



جامعة العلوم والتكنولوجيا هواري بومدين  
Université des Sciences et de la Technologie Houari  
Boumediene

## Faculté d’Informatique

### Mémoire de fin d’études

Pour l’obtention du diplôme de Master 2

Option : Réseaux et Systèmes Distribué

---

# Conception et implémentation d’une solution de déploiement et provisionnement automatiques sur des clusters physiques

---

*Réalisé par :*

M.SALMI Wail Abderrahim

M.BENMEDDOUR Mohamed Nafae

*Encadré par :*

M.ZERAOULIA Khaled

Mme.BENZAID Chafika

*Soutenu le 02 Juillet 2025, Devant le jury composé de :*

Mr.HAMMAL Youcef : - Président du jury  
Mr.DERDER Abdessamed : - Examinateur

Promotion : 2024/2025  
Code PFE : RSD-30

---

# Dédicace

---

66

*Premièrement El-hamdouliyah*

*Je dédie ce projet :*

*A ma chère mère,*

*A mon cher père,*

*Qui n'ont jamais cessé, de formuler des prières à mon  
égard, de me soutenir et de m'épauler pour que je puisse  
atteindre mes objectifs.*

*A mon frères , Oussama,*

*À mes chères sœurs,*

*Pour ses soutiens moral et leurs conseils précieux tout au  
long de mes études.*

*A mon ami Mustafa,*

*A mes amis de la cite universitaire et el-jama3,*

*A mes amis d'universite,*

*et à tous ceux qui nous ont aidés pour réaliser ce projet,*

*Merci.*

99

*Salmi Wail Abderrahim*

---

66

*Premièrement, je remercie Dieu de m'avoir guidé et soutenu tout au long de mon parcours d'études et en toutes choses.*

*C'est avec grand plaisir que je dédie ce modeste travail :*

*À mon père et ma mère,*

*À mes frères et sœurs,*

*À mes amis,*

*À mes professeurs*

*Et à tous ceux qui m'ont aidé pour réaliser ce projet.*

*Merci.*

77

*Benmeddour Mohamed Nafea*

---

# Remerciements

Tout d'abord, nous remercions Allah, le Tout-Puissant, de nous avoir donné le courage et la patience nécessaires pour mener ce travail à son terme.

Nous tenons à exprimer notre profonde gratitude à notre encadrant **M. ZERAOU-LIA Khaled** ainsi qu'à **Mme BENZAID Chafika**, pour leur aide précieuse, leur patience et leurs encouragements. Leurs regards critiques ont été inestimables pour la structuration de ce travail et l'amélioration de la qualité des divers aspects de notre projet.

Que les membres du jury trouvent ici l'expression de nos sincères remerciements pour l'honneur qu'ils nous font en prenant le temps de lire et d'évaluer notre travail.

Nous souhaitons également remercier l'équipe pédagogique et administrative de **l'USTHB** pour les efforts qu'elle déploie afin de nous offrir une formation de qualité.

Enfin, nous remercions toutes les personnes ayant contribué, de près ou de loin, à la réalisation de ce travail.

---

# Abstract

Les environnements cloud modernes nécessitent une gestion dynamique des ressources, mais un écart significatif subsiste entre l'intention humaine exprimée en langage naturel et le code bas niveau requis pour déployer l'infrastructure. Ce mémoire répond à ce défi en proposant la conception et la mise en œuvre d'un système autonome capable de traduire des commandes en langage naturel en actions de provisionnement entièrement automatisées sur un cluster physique.

La solution repose sur une architecture multi-couches. Elle débute par une base hyperconvergée composée de Proxmox VE pour la virtualisation et Ceph pour le stockage distribué. Par-dessus cette base résiliente, un cluster Kubernetes à haute disponibilité (k3s) est déployé et géré à l'aide d'un flux de travail Infrastructure as Code (IaC), utilisant Terraform pour le provisionnement et Ansible pour la configuration.

L'innovation principale réside dans la couche d'intelligence artificielle agentique, qui intègre un système multi-agents (MAS) alimenté par une intelligence artificielle générative. Cette couche intelligente interprète l'intention de l'utilisateur, analyse l'état du système, et génère dynamiquement le code de déploiement nécessaire.

La plateforme a été validée avec succès à travers des scénarios pratiques, notamment la création automatique d'une machine virtuelle à partir d'une commande en langage naturel. Cette validation démontre la viabilité de l'architecture proposée, prouvant qu'une intégration entre un MAS alimenté par une IA générative et une pile d'infrastructure automatisée peut efficacement combler le fossé entre l'intention humaine et l'exécution de l'infrastructure, ouvrant la voie à une gestion cloud plus intelligente, résiliente et autonome.

---

**Mots-clés :** Provisionnement cloud, Infrastructure as Code (IaC), Proxmox VE, Ceph, Kubernetes (k3s), système multi-agents (MAS), intelligence artificielle générative, traitement du langage naturel, virtualisation, infrastructure autonome.

---

# Table des matières

Dédicace	I
Remerciements	IV
Abstract	V
Table des matières	VI
Table des figures	X
Liste des tableaux	XIII
Introduction générale	1
<b>1 Optimisation de L'approvisionnement et du Dimensionnement dans les Clusters Virtualisés : Défis et Opportunités</b>	<b>3</b>
I.1 Concepts fondamentaux . . . . .	4
I.1.1 Pourquoi les environnements virtualisés ? . . . . .	4
I.1.2 Pourquoi les environnements conteneurisés ? . . . . .	5
I.1.3 L'utilisation des environnements virtualisés et conteneurisés pour l'approvisionnement et l'allocation des ressources . . . . .	5
I.2 Défis dans les environnements virtualisés et conteneurisés . . . . .	6
I.2.1 Équilibrer performance, disponibilité et coût dans l'approvisionnement en ressources . . . . .	6
I.2.2 Les limites de l'allocation statique des ressources et le besoin d'un approvisionnement dynamique . . . . .	6
I.2.3 La complexité des architectures multi-couches . . . . .	6
I.3 Synthèse . . . . .	7
II.1 Hyperviseurs et virtualisation . . . . .	9
II.1.1 Leur rôle dans la virtualisation et la gestion des ressources . . . . .	9
II.1.2 Types d'hyperviseurs et étude comparative . . . . .	9
II.2 Des hyperviseurs à l'orchestration : lever les limitations . . . . .	10
II.3 Orchestration et conteneurisation . . . . .	11
II.3.1 Le besoin de conteneurisation dans l'infrastructure moderne . . . . .	11

## Table des matières

---

II.3.2	Mise à l'échelle et planification dynamiques . . . . .	11
II.4	Le besoin d'IA dans les environnements orchestrés . . . . .	11
II.5	Mise à l'échelle avec l'IA : approvisionnement et optimisation prédictive de l'allocation des ressources . . . . .	12
1.3	Conclusion . . . . .	13
<b>2</b>	<b>Approvisionnement et Prédiction Intelligents des Ressources (SRPP)</b> . . . . .	<b>14</b>
2.1	introduction . . . . .	15
2.2	Approvisionnement et Prédiction Intelligents des Ressources (SRPP) Basés sur les Métaheuristiques . . . . .	15
2.2.1	Vue d'ensemble des algorithmes métahéuristiques . . . . .	15
2.2.2	Revue des travaux basés sur les métahéuristiques . . . . .	16
2.2.3	Analyse comparative des approches métahéuristiques . . . . .	20
2.2.4	Défis et lacunes de recherche . . . . .	20
2.3	Approvisionnement et Prédiction Intelligents des Ressources (SRPP) Basés sur le ML et le DL . . . . .	21
2.3.1	Vue d'ensemble des méthodes ML et DL . . . . .	21
2.3.2	Revue des travaux basés sur le ML et le DL . . . . .	23
2.3.3	Analyse comparative . . . . .	26
2.3.4	Limites et Lacunes de Recherche . . . . .	27
2.4	IA Générative pour l'Approvisionnement et la Prédiction Intelligents des Ressources (SRPP) . . . . .	27
2.4.1	Vue d'ensemble des méthodes d'IA Générative . . . . .	27
2.4.2	Revue des travaux basés sur l'IA Générative . . . . .	28
2.4.3	Cas d'usage et intégration pratique . . . . .	31
2.5	Systèmes Multi-Agents pour l'Approvisionnement et la Prédiction Intelligents des Ressources . . . . .	32
2.5.1	Vue d'ensemble des méthodes multi-agents . . . . .	32
2.5.2	Revue des travaux basés sur les systèmes multi-agents . . . . .	33
2.5.3	Analyse comparative . . . . .	36
2.5.4	Cas d'usage et intégration pratique . . . . .	36
2.6	Synthèse des Paradigmes pour l'Approvisionnement et la Prédiction Intelligents des Ressources . . . . .	37
2.7	Conclusion . . . . .	40
<b>3</b>	<b>Conception du système et vue d'ensemble architecturale</b> . . . . .	<b>41</b>
3.1	Introduction à la conception proposée . . . . .	42
3.2	Architecture conceptuelle . . . . .	42
3.3	Couches fondamentales de l'architecture proposée . . . . .	45

## Table des matières

---

3.3.1	Couche Physique . . . . .	45
3.3.2	Couche de Virtualisation . . . . .	46
3.3.3	Couche d'Orchestration . . . . .	52
3.3.4	Couche de Surveillance, Gestion et Automatisation . . . . .	56
3.3.5	Couche Agentique . . . . .	59
3.4	Flux de Charges de Travail et de Communication à Travers les Couches .	69
3.4.1	Synergie Entre les Couches . . . . .	69
3.4.2	Flux de Communication . . . . .	69
3.4.3	Intégration de l'IA Prédictive . . . . .	70
3.5	Conclusion . . . . .	71
<b>4</b>	<b>Implémentation Pratique</b>	<b>72</b>
4.1	Introduction . . . . .	73
4.2	Pile Technologique et Justification . . . . .	73
4.2.1	Sélection des Technologies de Base . . . . .	74
4.2.2	Justification des Choix . . . . .	75
4.3	Couche Fondamentale : Infrastructure Physique et Virtualisation . . . . .	80
4.3.1	Configuration Matérielle et Réseau . . . . .	80
4.3.2	Installation de Proxmox VE . . . . .	81
4.3.3	Configuration Réseau avec Open vSwitch . . . . .	83
4.3.4	Configuration du Cluster d'Hyperviseurs Proxmox VE . . . . .	84
4.3.5	Stockage Défini par Logiciel avec Ceph . . . . .	84
4.3.6	Réseautage Défini par Logiciel (SDN) et l'Appliance Virtuelle OPNsense . . . . .	86
4.3.7	Configuration de la Haute Disponibilité et de la Résilience du Système	89
4.4	Couche d'Orchestration de Conteneurs : Kubernetes Haute Disponibilité avec k3s . . . . .	93
4.4.1	Préparation du Template de VM . . . . .	93
4.4.2	Provisionnement Automatisé des Noeuds Kubernetes avec Terraform	94
4.4.3	Configuration et Déploiement du Cluster k3s HA avec Ansible .	95
4.5	Couche de Surveillance, de Gestion et d'Automatisation . . . . .	98
4.5.1	Le Flux de Travail Intégré IaC et d'Automatisation . . . . .	98
4.5.2	Surveillance Cloud-Native avec Prometheus et Grafana . . . . .	98
4.5.3	Interfaces de Gestion Centralisées . . . . .	99
4.6	La Couche d'IA Agentique . . . . .	100
4.6.1	Provisionnement du Serveur IA et Sélection du LLM . . . . .	100
4.6.2	Développement de l'Agent de Provisionnement Proxmox . . . . .	101
4.6.3	Interaction Utilisateur et Flux de Travail . . . . .	101
4.7	Validation de la Plateforme : Scénarios Intégrés . . . . .	102

## Table des matières

---

4.7.1	Scénario 1 : Mise à l’Échelle Pilotée par l’Automatisation . . . . .	103
4.7.2	Scénario 2 : Tolérance aux Pannes Multi-Couches . . . . .	104
4.7.3	Scénario 3 : Déploiement d’Application de Bout en Bout . . . . .	106
4.7.4	Scénario 4 : Provisionnement Intelligent avec la Couche d’IA Agentique . . . . .	108
4.8	Conclusion . . . . .	111
	<b>Conclusion et perspectives</b>	<b>112</b>
	<b>Bibliographie</b>	<b>115</b>
	<b>Annexes</b>	<b>122</b>
	<b>A PROXMOX</b>	<b>123</b>
A.1	Partitionnement d’un disque Proxmox en CLI (Post-Installation) . . . . .	123
A.1.1	Note de pré-installation . . . . .	123
A.1.2	Structure initiale du disque . . . . .	124
A.1.3	Repartitionnement avec gdisk . . . . .	125
A.1.4	Vérification après redémarrage . . . . .	126
A.2	Création d’un template de machine virtuelle dans Proxmox . . . . .	127
A.2.1	Prérequis . . . . .	127
A.2.2	Création manuelle du template (GUI & CLI) . . . . .	127
A.2.3	Création automatisée du template avec un script Bash . . . . .	132
A.3	Guide de configuration du cluster Ceph . . . . .	135
A.3.1	Étape 1 : Initialisation du cluster et quorum des moniteurs . . . . .	135
A.3.2	Étape 2 : Création des Daemons de Stockage d’Objets (OSDs) . . . . .	136
A.3.3	Étape 3 : Provisionnement des pools de stockage et des systèmes de fichiers . . . . .	136
A.3.4	Conclusion . . . . .	137
	<b>B OPNsense</b>	<b>139</b>
B.1	Installation et configuration d’OPNsense . . . . .	139
B.1.1	Périmètre de la configuration d’OPNsense . . . . .	139
B.1.2	Configuration finale des services . . . . .	140
	<b>C Diagrammes d’Architecture de l’Implémentation</b>	<b>143</b>
C.1	Architecture Globale de l’Implémentation . . . . .	143
C.2	Flux de Travail de la Couche d’IA Agentique . . . . .	145

