2 Points forts

— Architecture modulaire et évolutive
— Haute performance grâce à Firecracker
— Gestion fine des ressources
— Interface d'API RESTful complète
— Surveillance et journalisation avancées

6.3 Améliorations possibles

6.3.1 Court terme

— Interface web de gestion
— Support de conteneurs Docker
— API GraphQL en complément de REST

6.3.2 Moyen terme

— Support de la haute disponibilité
— Réplication géographique
— Auto-scaling automatique

6.3.3 Long terme

— Intégration avec Kubernetes
— Support du multi-cloud
— IA pour l'optimisation des ressources

Annexes

6.4 Exemple de fichier docker-compose.yml

```yaml
version: '3.8'

services:
  api-gateway:
    image: nginx:latest
    ports:
      - "80:80"
      - "443:443"
    depends_on:
      - service-cluster
      - service-vm-host
      - service-system-image
      - service-vm-offer

  service-cluster:
    build: ./service-cluster
    environment:
      - DB_HOST=db
      - DB_USER=user
      - DB_PASSWORD=password
    depends_on:
      - db
      - redis

  # Autres services...

  db:
    image: postgres:13
    environment:
      - POSTGRES_PASSWORD=password
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:6

volumes:
  postgres_data:
```

6.5 Exemple de configuration réseau

```
# Configuration r seau pour une VM
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      addresses: [192.168.1.10/24]
      gateway4: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

6.6 Glossaire

VM Machine Virtuelle

API Interface de Programmation d'Application

CPU Unité Centrale de Traitement

RAM Mémoire Vive

6.7 Références

— Documentation officielle de Firecracker
— Documentation FastAPI
— Documentation Docker