# Table des figures

I.1	Architecture multi-couches [1] . . . . .	7
II.2	Hyperviseurs de type 1 et type 2 [2] . . . . .	10
2.1	Techniques Métaheuristiques [3] . . . . .	16
3.1	Architecture globale . . . . .	43
3.2	Couche-Physique . . . . .	45
3.3	Couche-de-Virtualisation . . . . .	47
3.4	Couche-d'Orchestration-de-Conteneurs . . . . .	53
3.5	Couche de Surveillance, Gestion et Automatisation . . . . .	57
3.6	Coordination du Flux de Travail Agentique dans le Système Multi-Agents (SMA) . . . . .	61
3.7	Agent 1 - Flux de Travail du Validateur de Chat et Collecteur de Données . . . . .	62
3.8	Agent 2 - Flux de Travail de Formulation d'Analyse et de Stratégie . . . . .	65
3.9	Agent 3 - Flux de Travail de Construction et de Raffinement . . . . .	67
4.1	Options du disque dur Proxmox pour le dimensionnement des partitions . . . . .	82
4.2	Exemple de tableau de bord de l'interface web Proxmox après l'installation . . . . .	83
4.3	Aperçu de la configuration réseau avec le pont OVS . . . . .	83
4.4	Résumé du Datacenter Proxmox après la formation du cluster . . . . .	84
4.5	Disposition finale du disque avec la nouvelle partition pour Ceph . . . . .	85
4.6	Le tableau de bord du cluster Ceph . . . . .	86
4.7	Configuration de la zone SDN <b>mainzone</b> . . . . .	87
4.8	Définition du <b>mainvnet</b> et de son sous-réseau associé 192.168.0.0/16. . . . .	87
4.9	Configuration matérielle Proxmox pour la VM OPNsense . . . . .	88
4.10	Flux logique du trafic réseau . . . . .	88
4.11	Le tableau de bord d'OPNsense montrant l'état opérationnel final . . . . .	89
4.12	La configuration du groupe <b>mainHA</b> . . . . .	90
4.13	L'écran d'état du gestionnaire HA . . . . .	91
4.14	Création du <b>Backup-vms-pool</b> . . . . .	91
4.15	Le <b>Backup-vms-pool</b> entièrement peuplé . . . . .	92
4.16	Le calendrier de sauvegarde de Proxmox . . . . .	93
4.17	La configuration matérielle de la VM worker-node-2 provisionnée . . . . .	95

## Table des figures

---

4.18 La configuration Cloud-Init appliquée à une VM provisionnée . . . . .	95
4.19 Le fichier <code>hosts.ini</code> . . . . .	96
4.20 Le fichier <code>k3s_cluster.yaml</code> . . . . .	96
4.21 La sortie de <code>kubectl get nodes</code> . . . . .	97
4.22 la pile de surveillance Prometheus et Grafana . . . . .	99
4.23 Le tableau de bord Grafana pour le <code>k3s_cluster</code> . . . . .	99
4.24 L'architecture du Système de Provisionnement Intelligent. . . . .	101
4.25 Un exemple d'interaction avec l'IA Agentique . . . . .	102
4.26 La vue des serveurs du cluster Proxmox avant (gauche) et après (droite) la mise à l'échelle de trois à quatre nœuds physiques. . . . .	103
4.27 L'état des nœuds du cluster Kubernetes après l'opération de mise à l'échelle automatisée . . . . .	104
4.28 Basculement de Proxmox HA en action. La VM <code>worker-node-3</code> est auto- matiquement migrée de l'hôte défaillant <code>pmax04</code> (gauche) vers l'hôte sain <code>pmax01</code> (droite). . . . .	105
4.29 État du cluster Kubernetes lors d'une panne du plan de contrôle . . . . .	106
4.30 Les pods et services applicatifs restent entièrement opérationnels pendant la panne du nœud du plan de contrôle. . . . .	106
4.31 La sortie de <code>kubectl get services</code> . . . . .	107
4.32 Accès réussi à l'application web via son IP de LoadBalancer. . . . .	108
4.33 L'interprétation par l'agent de la requête de l'utilisateur et la demande de validation subséquente. . . . .	109
4.34 L'agent expliquant son processus de raisonnement en plusieurs étapes. .	109
4.35 L'exécution réussie du fichier Terraform généré par l'agent. . . . .	110
4.36 La vue du serveur Proxmox avant (gauche) et après (droite) le flux de travail de provisionnement intelligent . . . . .	110
A.1 Définition de l'option <code>hdszie</code> dans l'installateur Proxmox. . . . .	124
A.2 Téléversement de l'image Cloud Ubuntu. . . . .	128
A.3 Créer une VM sans média ISO. . . . .	129
A.4 Activer l'agent QEMU dans la fenêtre Système. . . . .	130
A.5 Supprimer le disque dur par défaut de la VM. . . . .	131
A.6 Personnaliser les paramètres de Cloud-Init. . . . .	132
A.7 Création de moniteurs Ceph supplémentaires pour le quorum . . . . .	135
A.8 Création d'un Daemon de Stockage d'Objets (OSD) Ceph sur la partition préparée . . . . .	136
A.9 Création et configuration du pool répliqué 'pmaxpool01' . . . . .	137
A.10 Création du système de fichiers CephFS . . . . .	137
B.1 Aperçu des interfaces OPNsense . . . . .	140

## Table des figures

---

B.2 Configuration de la passerelle dans OPNsense . . . . .	141
B.3 Configuration du serveur DHCP pour l'interface LAN . . . . .	141
B.4 Le tableau de bord OPNsense montrant l'état opérationnel final . . . . .	142
C.1 Un diagramme détaillé du système final implémenté . . . . .	144
C.2 Le flux de travail étape par étape du Système Multi-Agents pour le provi- sionnement sur Proxmox. . . . .	146

# Liste des tableaux

2.1	Summary of Surveyed Metaheuristic-Based Works . . . . .	18
2.2	Summary of Surveyed ML and DL-Based Works . . . . .	25
2.3	Summary of Surveyed Generative AI-Based Works . . . . .	30
2.4	Summary of Surveyed Multi Agent-Based Works . . . . .	35
4.1	Pile Technologique de Base . . . . .	74
4.1	Pile Technologique de Base . . . . .	75
4.2	Analyse Comparative des Plateformes d’Hyperviseurs . . . . .	76
4.3	Analyse Comparative des Plateformes d’Orchestration de Conteneurs . . . . .	77
4.4	Spécifications des Serveurs Physiques . . . . .	81
4.5	Configuration Réseau des Nœuds . . . . .	81

# Liste des sigles et acronymes

<b>ACO</b>	<i>Ant Colony Optimization</i>
<b>ADE</b>	<i>Adaptive Differential Evolution</i>
<b>ADK</b>	<i>Agent Development Kit</i>
<b>AMD-V</b>	<i>AMD Virtualization</i>
<b>ANN</b>	<i>Artificial Neural Network</i>
<b>API</b>	<i>Application Programming Interface</i>
<b>ARP</b>	<i>Address Resolution Protocol</i>
<b>AWS</b>	<i>Amazon Web Services</i>
<b>BG-LSTM</b>	<i>BiLSTM and GridLSTM</i>
<b>BGP</b>	<i>Border Gateway Protocol</i>
<b>CBRS</b>	<i>Content-based Recommender System</i>
<b>CephFS</b>	<i>Ceph File System</i>
<b>CI/CD</b>	<i>Continuous Integration/Continuous Deployment</i>
<b>CILP</b>	<i>Co-simulation-Based Imitation Learner</i>
<b>CNN</b>	<i>Convolutional Neural Network</i>
<b>CPU</b>	<i>Central Processing Unit</i>
<b>CRD</b>	<i>Custom Resource Definition</i>
<b>CRI-O</b>	<i>Container Runtime Interface - Open</i>
<b>CRS</b>	<i>Collaborative Recommender System</i>
<b>CSI</b>	<i>Container Storage Interface</i>
<b>DHCP</b>	<i>Dynamic Host Configuration Protocol</i>

## Liste des tableaux

---

<b>DL</b>	<i>Deep Learning</i>
<b>DNS</b>	<i>Domain Name System</i>
<b>DQN</b>	<i>Deep Q-Network</i>
<b>DRS</b>	<i>Distributed Resource Scheduler</i>
<b>DSL</b>	<i>Domain Specific Language</i>
<b>DT</b>	<i>Decision Tree</i>
<b>ECS</b>	<i>Elastic Container Service</i>
<b>ESXi</b>	<i>VMware vSphere Hypervisor</i>
<b>ETCD</b>	<i>Distributed Key-Value Store</i>
<b>ETL</b>	<i>Extract, Transform, Load</i>
<b>EULA</b>	<i>End User License Agreement</i>
<b>FQDN</b>	<i>Fully Qualified Domain Name</i>
<b>GA</b>	<i>Genetic Algorithm</i>
<b>GAN</b>	<i>Generative Adversarial Network</i>
<b>GCD</b>	<i>Google Cluster Dataset</i>
<b>GenAI</b>	<i>Generative Artificial Intelligence</i>
<b>GPU</b>	<i>Graphics Processing Unit</i>
<b>GRU</b>	<i>Gated Recurrent Unit</i>
<b>HA</b>	<i>High Availability</i>
<b>HCI</b>	<i>Hyper-Converged Infrastructure</i>
<b>HDD</b>	<i>Hard Disk Drive</i>
<b>HPA</b>	<i>Horizontal Pod Autoscaler</i>
<b>HRS</b>	<i>Hybrid Recommender System</i>
<b>HSOS-SA</b>	<i>Hybrid Social Spider Optimization - Simulated Annealing</i>
<b>HTTP</b>	<i>Hypertext Transfer Protocol</i>
<b>HTTPS</b>	<i>Hypertext Transfer Protocol Secure</i>
<b>IaC</b>	<i>Infrastructure as Code</i>

## Liste des tableaux

---

<b>IaaS</b>	<i>Infrastructure as a Service</i>
<b>IAM</b>	<i>Identity and Access Management</i>
<b>IMARM</b>	<i>Intelligent Multi-Agent Reinforcement Learning Model</i>
<b>IOMMU</b>	<i>Input-Output Memory Management Unit</i>
<b>IoT</b>	<i>Internet of Things</i>
<b>IP</b>	<i>Internet Protocol</i>
<b>ISO</b>	<i>International Organization for Standardization</i>
<b>JSON</b>	<i>JavaScript Object Notation</i>
<b>K8s</b>	<i>Kubernetes</i>
<b>KBRS</b>	<i>Knowledge Based Recommender System</i>
<b>KVM</b>	<i>Kernel-based Virtual Machine</i>
<b>LAN</b>	<i>Local Area Network</i>
<b>LIME</b>	<i>Local Interpretable Model-Agnostic Explanations</i>
<b>LLM</b>	<i>Large Language Model</i>
<b>LSTM</b>	<i>Long Short-Term Memory</i>
<b>LTS</b>	<i>Long Term Support</i>
<b>LVM</b>	<i>Logical Volume Manager</i>
<b>MAC</b>	<i>Media Access Control</i>
<b>MADQN</b>	<i>Multi-Agent Deep Q-Network</i>
<b>MAE</b>	<i>Mean Absolute Error</i>
<b>MAPE</b>	<i>Mean Absolute Percentage Error</i>
<b>MARL</b>	<i>Multi-Agent Reinforcement Learning</i>
<b>MAS</b>	<i>Multi-Agent System</i>
<b>MDS</b>	<i>Metadata Server</i>
<b>ML</b>	<i>Machine Learning</i>
<b>MON</b>	<i>Monitor</i>
<b>NAS</b>	<i>Network Attached Storage</i>

## Liste des tableaux

---

<b>NAT</b>	<i>Network Address Translation</i>
<b>NIC</b>	<i>Network Interface Card</i>
<b>NVMe</b>	<i>Non-Volatile Memory Express</i>
<b>OS</b>	<i>Operating System</i>
<b>OSD</b>	<i>Object Storage Daemon</i>
<b>OVS</b>	<i>Open vSwitch</i>
<b>PaaS</b>	<i>Platform as a Service</i>
<b>PCA</b>	<i>Principal Component Analysis</i>
<b>PSO</b>	<i>Particle Swarm Optimization</i>
<b>QoS</b>	<i>Quality of Service</i>
<b>RAG</b>	<i>Retrieval Augmented Generation</i>
<b>RAM</b>	<i>Random Access Memory</i>
<b>RBD</b>	<i>RADOS Block Device</i>
<b>ReAct</b>	<i>Reasoning and Acting</i>
<b>REST</b>	<i>Representational State Transfer</i>
<b>RF</b>	<i>Random Forest</i>
<b>RMSE</b>	<i>Root Mean Square Error</i>
<b>RNN</b>	<i>Recurrent Neural Network</i>
<b>RPO</b>	<i>Recovery Point Objective</i>
<b>SDS</b>	<i>Software-Defined Storage</i>
<b>SDN</b>	<i>Software-Defined Networking</i>
<b>SHAP</b>	<i>SHapley Additive exPlanations</i>
<b>SLA</b>	<i>Service Level Agreement</i>
<b>SRE</b>	<i>Site Reliability Engineering</i>
<b>SRPP</b>	<i>Smart Resource Provisioning and Prediction</i>
<b>SR-IOV</b>	<i>Single Root I/O Virtualization</i>
<b>SSD</b>	<i>Solid State Drive</i>

## Liste des tableaux

---

<b>SSH</b>	<i>Secure Shell</i>
<b>SVM</b>	<i>Support Vector Machine</i>
<b>TSDB</b>	<i>Time Series Database</i>
<b>USB</b>	<i>Universal Serial Bus</i>
<b>VAE</b>	<i>Variational Autoencoder</i>
<b>vCPU</b>	<i>Virtual Central Processing Unit</i>
<b>VE</b>	<i>Virtual Environment</i>
<b>VGA</b>	<i>Video Graphics Array</i>
<b>VIP</b>	<i>Virtual IP</i>
<b>VLAN</b>	<i>Virtual Local Area Network</i>
<b>VM</b>	<i>Virtual Machine</i>
<b>VMM</b>	<i>Virtual Machine Monitor</i>
<b>vNIC</b>	<i>Virtual Network Interface Card</i>
<b>VNet</b>	<i>Virtual Network</i>
<b>VNI</b>	<i>VXLAN Network Identifier</i>
<b>VPN</b>	<i>Virtual Private Network</i>
<b>vRAM</b>	<i>Virtual Random Access Memory</i>
<b>VT-x</b>	<i>Intel Virtualization Technology</i>
<b>VTGAN</b>	<i>Variational Transformer Generative Adversarial Network</i>
<b>WAN</b>	<i>Wide Area Network</i>
<b>WGAN-gp</b>	<i>Wasserstein Generative Adversarial Network with Gradient Penalty</i>
<b>YAML</b>	<i>Yet Another Markup Language</i>
<b>ZFS</b>	<i>Zettabyte File System</i>

# **General introduction**

Dans le monde numérique d'aujourd'hui, le cloud computing est devenu l'épine dorsale de l'infrastructure informatique moderne, offrant une scalabilité et une flexibilité sans précédent. Les entreprises et les chercheurs s'appuient sur les plateformes cloud pour héberger tout, des sites web simples aux applications complexes et gourmandes en données. Cependant, cette puissance s'accompagne d'une augmentation significative de la complexité. La gestion des charges de travail dynamiques, la garantie d'une haute disponibilité et l'optimisation de l'utilisation des ressources exigent une attention constante et une expertise technique approfondie. Les méthodes traditionnelles et manuelles de déploiement et de gestion de ces ressources sont souvent lentes, sujettes à l'erreur humaine et peinent à suivre les exigences rapides des applications modernes.

Le défi central que cette thèse aborde est le temps que prennent le déploiement et le provisionnement, ainsi que le fossé entre l'intention humaine et les commandes techniques nécessaires pour gérer un environnement cloud. Un développeur pourrait vouloir « déployer un nouveau serveur web avec trois instances redondantes », mais la traduction de cette simple requête en réalité implique une longue séquence de tâches spécifiques : provisionner des machines virtuelles, configurer des réseaux, mettre en place des répartiteurs de charge et appliquer des politiques de sécurité. Ce processus est non seulement fastidieux, mais il crée également une barrière à des opérations efficaces et agiles.

Pour combler ce fossé, cette thèse propose et met en œuvre une plateforme cloud multi-couches et autonome, conçue pour comprendre les requêtes en langage naturel et les traduire en ressources entièrement déployées. La fondation du système repose sur des technologies de virtualisation et de conteneurisation standard de l'industrie, notamment Proxmox VE et Kubernetes, pour créer une infrastructure robuste et résiliente. L'innovation clé, cependant, est la couche d'IA agentique (Agentic AI Layer) construite par-dessus. Cette couche utilise une équipe d'agents IA coopératifs, alimentés par l'IA générative, pour interpréter les objectifs de l'utilisateur, analyser l'état actuel du cluster et générer automatiquement le code de configuration nécessaire pour exécuter la requête.

Cette thèse est structurée pour guider le lecteur à travers l'ensemble du processus de recherche et de développement. Elle commence par :

## **Introduction générale**

---

Le premier chapitre, « **Optimisation du Provisionnement et de la Mise à l'Échelle dans les Clusters Virtualisés** », introduit le contexte et les défis de la gestion des infrastructures cloud. Ici, nous présentons les concepts fondamentaux de la virtualisation, de la conteneurisation et de l'orchestration, qui forment la base de notre étude.

Le deuxième chapitre, « **Provisionnement Intelligent de Ressources et Pré-diction (SRPP)** », constitue une étude approfondie des approches existantes pour l'automatisation intelligente. Nous analysons diverses techniques, y compris les métameuristiques, l'apprentissage automatique (ML/DL), l'IA générative (GenAI) et les systèmes multi-agents (SMA), afin de positionner notre contribution.

Le troisième chapitre, « **Conception du Système et Vue d'Ensemble de l'Architecture** », présente notre solution proposée en détail. Nous décrivons l'architecture à cinq couches, de la couche physique à la couche d'orchestration. L'accent est mis sur la conception de la couche d'IA agentique, un système multi-agents qui interprète les requêtes en langage naturel pour piloter l'infrastructure.

Le quatrième chapitre, « **Implémentation Pratique** », est consacré à la mise en œuvre concrète de notre architecture. Nous détaillons la pile technologique utilisée, incluant Proxmox VE, Ceph, Kubernetes (k3s), Terraform et Ansible. Ce chapitre présente également les scénarios de validation qui démontrent la résilience, la scalabilité et l'intelligence de la plateforme, en particulier sa capacité à provisionner des ressources à partir d'une commande en langage naturel.

Nous conclurons cette thèse par une « **Conclusion Générale** » et une discussion sur les « **Travaux Futurs** ».

## Chapitre 1

# Optimisation de L'approvisionnement et du Dimensionnement dans les Clusters Virtualisés : Défis et Opportunités

# 1.1 Environnements virtualisés et conteneurisés

## I.1 Concepts fondamentaux

L'informatique moderne repose de manière cruciale sur la virtualisation et la containerisation comme technologies de base, notamment dans l'informatique en nuage et les environnements d'entreprise. Par virtualisation, on entend le processus de création d'une version virtuelle de ressources telles que les serveurs de stockage et les réseaux, ce qui permet l'exécution simultanée de plusieurs systèmes d'exploitation et applications sur une seule machine. La containerisation, quant à elle, consiste à encapsuler des applications et leurs dépendances dans de petites unités portables pouvant fonctionner de manière cohérente à travers différents environnements. Ces technologies constituent une composante essentielle du paysage numérique actuel en raison de leur capacité à organiser et maximiser l'utilisation des ressources, à améliorer l'efficacité, à réduire les coûts tout en augmentant la flexibilité opérationnelle [4][5].

### I.1.1 Pourquoi les environnements virtualisés ?

La virtualisation est une technologie sur laquelle reposent les infrastructures informatiques modernes pour transformer la gestion des ressources, en particulier dans l'informatique en nuage et les environnements d'entreprise. Ce processus inclut l'exécution de plusieurs serveurs virtuels sur un seul hôte physique, c'est-à-dire que la virtualisation permet à une machine d'exécuter plusieurs machines virtuelles (VMs), chacune hébergeant des applications distinctes, ce qui augmente la flexibilité et la rentabilité. Grâce à la virtualisation, les organisations peuvent maximiser l'utilisation du matériel tout en réduisant les coûts énergétiques et en économisant de l'espace. Et dans les centres de données distribués actuels, cette technologie est fondamentale pour prendre en charge des charges de travail dynamiques et permettre une évolutivité rapide. Son efficacité constitue la base des systèmes cloud et d'entreprise modernes grâce à leur capacité à équilibrer performance, coût et efficacité des ressources à travers différents environnements [6].

### **I.1.2 Pourquoi les environnements conteneurisés ?**

Alors que la virtualisation traditionnelle repose sur des hyperviseurs pour gérer des systèmes d'exploitation entiers, la conteneurisation offre, quant à elle, une alternative plus fluide. Pour un développement plus rapide et une mise à l'échelle plus efficace, les conteneurs encapsulent les applications et leurs dépendances dans des unités isolées. Contrairement aux machines virtuelles, les conteneurs partagent le noyau du système hôte, ce qui minimise la surcharge et accélère le démarrage. Ce potentiel rend la conteneurisation vitale dans les architectures à base de microservices et les pratiques DevOps, où l'itération rapide et le déploiement continu sont essentiels [7].

### **I.1.3 L'utilisation des environnements virtualisés et conteneurisés pour l'approvisionnement et l'allocation des ressources**

L'adoption généralisée de la virtualisation est bien documentée dans divers secteurs. En raison des services que la virtualisation et la conteneurisation offrent — tels que l'amélioration de l'utilisation des ressources, la flexibilité et les économies de coûts — elles sont privilégiées aussi bien par les fournisseurs de services cloud que par les départements informatiques des entreprises. Par exemple, les principales plateformes de cloud public exploitent la virtualisation pour allouer les ressources à la demande. Quant aux entreprises, elles utilisent ces technologies pour soutenir les stratégies de cloud hybride et les initiatives de transformation numérique. L'importance de la virtualisation pour répondre aux exigences des charges de travail modernes est également renforcée par des tendances telles que l'informatique sans serveur (serverless) et l'informatique en périphérie (edge computing) [8].

La combinaison de la virtualisation et de la conteneurisation crée un environnement qui maximise les avantages des deux approches. Dans les environnements conteneurisés virtualisés, les machines virtuelles hébergent des plateformes de conteneurs, ce qui permet de détacher efficacement les dépendances entre le système d'exploitation et les applications. Autrement dit, cela permet de les découpler. Grâce à cette architecture à double couche, la répartition des ressources est optimisée et la flexibilité renforcée. La gestion du développement, de la mise à l'échelle et de la planification des conteneurs à travers les clusters est un rôle crucial aujourd'hui rempli par les outils d'automatisation et d'orchestration, tels que Kubernetes. Les applications concrètes incluent les déploiements à grande échelle de microservices (Les microservices constituent un style d'architecture dans lequel les applications complexes sont décomposées en une collection de services faiblement couplés et déployables indépendamment. Chaque service encapsule une fonctionnalité métier spécifique et communique avec les autres via des protocoles légers. Pour plus de détails, voir [9]) ainsi que les pipelines d'intégration et de déploiement continus

(CI/CD), où l'approvisionnement dynamique garantit une allocation des ressources en temps réel pour répondre aux besoins des charges de travail [10].

## **I.2 Défis dans les environnements virtualisés et containerisés**

### **I.2.1 Équilibrer performance, disponibilité et coût dans l'approvisionnement en ressources**

Optimiser la performance, la disponibilité et le coût lors de l'approvisionnement constitue un problème majeur auquel sont confrontées les organisations. Car une sous-allocation peut engendrer des goulets d'étranglement et une baisse de performance, tandis qu'une sur-allocation conduit à un gaspillage des ressources et une augmentation des coûts opérationnels. Un approvisionnement efficace doit équilibrer ces facteurs, en veillant à ce que les systèmes restent réactifs même en période de forte charge, sans entraîner de dépenses inutiles. Des stratégies avancées de gestion des ressources, incluant la surveillance en temps réel et la mise à l'échelle adaptative, sont essentielles pour maintenir cet équilibre [11].

### **I.2.2 Les limites de l'allocation statique des ressources et le besoin d'un approvisionnement dynamique**

L'allocation traditionnelle des ressources, qui repose souvent sur des méthodes statiques, attribue des quantités fixes de CPU, de mémoire et de stockage en fonction de la demande prévue. Par conséquent, dans des environnements où les charges de travail fluctuent, cette approche se révèle intrinsèquement inefficace. Les techniques d'approvisionnement dynamique, rendues possibles par l'utilisation de mécanismes d'échelle pilotés par l'IA, offrent une solution plus réactive. Les algorithmes d'apprentissage automatique prédisent les modèles de demande à partir de données historiques, permettant des ajustements proactifs dans l'allocation des ressources. La sur-allocation et la sous-allocation sont toutes deux minimisées grâce à cette approche adaptative, ce qui conduit en fin de compte à des infrastructures cloud plus efficaces et plus résilientes [12].

### **I.2.3 La complexité des architectures multi-couches**

Les architectures multi-couches, englobant le matériel, les hyperviseurs, les conteneurs et les charges de travail applicatives, caractérisent les infrastructures informatiques mo-

dernes. Chaque couche présente son propre ensemble de complexités, allant de la gestion des performances matérielles à la garantie du bon fonctionnement des environnements virtuels sur les hyperviseurs, en passant par l'exécution efficace des conteneurs au sein des VMs. Cette complexité en couches engendre des problèmes de coordination des ressources et accroît la charge administrative. Cependant, de nouveaux outils et cadres sont en cours de développement pour rationaliser la gestion entre ces couches, en intégrant des plateformes d'orchestration avec des analyses pilotées par l'IA afin d'optimiser l'utilisation des ressources à tous les niveaux de la pile [13]. La figure I.1 illustre les trois couches.

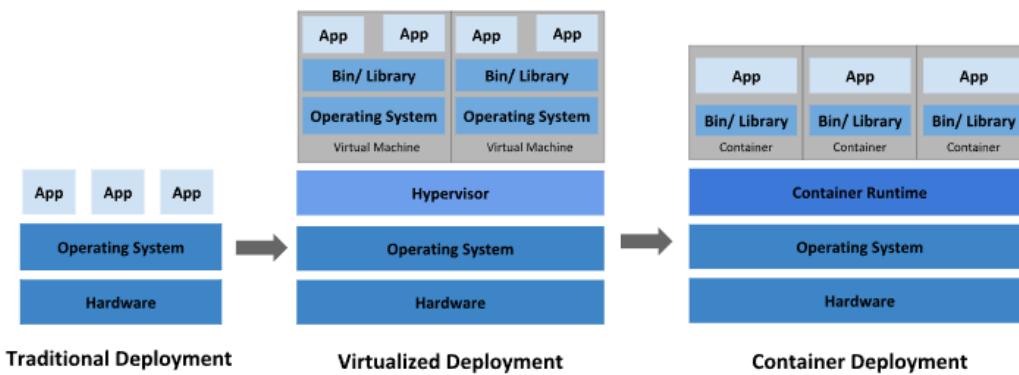


FIGURE I.1 – Architecture multi-couches [1]

### I.3 Synthèse

La virtualisation et la conteneurisation sont deux des technologies les plus fondamentales de l'informatique en nuage moderne et des infrastructures informatiques d'entreprise. Les environnements virtualisés offrent une meilleure efficacité des ressources et une rentabilité accrue en consolidant plusieurs charges de travail sur des serveurs physiques partagés, permettant ainsi des gains significatifs dans l'utilisation du matériel. Pendant ce temps, les environnements conteneurisés permettent un déploiement rapide et une évolutivité fluide en encapsulant les applications et leurs dépendances dans des unités portables, permettant la mise à jour et la mise à l'échelle indépendantes des services sans affecter l'ensemble du système.

L'importance de la virtualisation et de la conteneurisation s'accentue à mesure que les organisations adoptent de plus en plus des stratégies hybrides et multi-cloud : ces technologies permettent de déplacer les charges de travail sans effort entre les centres de données locaux et les plateformes cloud publiques en réponse aux considérations de coût, aux exigences réglementaires ou aux besoins de performance. Dans le même temps, les charges de travail dynamiques et pilotées par la demande — telles que l'analyse en temps réel

— exercent de nouvelles pressions sur les modèles d'approvisionnement statiques conçus pour des schémas prévisibles. Des défis subsistent donc quant à l'équilibre entre performance, disponibilité et coût dans les méthodes d'approvisionnement traditionnelles et statiques. Cela conduit à une question de recherche essentielle : Comment peut-on optimiser l'approvisionnement et le dimensionnement dans un cluster virtualisé afin d'assurer une gestion efficace des ressources ?

Pour traiter cette problématique, nous allons observer et analyser les différentes technologies proposées dans le domaine, telles que les hyperviseurs et les cadres d'orchestration, afin de comprendre comment celles-ci peuvent contribuer à relever ces défis et répondre à la question de recherche.

## 1.2 Étude approfondie des hyperviseurs et de l'orchestration

### II.1 Hyperviseurs et virtualisation

#### II.1.1 Leur rôle dans la virtualisation et la gestion des ressources

En agissant comme intermédiaire entre le matériel physique et les machines virtuelles (VMs), le **hyperviseur** constitue un composant fondamental de la virtualisation. Il abstrait et gère les ressources matérielles, ce qui permet à plusieurs VMs de fonctionner simultanément sur un même hôte physique, chacune agissant comme un environnement informatique indépendant. Cette abstraction améliore l'utilisation des ressources et assure une isolation entre les VMs, garantissant que les performances ou la sécurité d'une VM n'impactent pas les autres. Cette isolation renforce la sécurité et optimise l'utilisation du matériel, ce qui rend les hyperviseurs essentiels pour les centres de données et les environnements cloud modernes. De plus, les hyperviseurs facilitent l'allocation dynamique des ressources, permettant une distribution efficace du CPU, de la mémoire et du stockage en fonction des besoins en temps réel. Cette gestion dynamique permet une meilleure évolutivité et flexibilité des infrastructures informatiques. Les hyperviseurs prennent également en charge des fonctionnalités avancées comme la migration à chaud [14], permettant le transfert de VMs en cours d'exécution entre des hôtes avec un minimum d'interruption, renforçant ainsi la disponibilité et la maintenance du système [15].

#### II.1.2 Types d'hyperviseurs et étude comparative

Il existe deux types principaux d'hyperviseurs, comme le montre la Figure II.2 [16] :

- **Type-1 (Hyperviseurs Bare-Metal) :**

Ces hyperviseurs s'exécutent directement sur le matériel physique, sans nécessiter de système d'exploitation hôte. Exemples : VMware ESXi, Microsoft Hyper-V, KVM,

et Proxmox. Ils sont réputés pour leurs performances élevées, leur sécurité robuste et leur grande évolutivité.

- **Type-2 (Hyperviseurs hébergés)** : Ces hyperviseurs fonctionnent au-dessus d'un système d'exploitation existant, et sont souvent utilisés pour la virtualisation de bureau. Bien qu'ils soient plus simples à configurer, ils introduisent généralement plus de surcharge que leurs équivalents de type 1.

Une étude comparative de ces hyperviseurs met en évidence des différences clés en matière de performance, sécurité et évolutivité. Les hyperviseurs de type 1 offrent une meilleure performance et une gestion directe des ressources, tandis que ceux de type 2 conviennent davantage à des cas d'usage non critiques et individuels [17].

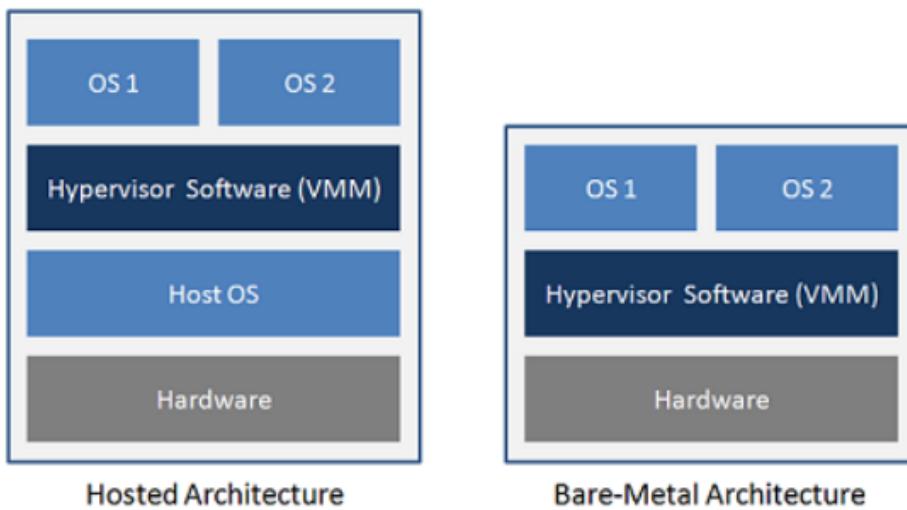


FIGURE II.2 – Hyperviseurs de type 1 et type 2 [2]

## II.2 Des hyperviseurs à l'orchestration : lever les limitations

Bien que les hyperviseurs fournissent une base solide à la virtualisation, ils posent également des problèmes, notamment en matière de surcharge des ressources et de prolifération incontrôlée des VMs (VM sprawl), ce qui peut entraîner une utilisation inefficace des ressources. Une solution plus légère consiste à encapsuler les applications dans des conteneurs portables et isolés. En s'appuyant sur cette conteneurisation, les outils d'orchestration automatisent le déploiement, la mise à l'échelle et la gestion des conteneurs, contribuant ainsi à surmonter les limitations inhérentes aux approches traditionnelles basées sur les hyperviseurs [18].

## **II.3 Orchestration et conteneurisation**

### **II.3.1 Le besoin de conteneurisation dans l'infrastructure moderne**

Les applications cloud-native modernes dépendent des environnements conteneurisés, car ceux-ci sont portables, rapidement déployables et évolutifs. Les plateformes d'orchestration de conteneurs telles que Kubernetes permettent une répartition efficace des charges de travail et assurent une tolérance aux pannes. Ces systèmes, largement utilisés dans divers secteurs, y compris la finance, permettent aux entreprises de gérer facilement des applications complexes. De plus, la conteneurisation soutient l'architecture micro-services, qui divise les applications en services plus petits et plus gérables. En séparant les applications du matériel sous-jacent, la conteneurisation permet des opérations informatiques agiles et résilientes, favorisant le développement, le test et le déploiement indépendants des services [19].

### **II.3.2 Mise à l'échelle et planification dynamiques**

Les plateformes d'orchestration automatisent des aspects essentiels de la gestion des ressources, tels que la mise à l'échelle, la planification et la répartition de charge. La plupart des modèles d'approvisionnement traditionnels reposent sur des techniques de mise à l'échelle statique inefficaces dans des environnements dynamiques. Les systèmes modernes d'orchestration, en revanche, utilisent des méthodes de mise à l'échelle dynamiques qui adaptent les ressources en temps réel aux exigences des charges de travail. Pour gérer les systèmes distribués à grande échelle, ces frameworks sont devenus des outils standards offrant une mise à l'échelle et une planification automatiques, essentielles au maintien de performances optimales dans les infrastructures cloud [20].

## **II.4 Le besoin d'IA dans les environnements orchestrés**

Bien que des systèmes comme Kubernetes, Docker Swarm et Apache Mesos aient fortement amélioré l'automatisation de la distribution des charges de travail et de la mise à l'échelle des ressources, ils reposent encore largement sur des heuristiques basées sur des règles et des déclencheurs à seuil pour prendre des décisions. Ces systèmes sont efficaces pour des charges de travail cohérentes, mais moins adaptés aux environnements très dynamiques, où les charges de travail fluctuent de manière imprévisible ou suivent des modèles non linéaires [21]. Par exemple, un cluster Kubernetes peut être configuré pour autoscaler un déploiement lorsque l'utilisation du CPU dépasse 80% pendant 5 minutes.

Cependant, ces seuils fixes ne tiennent pas compte de facteurs tels que les tendances comportementales des utilisateurs, les variations saisonnières ou les métriques spécifiques aux applications. Cela peut entraîner des retards de mise à l'échelle, une sous-utilisation des ressources, voire des interruptions de service lors de pics de demande [22]. Cette lacune met en évidence la nécessité d'intégrer des modèles d'intelligence artificielle (IA) et d'apprentissage automatique dans les environnements orchestrés. L'IA offre une approche prédictive et basée sur les données pour permettre une prise de décision contextuelle, la prévision des tendances de la demande et l'optimisation en temps réel de l'allocation des ressources. Les systèmes basés sur l'IA analysent de manière proactive les données historiques, prévoient la consommation des ressources et ajustent dynamiquement les stratégies d'approvisionnement. Cela est particulièrement important à mesure que les applications deviennent plus distribuées, hétérogènes et sensibles à la latence [23]. L'intégration de l'IA dans l'orchestration n'est pas seulement une amélioration, mais une évolution nécessaire. L'IA permet la prochaine génération d'orchestration intelligente où les plateformes sont non seulement automatisées, mais aussi adaptatives et prédictives, s'alignant étroitement avec les objectifs de l'entreprise tels que l'efficacité des coûts, la fiabilité et l'expérience utilisateur.

## **II.5 Mise à l'échelle avec l'IA : approvisionnement et optimisation prédictive de l'allocation des ressources**

L'autoscaling conventionnel repose fortement sur des métriques simples comme l'utilisation du CPU et de la mémoire, qui ne reflètent pas toujours fidèlement les fluctuations de la demande en temps réel. La **mise à l'échelle intelligente** s'appuie sur des analyses pilotées par l'IA pour prédire les besoins en ressources, offrant ainsi une approche plus nuancée. Les techniques de mise à l'échelle prédictive analysent les données historiques et les tendances actuelles pour allouer les ressources de manière proactive, réduisant ainsi les risques de sous- ou sur-approvisionnement. Cela permet des économies de coûts et une amélioration des performances du système en alignant l'allocation des ressources sur la demande réelle.

Les modèles d'intelligence artificielle et d'apprentissage automatique sont devenus essentiels dans la transformation de l'approvisionnement en ressources. En analysant les modèles d'utilisation et en prévoyant la demande future, les systèmes pilotés par l'IA peuvent ajuster automatiquement les allocations de ressources pour optimiser les performances et réduire les coûts. Des fournisseurs comme AWS et Google Cloud ont intégré des systèmes d'approvisionnement basés sur l'IA qui adaptent dynamiquement les ressources, démontrant des améliorations significatives en matière d'efficacité et de réactivité [24][25]. L'IA améliore aussi bien la mise à l'échelle horizontale que verticale. La **mise à l'échelle**

**horizontale** consiste à ajouter des instances pour répartir la charge, tandis que la **mise à l'échelle verticale** ajuste les ressources d'une instance unique. Les techniques de distribution de charge pilotées par l'IA garantissent une allocation optimale des ressources en fonction des besoins en temps réel. Des études de cas réelles ont montré que les organisations utilisant l'IA dans leurs stratégies de mise à l'échelle bénéficient d'une réduction des interruptions et d'une amélioration du niveau de service, démontrant encore davantage l'efficacité d'une gestion intelligente des ressources. De nouveaux outils basés sur l'IA, tels que l'autoscaling Kubernetes intégré à des modèles d'apprentissage automatique, révolutionnent la gestion des infrastructures cloud. Les algorithmes d'apprentissage par renforcement sont notamment utilisés pour optimiser continuellement les performances du cloud à partir des données opérationnelles [26][27]. À l'avenir, l'approvisionnement piloté par l'IA jouera un rôle central dans les environnements hybrides et multi-clouds, ouvrant la voie à une gestion de l'infrastructure plus agile et plus intelligente.

### 1.3 Conclusion

En résumé, les hyperviseurs, l'orchestration et l'approvisionnement piloté par l'IA sont fondamentaux pour les infrastructures informatiques modernes. Les hyperviseurs constituent la base de la virtualisation, tandis que la conteneurisation et l'orchestration surmontent les limitations traditionnelles en automatisant le déploiement et la mise à l'échelle. L'approvisionnement basé sur l'IA améliore encore l'allocation des ressources en anticipant dynamiquement la demande et en optimisant l'approvisionnement de manière prédictive. Ensemble, ces avancées améliorent non seulement la performance, l'efficacité des coûts et la résilience des systèmes, mais répondent directement à la question de recherche concernant l'optimisation de l'approvisionnement et de la mise à l'échelle dans les clusters virtualisés. À l'avenir, l'intégration continue de la mise à l'échelle dynamique et des analyses pilotées par l'IA promet de révolutionner le cloud computing, ouvrant la voie à des environnements informatiques plus agiles et plus robustes.

## Chapitre 2

# Approvisionnement et Prédition Intelligents des Ressources (SRPP)

## 2.1 introduction

Un approvisionnement efficace des ressources et une prédition précise de la demande sont essentiels dans l'informatique en nuage et les environnements virtualisés afin d'optimiser les performances, contrôler les coûts et maintenir la disponibilité face à des charges de travail dynamiques. Les méthodes traditionnelles d'approvisionnement statique, qui allouent les ressources selon une demande de pointe ou moyenne, entraînent souvent une sous-utilisation lors des périodes de faible demande et une dégradation des performances en cas de pics.

Ce chapitre explore l'évolution et l'application de techniques intelligentes pour l'approvisionnement et la prédition des ressources dans le contexte d'un cluster virtualisé, à travers quatre paradigmes complémentaires : textbf{Algorithmes Métaheuristiques}, textbf{Apprentissage Machine et Profond (ML/DL)}, textbf{IA Générative (GenAI)} et textbf{Systèmes Multi-Agents (MAS)}. Ces paradigmes ont montré un potentiel significatif pour modéliser le problème d'approvisionnement comme :

- **Une tâche dynamique.**
- **Une optimisation multi-objectifs.**
- **Capable de s'adapter à des charges incertaines.**
- **Des environnements hétérogènes.**
- **Et des contraintes opérationnelles en temps réel.**

En étudiant la littérature existante et en analysant les performances comparatives de ces méthodes, nous discuterons à la fois des **forces et limitations** des approches actuelles, tout en mettant en lumière les tendances émergentes qui ouvrent la voie à des systèmes de gestion d'infrastructure plus adaptatifs, évolutifs et autonomes.

## 2.2 Approvisionnement et Prédition Intelligents des Ressources (SRPP) Basés sur les Métaheuristiques

### 2.2.1 Vue d'ensemble des algorithmes métahéuristiques

Les algorithmes métahéuristiques sont des stratégies de haut niveau, indépendantes du problème, conçues pour explorer efficacement de vastes espaces de solutions complexes en combinant une exploration stochastique avec une amélioration itérative. Cela les rend particulièrement adaptés aux défis d'optimisation NP-difficiles liés à l'approvisionnement en ressources dans le cloud. Inspirées par des processus naturels et sociaux, ces méthodes comprennent notamment les algorithmes génétiques (AG), qui simulent l'évolution par

sélection, croisement et mutation ; l'optimisation par essaim particulaire (PSO), qui modélise le comportement collectif en faisant évoluer les solutions candidates (“particules”) en fonction de leurs meilleurs résultats personnels et globaux ; et l'optimisation par colonie de fourmis (ACO), qui repose sur une construction de chemins guidée par les phéromones. Leurs cadres de type population ou essaim soutiennent intrinsèquement le parallélisme, ce qui permet des déploiements réactifs et évolutifs dans les plateformes cloud distribuées. Leur nature non déterministe leur permet également de s'adapter à l'incertitude et à l'hétérogénéité des infrastructures virtualisées [28][29][30][31]. La figure 2.1 ci-dessous illustre quelques exemples de techniques métahéuristiques.

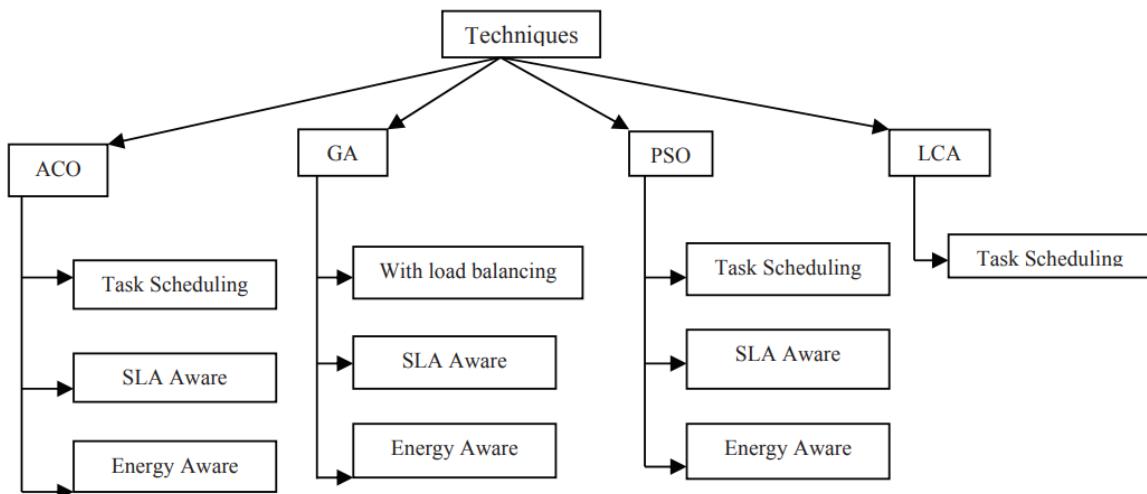


Fig. 1. Classification of Algorithms

FIGURE 2.1 – Techniques Métaheuristiques [3]

### 2.2.2 Revue des travaux basés sur les métahéuristiques

L'efficacité des techniques métahéuristiques dans la modélisation de l'approvisionnement en ressources cloud a été démontrée par plusieurs études, sous forme de problèmes d'optimisation combinatoire, en adaptant les fonctions objectifs aux contraintes réelles. Les travaux précurseurs de Wang et al. [32] ont appliqué l'optimisation par essaim particulaire (PSO) au placement de machines virtuelles (VM) dans des centres de données virtualisés sensibles à l'énergie, permettant des réductions significatives de la consommation d'énergie via la consolidation dynamique des VMs. Xiong et Xu [33] ont étendu PSO aux environnements OpenStack, en formulant des modèles d'allocation multi-ressources équilibrant les besoins en CPU, mémoire et stockage. Les algorithmes génétiques ont également été utilisés pour l'équilibrage de charge et la consolidation : les heuristiques adaptatives de consolidation de VMs proposées par Beloglazov et Buyya [34] ont amélioré la conformité aux SLA et l'efficacité énergétique dans les clouds IaaS, tandis que des ap-

proches intelligentes récentes d'AG pour l'équilibrage de charge dans les clouds hybrides, proposées par Rajkumar et Katiravan [30], ont montré des performances robustes dans des charges de travail hétérogènes. Les méthodes basées sur l'ACO (optimisation par colonie de fourmis), telles que celles introduites par Ferdaus et al. [35], utilisent des stratégies de placement guidées par phéromones pour réduire les coûts énergétiques des centres de données. Des métaheuristiques hybrides combinant AG et PSO, comme l'évaluation comparative de GA-PSO pour l'ordonnancement des workflows par Subramoney et Nyirenda [36], et le GA+PSO hybride de Calheiros et al. pour les workflows intensifs en données [37], offrent une convergence plus rapide et de meilleurs compromis coût/performance. Les variantes adaptatives de PSO, incluant PSO à poids d'inertie linéairement décroissant ou adaptatif [38], ainsi que le PSO binaire pour le placement initial de VMs [39], renforcent la résilience face à des demandes dynamiques et multi-objectifs. Enfin, des algorithmes hybrides chaotiques, comme ceux de Mohammadzadeh et al. [40], intègrent des cartes chaotiques dans HSOS-SOA pour éviter les optima locaux et optimiser l'ordonnancement de workflows scientifiques dans des clouds multi-sites. Ces divers cadres métaheuristiques soulignent leur polyvalence et leur efficacité dans l'optimisation des défis complexes et dynamiques liés à l'approvisionnement dans les infrastructures cloud modernes [29][41].

Le tableau 2.1 présente un résumé consolidé des principales recherches basées sur les métaheuristiques dans le domaine de l'allocation des ressources cloud. Il met en évidence les techniques appliquées, les performances obtenues, ainsi que les forces et les limites de chaque méthode, permettant ainsi une analyse comparative des différentes stratégies d'optimisation.

TABLE 2.1 – Summary of Surveyed Metaheuristic-Based Works

Technology	Research Work (Authors, Year)	Technique Used	Proficiency Achieved	Strength	Weakness
Metaheuristics	Wang et al. (2013)	PSO	Significant reduction in power consumption through dynamic VM consolidation.	Energy-efficient allocation (substantially lowers power use).	Static optimization; requires offline tuning and lacks demand forecasting.
	Xiong & Xu (2014)	Extended PSO (multi-resource VM allocation)	Balanced CPU, memory and storage allocation in OpenStack (improving resource utilization).	Handles multiple resource types simultaneously.	Static optimization; needs careful parameter tuning.
	Beloglazov & Buyya (2012)	Genetic Algorithm (adaptive VM consolidation)	Improved SLA compliance and higher energy efficiency in IaaS clouds.	Adaptive consolidation adapts to workload changes.	Heuristic thresholds may not generalize to all scenarios.
	Rajkumar & Kattiravan (2023)	Genetic Algorithm (hybrid cloud load balancing)	Robust performance across heterogeneous workloads.	Effective for diverse, multi-cloud environments.	Computationally intensive; slower convergence on large-scale.

**Table 2.1 – continued from previous page**

Technology	Research Work	Technique Used	Proficiency Achieved	Strength	Weakness
	Ferdaus et al. (2014)	Ant Colony Optimization (ACO)	Reduced data center energy costs via pheromone-based VM placement.	Finds global allocation solutions effectively.	Convergence can be slow; sensitive to pheromone settings.
	Subramoney & Nyirenda (2020)	Hybrid GA-PSO	Faster convergence and superior cost-performance trade-off in workflow scheduling.	Combines GA's exploration with PSO's exploitation.	Increased algorithmic complexity and tuning overhead.
	Calheiros et al. (Year N/A)	Hybrid GA+PSO	Superior cost-performance trade-offs in data-intensive workflow scheduling.	Leverages strengths of both GA and PSO searches.	Requires domain-specific tuning; complex hybrid setup.
	Mohammadzadeh et al. (2023)	Chaotic HSOS-SA (GA/PSO with chaotic maps)	Avoids local optima in multi-site workflow scheduling.	Enhanced exploration using chaos theory.	Very complex; chaotic component adds uncertainty and tuning needs.

### 2.2.3 Analyse comparative des approches météahuristiques

Lors de l'évaluation des algorithmes météahuristiques pour l'allocation des ressources, plusieurs compromis deviennent apparents. Les **algorithmes génétiques (GA)**, avec leurs opérateurs de croisement et de mutation, offrent généralement une meilleure qualité de solution et une exploration globale plus efficace que les heuristiques plus simples ; cependant, cela se fait au prix d'une convergence plus lente et d'un surcoût computationnel accru, en raison de l'évaluation de la condition physique à l'échelle de la population. Le **Particle Swarm Optimization (PSO)** converge généralement plus rapidement — un avantage pour l'adaptation quasi temps réel — mais peut souffrir de convergence prématuée, notamment dans les espaces de recherche multimodaux sans mécanismes adéquats de maintien de la diversité. L'**Ant Colony Optimization (ACO)** excelle dans les problèmes d'affectation de tâches discrètes en exploitant des traces de phéromones pour guider les solutions successives, mais ses mises à jour itératives de phéromones posent des problèmes de passage à l'échelle lorsque la taille du problème augmente. Pour l'ensemble de ces méthodes, la **sensibilité aux paramètres** — tels que le taux de croisement pour GA, le poids d'inertie pour PSO ou le taux d'évaporation des phéromones pour ACO — nécessite un réglage minutieux hors ligne, ce qui complique leur déploiement dans des environnements cloud dynamiques et hétérogènes. De plus, bien que les structures de population ou d'essaim permettent une exécution parallèle sur une infrastructure distribuée, leurs coûts d'exécution et de communication peuvent limiter leur passage à l'échelle, en particulier sous des contraintes strictes de SLA. Ces constats soulignent la nécessité de mécanismes adaptatifs de réglage des paramètres et de stratégies hybrides capables de trouver un équilibre dynamique entre exploration et exploitation afin de répondre aux exigences de provisionnement en temps réel.

### 2.2.4 Défis et lacunes de recherche

Malgré des avancées notables, plusieurs défis persistants freinent l'adoption généralisée des algorithmes météahuristiques pour l'approvisionnement en ressources dans les environnements cloud dynamiques. Premièrement, la **prise de décision en temps réel** reste problématique, car la majorité des météahuristiques fonctionnent en mode batch ou hors ligne, nécessitant de nombreuses évaluations de fonctions d'adaptation (fitness) qui entraînent une latence et une surcharge computationnelle importantes, limitant ainsi leur applicabilité aux décisions instantanées de mise à l'échelle ou de planification [42][28]. Deuxièmement, l'**approvisionnement sensible à la consommation énergétique** exige des modèles de puissance précis reflétant les comportements non linéaires de la consommation dans des infrastructures matérielles hétérogènes. L'intégration de ces modèles dans les fonctions d'optimisation introduit des contraintes complexes et multidi-

mensionnelles que beaucoup d’algorithmes existants n’approchent qu’approximativement, ce qui mène à des compromis sous-optimaux entre performance et efficacité énergétique [43]. Troisièmement, l'**optimisation multi-objectifs** visant simultanément à minimiser le coût, la latence et l’énergie tout en maximisant la disponibilité pose un défi pour les métahéuristiques à objectif unique. Bien que des extensions basées sur Pareto existent, elles souffrent souvent d’une vitesse de convergence réduite et d’une complexité computationnelle accrue, en particulier lorsqu’il s’agit de gérer des centaines, voire des milliers de VMs et de conteneurs [44].

De plus, la **modélisation précise des charges de workload dynamiques**, caractérisées par une demande explosive, imprévisible et des chaînes de services interdépendantes, demeure un défi majeur. Les métahéuristiques traditionnelles ne disposent pas de mécanismes intrinsèques pour la prédition temporelle et se basent souvent sur des instantanés statiques de l’état du système, ce qui entraîne un approvisionnement sous-optimal en cas de variations soudaines de la charge ou de cycles journaliers [45]. La **sensibilité aux paramètres** complique davantage le déploiement : les réglages de contrôle tels que la taille de la population, les taux de mutation ou les poids d’inertie nécessitent généralement un ajustement hors ligne approfondi, qui ne se généralise pas d’un environnement cloud ou d’un profil de charge à l’autre [46].

Enfin, malgré des efforts initiaux pour combiner les métahéuristiques avec l’apprentissage automatique, l’intégration de **grands modèles de langage (LLMs)** ou d’autres systèmes prédictifs avancés reste à ses débuts.

Exploiter les LLMs pourrait enrichir la prévision de la demande sensible au contexte en interprétant les journaux (logs), le comportement utilisateur, ou encore les signaux externes (par exemple, les tendances sur les réseaux sociaux), ce qui permettrait d’orienter plus intelligemment les recherches métahéuristiques et de réduire la dépendance aux paramétrages statiques.

Répondre à ces lacunes nécessite le développement de **cadres métahéuristiques adaptatifs** intégrant l’apprentissage en ligne, des fonctions de coût sensibles à l’énergie, et des boucles d’optimisation hybrides basées sur la prédition pour satisfaire les exigences multi-objectifs et en temps réel des infrastructures cloud de nouvelle génération.

## 2.3 Approvisionnement et Prédition Intelligents des Ressources (SRPP) Basés sur le ML et le DL

### 2.3.1 Vue d’ensemble des méthodes ML et DL

Les systèmes d’approvisionnement et de prédition des ressources utilisent couramment une variété d’algorithmes d'**apprentissage automatique (ML)** pour modéliser

et anticiper les besoins en charge de travail à partir de la télémétrie historique et des métriques système. Parce qu'ils offrent un bon compromis entre interprétabilité et capacité prédictive, les algorithmes d'apprentissage supervisé tels que les **arbres de décision**, les **machines à vecteurs de support (SVM)**, les **forêts aléatoires** et le **gradient boosting** sont particulièrement populaires. Les arbres de décision partitionnent l'espace des caractéristiques de manière récursive pour produire une structure de modèle transparente, tandis que les forêts aléatoires améliorent la généralisation en moyennant les prédictions d'un ensemble d'arbres décorrelés — un procédé qui réduit considérablement le surapprentissage et la variance dans les prévisions de charge cloud. Les SVM construisent des hyperplans maximisant les marges entre les classes (ou s'ajustant aux cibles de régression), ce qui les rend efficaces pour des ensembles de données de taille moyenne avec des frontières bien définies. Bien que les méthodes de boosting par gradient, qui construisent de manière itérative des modèles additifs en se concentrant sur les erreurs résiduelles, offrent une grande précision, elles nécessitent un réglage minutieux des hyperparamètres pour éviter le surapprentissage et maîtriser le temps d'apprentissage [47]. En identifiant des motifs d'usage et en réduisant la dimensionnalité des caractéristiques, les techniques non supervisées telles que le **clustering k-moyennes** et l'**analyse en composantes principales (PCA)** contribuent également aux solutions d'approvisionnement et de prédition, améliorant ainsi les performances et l'évolutivité des modèles de prévision en aval.

Les techniques d'**apprentissage profond (DL)** excellent quant à elles dans la capture des relations temporelles et non linéaires complexes dans les séries temporelles — une capacité essentielle pour prédire les variations rapides de la demande en ressources cloud. Bien que les réseaux de neurones artificiels (**ANNs**) en architecture feedforward ne disposent pas de mécanismes de dépendance séquentielle, ils ont jeté les bases de la modélisation des interactions non linéaires. Les **réseaux de neurones récurrents (RNNs)** et en particulier les réseaux **Long Short-Term Memory (LSTM)** traitent cette limitation en conservant en mémoire les entrées précédentes, permettant ainsi des prédictions plus précises de l'utilisation du CPU, de la mémoire et du réseau sur des horizons temporels variables [48]. Les extensions comme les architectures **encodeur–décodeur LSTM avec mécanismes d'attention** renforcent encore la capacité du modèle à se concentrer sur les étapes temporelles pertinentes, améliorant ainsi la précision des prévisions dans les scénarios de charge par lots [49].

Cette combinaison de méthodes ML et DL offre aux concepteurs de systèmes une boîte à outils complète : des algorithmes ML pour des prévisions rapides et interprétables, et des architectures DL pour une modélisation temporelle profonde, permettant une gestion adaptative et fondée sur les données des ressources dans les infrastructures cloud modernes.

### 2.3.2 Revue des travaux basés sur le ML et le DL

#### a. Encodeur–Décodeur LSTM avec Attention pour la Prédition de Charges de Travail Mixtes

Ding et al. dans [49] ont proposé un réseau encodeur–décodeur LSTM enrichi par un mécanisme d’attention pour prédire les charges de travail mixtes (batch et interactives) dans des clusters cloud en périphérie. Le modèle extrait des caractéristiques séquentielles et contextuelles à partir des données historiques d’utilisation, en appliquant l’attention dans le décodeur pour se concentrer sur les étapes temporelles pertinentes. Les expériences menées sur les ensembles de données de traces Alibaba et Dinda montrent une précision prédictive de pointe, avec une réduction de l’erreur quadratique moyenne (RMSE) d’environ 15% par rapport aux modèles LSTM standards et aux méthodes traditionnelles de séries temporelles dans un environnement cloud public.

#### b. BG LSTM : BiLSTM et GridLSTM pour la Prédition Conjointe des Séries Temporelles de Charges et de Ressources

Zhou et al. dans [50] présentent l’architecture BG LSTM, qui intègre des couches BiLSTM capables de capturer les dépendances temporelles dans les deux directions (avant et arrière) avec des cellules GridLSTM traitant les données à travers les dimensions temporelles et de caractéristiques. En utilisant les données de traces du cluster Google, la chaîne de prétraitement du modèle applique un redimensionnement logarithmique et un filtrage du bruit avant l’apprentissage. BG LSTM atteint une généralisation supérieure, surpassant les modèles LSTM et GRU de référence avec une amélioration de 20% de la RMSE pour la prédition conjointe de la charge de travail et de l’utilisation des ressources dans les environnements de centres de données à grande échelle.

#### c. Réseau de Neurones Artificiel (ANN) avec Évolution Différentielle Adaptive pour la Prédition de Charges Cloud

Santos et al. dans [51] explorent une approche hybride combinant un réseau de neurones artificiel feedforward (ANN) avec un optimiseur par évolution différentielle adaptive (ADE) pour le réglage des poids. Utilisant les traces des jeux de données Bitbrains et Google Cluster Dataset (GCD), le modèle intègre des variables telles que la charge CPU, l’utilisation mémoire et les entrées/sorties réseau, en optimisant itérativement les poids pour minimiser la RMSE et l’erreur absolue moyenne (MAE). La méthode proposée surpassé les performances de l’ANN seul et de l’évolution différentielle classique avec des gains de 18 à 22% sur plusieurs intervalles de prédition dans des bancs d’essai de cloud hybride.

Ces exemples illustrent la diversité des travaux de recherche sur les solutions d'approvisionnement et de prédition basées sur le ML/DL : des modèles purement séquentiels enrichis par l'attention aux architectures multidimensionnelles en passant par des réseaux optimisés par évolution, tous appliqués à des environnements cloud où des prévisions précises de la charge de travail et de l'utilisation des ressources guident directement les décisions d'approvisionnement dynamique.

Le tableau 2.2 présente un résumé des principales approches basées sur l'apprentissage automatique (ML) et l'apprentissage profond (DL) appliquées à la gestion des ressources cloud. Il décrit les techniques spécifiques utilisées, les améliorations de performance obtenues, ainsi que les forces et limites inhérentes à chaque méthode pour relever les défis liés à la prévision des charges de travail et à l'allocation des ressources dans des ensembles de données cloud variés.

TABLE 2.2 – Summary of Surveyed ML and DL-Based Works

Technology	Research Work (Authors, Year)	Technique Used	Proficiency Achieved	Strength	Weakness
ML/DL	Ding et al. (2019)	Attention-based LSTM Encoder–Decoder	15% RMSE reduction vs. standard LSTM on Alibaba/Dinda traces.	Attention mechanism focuses on relevant time steps, improving accuracy.	Requires substantial training data; may overfit limited traces.
	Zhou et al. (Year N/A)	BG-LSTM (BiLSTM + GridLSTM)	20% lower RMSE than LSTM/GRU baselines on Google Cluster data.	Captures forward/backward and spatial patterns (strong generalization).	Complex model; high training and inference costs.
	Santos et al. (Year N/A)	ANN + Adaptive Differential Evolution	18–22% lower RMSE/MAE than standalone ANN or DE on cloud workloads.	Differential evolution optimizes weights for better forecasting.	Evolutionary optimization is computationally expensive.

### 2.3.3 Analyse comparative

Les algorithmes de **ML** tels que les **arbres de décision**, les **forêts aléatoires** (Random Forests) et le **gradient boosting** offrent généralement des **temps d'apprentissage et d'inférence plus rapides** ainsi que des **besoins computationnels réduits**, ce qui les rend bien adaptés aux décisions d'approvisionnement en temps réel ou quasi temps réel dans des environnements à ressources limitées.

Ces méthodes offrent également une **interprétabilité accrue** — par exemple, les scores d'importance des caractéristiques dans les forêts aléatoires permettent aux opérateurs d'identifier les métriques (charge CPU, utilisation mémoire, E/S réseau) qui influencent le plus les prédictions, facilitant ainsi le diagnostic et la conformité aux SLA [52].

Cependant, les modèles ML peinent souvent à **capturer les dépendances temporelles à long terme** et les interactions non linéaires complexes propres aux séries temporelles de charges cloud, ce qui entraîne des **erreurs de prévision plus élevées** en présence de variations saisonnières ou de pics de demande [52][53].

À l'inverse, les architectures de **DL**, en particulier les **réseaux LSTM** et les **Transformers**, excellent dans la modélisation des données séquentielles et l'**apprentissage des corrélations temporelles complexes**, ce qui se traduit par une **réduction de 15 à 25 % de la RMSE** par rapport aux méthodes statistiques et aux modèles ML traditionnels sur des ensembles de données de traces cloud à grande échelle [54].

Leur capacité à extraire automatiquement des caractéristiques hiérarchiques à partir des données brutes de télémétrie réduit le besoin d'ingénierie manuelle des caractéristiques, permettant une **meilleure généralisation** à des types de charges non vues.

Néanmoins, les solutions DL impliquent une **latence d'apprentissage et d'inférence plus élevée**, nécessitent un **volume important de données annotées** pour un entraînement efficace, et souffrent souvent d'un manque de transparence, ce qui complique l'analyse des causes racines et ralentit la mise à jour des modèles dans des environnements dynamiques [54][55].

Pour concilier ces compromis, de nombreuses solutions modernes d'approvisionnement et de prédiction adoptent des **stratégies hybrides** combinant l'interprétabilité et la rapidité du ML pour les prévisions à court terme, avec la profondeur et la précision du DL pour les horizons plus longs. En sélectionnant dynamiquement ou en combinant des modèles selon la volatilité prévue des charges, ces systèmes atteignent une **gestion des ressources robuste et évolutive**, répondant à la fois aux exigences de réactivité en temps réel et à la précision des prévisions dans les infrastructures cloud modernes.

### 2.3.4 Limites et Lacunes de Recherche

Malgré les avancées significatives dans les solutions d'approvisionnement et de prédition basées sur le ML/DL, plusieurs défis critiques subsistent.

Tout d'abord, le **problème du démarrage à froid** persiste lorsqu'une nouvelle application ou un nouveau service ne dispose pas encore de données de télémétrie historiques suffisantes. Dans ces situations, les modèles prédictifs offrent des performances médiocres jusqu'à l'accumulation d'assez de données. Bien que l'apprentissage méta (meta-learning) et l'exploitation d'informations auxiliaires puissent atténuer cet effet, peu d'études ont abordé de manière approfondie ces scénarios dans les environnements cloud [56].

Deuxièmement, l'**adaptabilité aux charges de travail non stationnaires**, caractérisées par des pics soudains de demande, des tendances saisonnières ou des changements de comportement utilisateur, reste un défi majeur. Ce contexte nécessite des modèles capables de généraliser efficacement dans le temps, notamment via l'apprentissage par transfert ou des approches tenant compte de l'incertitude. Des travaux récents sur l'apprentissage par transfert appliqués aux traces cloud montrent un fort potentiel, bien que ces méthodes soient encore peu adoptées en production [57].

Enfin, les architectures DL souffrent souvent d'un manque de transparence, ce qui complique l'analyse des causes racines des erreurs de prédition et ralentit la mise à jour des modèles dans des environnements évolutifs [55].

La prise en compte de ces lacunes sera essentielle pour concevoir la prochaine génération de systèmes cloud intelligents, résilients et capables de s'auto-optimiser de manière fiable face à des charges dynamiques.

## 2.4 IA Générative pour l'Approvisionnement et la Prédition Intelligents des Ressources (SRPP)

### 2.4.1 Vue d'ensemble des méthodes d'IA Générative

Le développement rapide du cloud computing exige des techniques de gestion des ressources plus intelligentes et flexibles. L'intelligence artificielle générative (GenAI), qui englobe des modèles tels que les réseaux antagonistes génératifs (GANs), les autoencodeurs variationnels (VAEs) et les grands modèles de langage (LLMs), s'est imposée comme une force de transformation dans ce domaine. En exploitant GenAI, les infrastructures cloud peuvent atteindre une meilleure automatisation, une précision prédictive accrue et une efficacité opérationnelle renforcée.

Grâce à la génération de données synthétiques, l'analyse prédictive et l'allocation dynamique des ressources, les capacités de la GenAI dépassent celles des modèles classiques

d'intelligence artificielle. Ces caractéristiques sont particulièrement utiles pour gérer la complexité des environnements cloud modernes, où les charges de travail sont de plus en plus hétérogènes et imprévisibles. En intégrant la GenAI dans certains systèmes d'allocation et de prédition des ressources, on peut améliorer la compréhension et l'anticipation des besoins en ressources, menant ainsi à un approvisionnement optimisé et à une amélioration du respect des accords de niveau de service (SLA).

L'intelligence artificielle générative repose sur des modèles qui **apprennent des distributions latentes** ou **génèrent des données réalistes** afin de renforcer les systèmes prédictifs et d'approvisionnement des environnements cloud. En combinant l'apprentissage antagoniste avec des composants Transformer ou LSTM, les **réseaux antagonistes génératifs (GANs)** — comme l'architecture hybride VTGAN — produisent des traces synthétiques de charge de travail et classifient les tendances, atteignant plus de 95 % de précision sur des jeux de données cloud réels, tout en renforçant les tests de résistance des algorithmes d'approvisionnement [58][59]. L'intégration de LLMs dans des plateformes d'observabilité — telles que Splunk — permet également l'automatisation de l'interprétation des journaux en temps réel et des actions correctives, fournissant ainsi des interfaces conversationnelles pour la gestion des infrastructures [60]. En **augmentant les données d'entraînement**, en **capturant des dépendances complexes**, et en permettant un **approvisionnement adaptatif piloté par l'IA** à grande échelle, ces méthodes génératives enrichissent collectivement les solutions cloud.

### 2.4.2 Revue des travaux basés sur l'IA Générative

#### a. WGAN gp Transformer pour la Prédition de Charge Cloud

Arbat et al. dans [61] proposent le **WGAN gp Transformer**, une combinaison novatrice d'un générateur de type Transformer et d'un critique GAN de Wasserstein, conçue pour prédire les taux d'arrivée des charges cloud avec une grande précision et une faible latence d'inférence. Le modèle a été évalué sur des traces réelles des clusters Alibaba et Dinda, en utilisant les horodatages d'arrivée des jobs, l'utilisation CPU et les métriques d'I/O comme caractéristiques d'entrée. Comparé à des prédicteurs basés sur LSTM de pointe, le WGAN gp Transformer a atteint jusqu'à **5,1 % de précision supplémentaire** (réduction de la RMSE) et une vitesse d'inférence **5 fois supérieure**, tout en réduisant de manière significative la sur- et sous-allocation de machines virtuelles lorsqu'il est intégré dans un mécanisme d'auto-scaling Google Cloud.

### b. CILP : Apprentissage par Imitation Basé sur la Co-Simulation

Tuli et al. présentent **CILP**, qui formule l’approvisionnement proactif en VMs comme deux sous-problèmes : la prédition de charge via un modèle de substitution neuronal de type Transformer, et l’optimisation de l’approvisionnement à l’aide d’un apprenant par imitation couplé à un jumeau numérique co-simulé. Les expériences, menées sur trois jeux de données publics — **Azure2017**, **Azure2019** et **Bitbrain** — utilisent comme entrées historiques la charge CPU, l’usage mémoire et le trafic réseau. Comparé aux méthodes d’optimisation en ligne et hors ligne existantes, CILP a permis jusqu’à **22% d’amélioration de l’utilisation des ressources, 14% d’amélioration du score de QoS et 44 % de réduction des coûts d’exécution**, démontrant l’intérêt d’intégrer boucles prédictives et optimisation dans les cadres GenAI [62][63].

### c. VTGAN : GANs Hybrides pour la Prédition de Tendance des Charges

Maiyza et al. [58] proposent **VTGAN**, une architecture hybride combinant des GANs avec des générateurs empilés LSTM ou GRU, et un discriminateur CNN 1D pour prédire à la fois les valeurs de charge de travail et leur *tendance* (hausse/baisse). En utilisant le jeu de données Bitbrains — incluant l’utilisation CPU, la mémoire et les séries temporelles d’I/O — et en enrichissant les variables par des indicateurs techniques, des transformées de Fourier et d’ondelettes, VTGAN surpassé les méthodes ARIMA, les LSTM/GRU purs, et les modèles hybrides CNN-LSTM/GRU. Il atteint une **précision de classification des tendances entre 95,4% et 96,6%** comme le montre [64], tout en obtenant une erreur MAPE plus faible pour des prédictions multi-pas selon diverses configurations de fenêtres glissantes.

Ces études illustrent la diversité des applications de l’IA générative dans les environnements virtualisés : de la génération synthétique de charges et l’amélioration de la précision prédictive, à l’intégration dans des systèmes de co-simulation et de provisionnement sensible aux tendances. Elles soulignent ainsi le potentiel de la GenAI pour créer des systèmes de gestion des ressources cloud plus résilients et efficaces.

Le tableau 2.3 fournit un résumé structuré des travaux de recherche récents exploitant les techniques d’IA générative pour la prédition et l’optimisation des ressources cloud. Il présente les principaux modèles génératifs utilisés, quantifie les gains de performance en termes de précision et d’efficacité, et met en évidence les avantages clés ainsi que les défis liés à l’application de modèles tels que les GAN et les Transformers dans des environnements cloud dynamiques.

TABLE 2.3 – Summary of Surveyed Generative AI-Based Works

Technology	Research Work (Authors, Year)	Technique Used	Proficiency Achieved	Strength	Weakness
<b>Generative AI</b>	Arbat et al. (2022)	WGAN-gp Transformer	Up to 5.1% higher prediction accuracy and 5× faster inference than LSTM.	High accuracy with low-latency inference.	Complex GAN training; risk of mode collapse.
	Tuli et al. (2023)	Transformer + Imitation Learning (CILP)	Achieved 22% higher resource utilization, 14% improvement in QoS, and 44% lower cost.	Integrates forecasting and optimization loops.	High simulation and training overhead.
	Maiyza et al. (2023)	VTGAN (GAN + RNN/CNN)	Trend classification accuracy between 95.4% and 96.6%; lower MAPE across sliding windows.	Accurate trend-aware forecasting.	Very complex; requires extensive feature engineering.

### 2.4.3 Cas d'usage et intégration pratique

L'IA générative trouve des applications concrètes dans l'automatisation et l'amélioration des workflows de prédition et d'approvisionnement dans les principales plateformes cloud :

- **Génération d'IaC (Infrastructure as Code) avec Amazon Bedrock Agents :** Les équipes utilisent les agents Amazon Bedrock pour convertir des schémas d'architecture de haut niveau en scripts Terraform ou CloudFormation conformes. Lors d'une démonstration, les agents ont interprété des schémas Visio, généré automatiquement des configurations respectant les politiques de sécurité, et déployé les piles, réduisant de plus de 70% l'effort de scripting manuel tout en assurant la conformité [65].
- **Débogage IA des jobs Spark dans AWS Glue :** La fonction de débogage IA de Glue s'appuie sur les modèles d'Amazon Bedrock pour analyser les logs de jobs Spark échoués, identifier les causes racines et recommander des corrections de code ou de configuration. En un clic, l'utilisateur peut lancer une analyse automatisée qui remplace des jours de débogage manuel par quelques minutes d'analyse générative [66].
- **RAG (Retrieval-Augmented Generation) pour le scaling piloté par la connaissance :** Certaines organisations construisent des pipelines RAG avec Bedrock Knowledge Bases et LlamaIndex pour entraîner les modèles sur leurs propres données de télémétrie et de configuration. Ces systèmes répondent à des requêtes comme «Qu'est-ce qui a causé le pic CPU à 3h ? » et ajustent automatiquement les seuils de Kubernetes Horizontal Pod Autoscaler (HPA) selon les recommandations RAG, améliorant la précision de l'auto-scaling et réduisant les violations de SLA [67].
- **Analyse prédictive pour le scaling de conteneurs :** Bien que le scaling prédictif d'AWS pour ECS s'appuie sur du ML avancé plutôt que sur des modèles génératifs purs, sa technologie formée sur des millions de points de télémétrie démontre l'intégration fluide de la prévision dans l'approvisionnement. En ajustant préventivement le nombre de conteneurs avant les pics de demande, ce système permet de réduire les coûts de sur-approvisionnement de jusqu'à 25% tout en maintenant la réactivité applicative [68].

Ces cas mettent en évidence des schémas communs d'intégration :

1. **Points d'accès GenAI serverless :** Les modèles et agents Bedrock s'exécutent sur des points d'accès entièrement gérés et auto-scalés, éliminant la charge de gestion d'infrastructure et garantissant une haute disponibilité.

2. **Augmentation basée sur RAG** : Les bases de connaissances fournissent des données de contexte à jour — télémétrie, runbooks, diagrammes — que les agents utilisent pour générer du code IaC ou des étapes de remédiation.
3. **Boucles de rétroaction (Feedback Loops)** : Les données de performance des modèles et la télémétrie issue des changements déployés alimentent en retour l'amélioration continue des prédictions et de la prise de décision.

## 2.5 Systèmes Multi-Agents pour l'Approvisionnement et la Prédition Intelligents des Ressources

### 2.5.1 Vue d'ensemble des méthodes multi-agents

Les environnements cloud dynamiques nécessitent des systèmes de prise de décision décentralisés capables de coordonner plusieurs entités autonomes pour gérer les charges fluctuantes, l'hétérogénéité et des objectifs de service complexes (SLA). En répartissant l'autorité entre un réseau d'agents spécialisés, chacun responsable de la surveillance, de la prédition ou de l'approvisionnement d'un sous-ensemble de ressources, les **systèmes multi-agents (MAS)** répondent à ces exigences. Les architectures MAS offrent un paradigme robuste pour l'approvisionnement et la prédition intelligents à grande échelle dans des infrastructures virtualisées, en permettant la négociation collaborative, l'autonomie locale et l'adaptation en temps réel [69][70].

Les **agents** dans un MAS sont des objets logiciels autonomes dotés de capacités de perception, de raisonnement et d'action. Ils interagissent dans un environnement — tel qu'un cluster cloud — en appliquant des techniques de coordination et des protocoles de communication. Les approches MAS courantes incluent :

- **Systèmes multi-agents à base de règles** : Les agents allouent les ressources via des procédures de négociation ou d'enchères pré-définies, ce qui garantit un comportement prévisible mais limite l'adaptabilité [71].
- **MAS basés sur l'apprentissage par renforcement (MARL)** : Les agents apprennent des politiques d'approvisionnement optimales par essais-erreurs, équilibrant récompenses locales (ex : respect des SLA) et objectifs globaux (ex : efficacité énergétique) [72].
- **Architectures MAS hiérarchiques** : Les tâches sont réparties sur plusieurs couches, entre agents orchestrateurs globaux et agents exécutants locaux, ce qui permet des hiérarchies de décision évolutives et une convergence plus rapide [73].

Ces configurations favorisent la **scalabilité**, la **tolérance aux pannes** et le **contrôle**

décentralisé, ce qui rend les MAS bien adaptés à la complexité des charges de travail cloud contemporaines.

### 2.5.2 Revue des travaux basés sur les systèmes multi-agents

#### a. IMARM (Modèle Intelligent Multi-Agent par Renforcement)

Un système combinant des agents à noyaux d'apprentissage par renforcement pour allouer des VMs et des conteneurs sous contraintes de QoS. Évalué sur des charges de travail cloud simulées, IMARM a montré des améliorations significatives en termes de respect des SLA et d'efficacité énergétique par rapport aux heuristiques centralisées [70].

#### b. Cadre MAS Cloud

Ce cadre emploie des agents de surveillance pour chaque hôte, des agents de prédition utilisant des modèles de séries temporelles locales, et des agents d'approvisionnement mettant en œuvre une allocation par négociation. Déployé sur un banc d'essai OpenStack, MAS Cloud a démontré des gains substantiels en termes d'utilisation des ressources et de respect des SLA par rapport aux méthodes de référence [69].

#### c. Étude de cas des agents EC2 chez Amazon

Des recherches internes chez Amazon ont montré que des agents autonomes pouvaient sélectionner et allouer dynamiquement des types d'instances EC2 à travers plusieurs zones de disponibilité. Ce système à base d'agents a réduit les coûts d'approvisionnement par rapport à des stratégies statiques et géré les pics de charge avec un impact minimal sur les performances [71].

#### d. Réseau Multi-Agent Deep Q (MADQN)

Une approche d'apprentissage par renforcement profond multi-agent dans laquelle chaque agent utilise DQN pour planifier les tâches sur des noeuds virtualisés. Dans des expérimentations sur un simulateur de cluster de type Kubernetes, MADQN a surpassé le DQN mono-agent en obtenant **25% de réduction du temps de réponse et 22% de baisse des coûts opérationnels** [26].

#### e. MARL avec Politique Partagée pour la Gestion des Quotas

Les agents partagent un réseau de politique commun pour faire respecter dynamiquement les quotas dans une plateforme cloud. Cette approche MARL à politique partagée

a convergé **30% plus vite** que des apprenants indépendants et géré divers modèles de requêtes avec **plus de 90% de taux de succès** [72].

Le tableau 2.4 résume les principales contributions de recherche utilisant les systèmes multi-agents (MAS) pour la gestion des ressources cloud. Ces travaux intègrent des agents autonomes et des stratégies d'apprentissage par renforcement afin d'améliorer l'utilisation des ressources, le respect des SLA et l'efficacité des coûts. Le tableau présente la technique employée par chaque approche, les bénéfices démontrés ainsi que les limitations pratiques, en mettant l'accent sur la nature décentralisée et adaptative des MAS dans des environnements cloud dynamiques.

TABLE 2.4 – Summary of Surveyed Multi Agent-Based Works

Technology	Research Work (Authors, Year)	Technique Used	Proficiency Achieved	Strength	Weakness
Multi-Agent Systems	Belgacem et al. (2022)	MAS + Q-Learning (IMARM)	20% reduction in SLA violations and 15% energy savings.	Autonomous RL agents adapt to QoS goals and save energy.	RL training time and scalability can be challenging.
	MAS Cloud Framework (Year N/A)	Multi-agent (monitor/predict/negotiate)	18% higher resource utilization and 95% SLA adherence.	Decentralized agents improve utilization and reliability.	Negotiation overhead; complex coordination design.
	Amazon EC2 Agent (Year N/A)	Autonomous agents in EC2	12% cost reduction and robust spike handling.	Proven cost savings in practice.	Proprietary; limited generality.
	Pei et al. (2024)	Multi-Agent Deep Q-Network (MADQN)	25% faster response times and 22	Significantly improved scheduling performance.	Requires extensive training data; potential RL convergence issues.

### 2.5.3 Analyse comparative

- **Adaptabilité vs Complexité** : Les systèmes MARL (ex. : MARL à politique partagée, MADQN) s'adaptent aux changements en temps réel, mais introduisent une complexité d'entraînement et une non-stationnarité dans l'apprentissage multi-agent [72] [26].  
À l'inverse, les MAS à base de règles (ex. : agents EC2 d'Amazon) offrent un comportement prévisible avec peu d'effort d'entraînement, mais manquent de flexibilité pour s'adapter à des conditions nouvelles [71].
- **Scalabilité** : Les architectures MAS hiérarchiques décomposent la prise de décision afin d'améliorer la scalabilité dans de grands clusters, réduisant la communication inter-agent de **40%** selon des études d'évaluation [73]. Les approches MARL plates peuvent souffrir d'engorgements de communication à mesure que le nombre d'agents augmente.
- **Surcharge de coordination** : Les protocoles d'enchères et de négociation dans MAS Cloud assurent une distribution équitable des ressources, mais engendrent une latence proportionnelle au nombre d'agents impliqués. À l'inverse, le MARL à politique partagée contourne la négociation explicite via une coordination implicite fondée sur des signaux de récompense partagés [72][69].

### 2.5.4 Cas d'usage et intégration pratique

- **Intégration avec OpenStack et Kubernetes** : MAS Cloud s'intègre parfaitement avec OpenStack, en utilisant la télémétrie de Ceilometer et Heat Orchestration pour orienter les décisions des agents. Les frameworks de planification Kubernetes peuvent accueillir des agents sous forme de plugins capables de surpasser les ordonneurs par défaut grâce à des informations locales [69].
- **Services de courtage cloud (Cloud Brokerage)** : Des agents courtiers dans les clouds fédérés négocient des baux de ressources entre fournisseurs en tenant compte de critères multiples (coût, latence, conformité). Ces agents interagissent avec Terraform et Ansible pour automatiser le déploiement de l'infrastructure [74].
- **Continuité Edge–Cloud** : L'orchestration par agents s'étend aux noeuds en périphérie, où des agents légers gèrent des dispositifs contraints et se coordonnent avec les agents cloud pour le déport de charges. Cela permet une **réduction de 30 % de la latence bout-à-bout** dans les scénarios IoT [75].

## 2.6 Synthèse des Paradigmes pour l’Approvisionnement et la Prédition Intelligents des Ressources

La synthèse suivante consolide les résultats clés issus des quatre paradigmes explorés afin de dégager une perspective unifiée.

**Optimisation Métaheuristique :** Les algorithmes métaheuristiques basés sur des populations (par exemple : GA, PSO, ACO) excellent dans la recherche globale multi-objectifs : ils explorent de vastes espaces de solutions pour équilibrer coût, énergie, latence et objectifs SLA dans des environnements cloud hétérogènes. En s’appuyant sur une exploration stochastique et un contrôle adaptatif, ils produisent des plans d’allocation robustes qui surpassent les heuristiques statiques. Cependant, les métaheuristiques classiques manquent de mécanismes intégrés de prévision de la demande et opèrent souvent sur des instantanés statiques. Elles nécessitent un réglage manuel approfondi hors ligne (tailles de population, inertie, etc.) et peuvent se bloquer face à des charges éruptives ou non stationnaires. En pratique, les métaheuristiques pures peuvent converger lentement à mesure que les systèmes s’étendent (milliers de VM). Les recherches récentes les hybrident donc avec l’apprentissage ou le RL : par exemple, l’intégration de modèles prédictifs ou de LLMs pour guider la recherche accélère considérablement la convergence et l’adaptabilité. En résumé, les métaheuristiques offrent une puissante colonne vertébrale d’optimisation, mais doivent être augmentées pour répondre aux exigences de prévision en temps réel et de réglage automatique.

**Apprentissage Machine et Profond :** Les méthodes basées sur l’apprentissage offrent une approche complémentaire : les modèles ingèrent des données de télémétrie pour anticiper la charge et guider le scaling en temps réel. Les modèles ML classiques (forêts aléatoires, boosting, SVM) offrent de bonnes performances dans des scénarios relativement stables, tandis que les réseaux profonds (RNN LSTM/GRU) capturent les schémas temporels complexes sous des charges fluctuantes. Les grandes plateformes cloud (Azure, Alibaba, etc.) ont démontré une amélioration de l’utilisation, des économies de coûts et du respect des SLA grâce à la prévision ML/DL. Contrairement aux règles statiques, le ML/DL s’adapte aux modèles d’usage et se généralise à partir des données historiques. Toutefois, ces modèles ont leurs faiblesses connues : ils sont souvent des “boîtes noires” difficiles à interpréter, peuvent se dégrader lorsque les charges évoluent ou lorsque les données d’entraînement sont rares (démarrage à froid). Le surapprentissage et la sensibilité aux hyperparamètres constituent également des défis. Pour y remédier, les solutions modernes proposent le transfert d’apprentissage, l’estimation d’incertitude ou l’AutoML afin de renforcer leur robustesse. Globalement, le ML/DL apporte un socle prédictif essentiel

au SRPP — en servant de pilier de prévision — mais peut rencontrer des difficultés pour l’adaptation rapide en ligne et l’interprétabilité.

**IA Générative** : Elle injecte une forme d’intelligence plus riche dans l’approvisionnement. Les modèles tels que les GANs, VAEs, et surtout les grands modèles de langage (LLMs), peuvent synthétiser des données de charge réalistes et des représentations latentes, **enrichissant ainsi les ensembles d’entraînement et le contexte**. Par exemple, les architectures de prédition de charge basées sur des GANs ont atteint plus de 95 % de précision en génération de tendances en synthétisant des traces de charge. La GenAI peut intégrer un contexte non numérique (logs, comportement utilisateur, télémétrie multimodale) pour anticiper les besoins en ressources. Cela permet un dimensionnement plus nuancé et anticipatif : le système peut proposer des configurations inédites et tester des scénarios avant qu’ils ne surviennent. Dans le cadre de notre cluster orch–hyperviseur, les atouts de la GenAI en font un pilier architectural essentiel : elle produit des données d’entraînement enrichies, modélise des dépendances complexes, et génère des stratégies de réponse dynamique alignées avec les objectifs adaptatifs du cluster. En résumé, la GenAI étend le ML/DL en apportant des capacités de prédition **proactives et sensibles au contexte** que les modèles statiques ne peuvent égaler.

**Systèmes Multi-Agents (MAS)** : Les MAS mettent l’accent sur la prise de décision décentralisée et autonome, particulièrement adaptée aux clouds distribués. Dans les architectures MAS, chaque agent (au niveau de l’orchestration ou de l’hyperviseur) surveille et gère un sous-ensemble de ressources, en collaborant via des protocoles négociés. Cette autonomie locale et coordination permet au système de s’adapter rapidement aux fluctuations et pannes locales. Par exemple, des agents à base de règles ou pilotés par apprentissage par renforcement peuvent allouer des VM avec peu ou pas de supervision centrale, garantissant le respect des SLA dans des environnements hétérogènes. Dans un cluster orch–hyperviseur, les MAS apportent flexibilité et scalabilité : des agents légers en périphérie gèrent les tâches au niveau des nœuds, tandis que des agents de niveau supérieur gèrent les politiques globales, assurant tolérance aux pannes et orchestration en temps réel. Les MAS complètent également les méthodes prédictives : combinés à la GenAI, les agents peuvent utiliser des prévisions riches pour prendre des décisions d’approvisionnement locales. En effet, les MAS fournissent la **coordination adaptative** qui manque aux méthodes d’optimisation pure ou de ML. Seuls, les MAS nécessitent une conception robuste des agents, mais couplés à des prédicteurs puissants, ils forment une couche de contrôle résiliente et auto-optimisante.

**Cadre hybride GenAI + MAS** : La synthèse de la GenAI avec les MAS donne l’architecture SRPP la plus efficace. Les études empiriques montrent que cette combinaison

exploite le meilleur des deux mondes : la GenAI offre prévision et conscience contextuelle, tandis que les MAS assurent une exécution décentralisée et une adaptabilité. Concrètement, cela signifie intégrer des agents intelligents (possiblement basés sur des LLM) à chaque niveau d'orchestration et d'hyperviseur, partageant les insights et négociant les actions. Par exemple, un agent basé sur LLM pourrait interpréter la télémétrie en temps réel et générer des plans de scaling, ensuite appliqués de façon autonome par des agents locaux. La revue formelle conclut que **GenAI + MAS forment un socle puissant**, avec une intégration synergique permettant une gestion des ressources véritablement intelligente et en temps réel. Contrairement aux heuristiques statiques ou aux modèles ML isolés, cette approche hybride s'adapte en continu à la demande changeante : la GenAI anticipe et planifie, les MAS localisent et exécutent. Ce cadre unifié comble des lacunes clés : il ajoute une puissance prédictive long terme (comblant les angles morts des mét-heuristiques/ML) et une adaptation fine (au-delà de ce que permettent les modèles centralisés). Ainsi, pour notre cluster orchestration–hyperviseur, une architecture MAS pilotée par GenAI promet une solution SRPP **résiliente et auto-optimisante**, en phase avec les objectifs du projet.

## 2.7 Conclusion

Ce chapitre a présenté une revue approfondie des approches intelligentes pour l'**approvisionnement et la prédition intelligents des ressources** dans les environnements cloud et virtualisés. À travers l'exploration systématique de l'**optimisation métahéuristique**, des **modèles d'apprentissage automatique et profond**, des **systèmes d'IA générative** et des **architectures multi-agents**, nous avons montré comment chaque paradigme contribue de façon unique à relever les défis de la prévision dynamique des charges, de l'optimisation multi-objectif et de la prise de décision autonome.

Chaque méthode implique des compromis en termes de **vitesse, interprétabilité, adaptabilité et coût computationnel**, rendant nécessaire un cadre hybride combinant leurs forces respectives. Dans le contexte de notre projet — axé sur l'approvisionnement et la prédition au sein d'un cluster hybride orch-hyperviseur — nous avons identifié qu'un usage combiné de l'IA générative et des systèmes multi-agents (MAS) constitue une base puissante pour développer une infrastructure **résiliente et auto-optimisante**. Cela s'explique en grande partie par la **nature homogène et synergique** de leur intégration : la GenAI apporte les capacités prédictives et la conscience du contexte, tandis que les MAS assurent une coordination décentralisée et des décisions adaptatives. Ensemble, ils permettent une gestion intelligente des ressources, en temps réel, au sein d'environnements virtualisés complexes.

# Chapitre 3

## Conception du système et vue d'ensemble architecturale

### **3.1 Introduction à la conception proposée**

Les infrastructures cloud modernes font face à une complexité croissante dans la gestion des charges de travail dynamiques, des ressources hétérogènes et des exigences strictes de niveau de service. L'objectif principal de cette conception est d'introduire une architecture intelligente, pilotée par l'IA, qui améliore l'utilisation des ressources, assure une haute disponibilité et minimise les coûts opérationnels dans un cluster virtualisé. Spécifiquement, en intégrant des techniques de provisionnement avancées exploitant les LLM pour la prédiction de séries temporelles multi-étapes, notre conception cherche à exécuter des actions de provisionnement proactives à la fois au niveau des VM et des conteneurs. À un niveau élevé, l'architecture proposée englobe cinq couches interdépendantes : la **Couche d'Infrastructure Physique**, la **Couche de Virtualisation**, la **Couche d'Orchestration de Conteneurs**, la **Couche de Surveillance et de Gestion**, et la **Couche d'Agent IA**. L'innovation principale de cette conception réside dans l'intégration d'un agent de provisionnement basé sur les LLM au sein de la pile cloud, nous permettant d'obtenir des prédictions plus précises et contextuelles que les méthodes statistiques ou d'apprentissage automatique conventionnelles. La portée de cette conception se concentre principalement sur l'intégration et l'interaction entre la Couche d'Agent IA, l'orchestration et la virtualisation. Bien que la conception suppose la présence d'un cluster fonctionnel, elle fait abstraction des configurations réseau de bas niveau et de la sélection matérielle. Au lieu de cela, ce chapitre met l'accent sur les définitions conceptuelles des composants, les flux de données et les flux de contrôle, établissant ainsi une base solide pour l'implémentation et l'évaluation ultérieures.

### **3.2 Architecture conceptuelle**

Le système de gestion proactive des ressources piloté par l'IA proposé est organisé en cinq couches stratifiées, chacune responsable d'un domaine fonctionnel distinct. La Figure 3.1 illustre le diagramme de blocs conceptuel, distinguant entre les composants open source existants et les modules IA novateurs. Cette organisation en couches assure une séparation claire des préoccupations, améliore la maintenabilité et facilite l'évolution incrémentale.

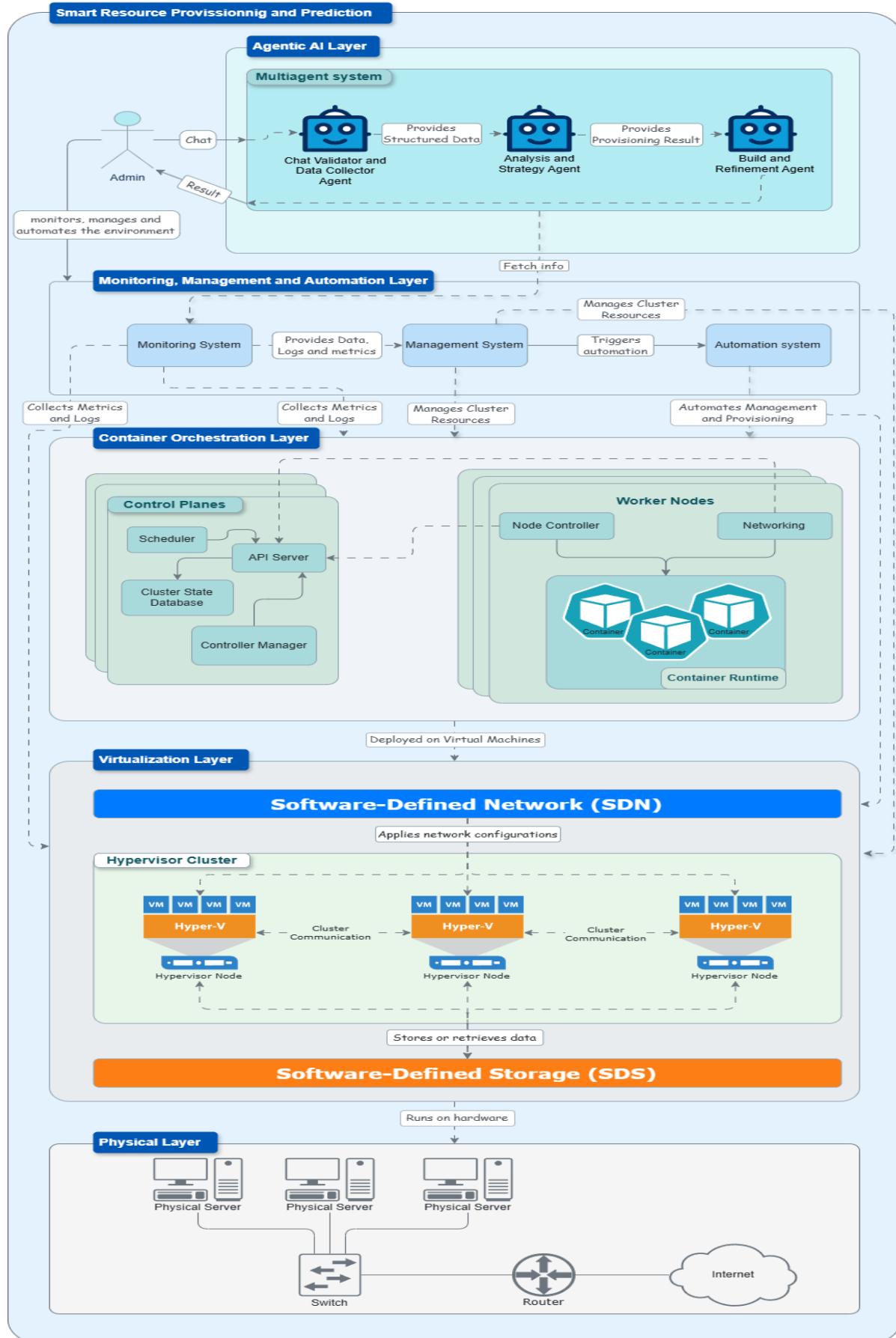


FIGURE 3.1 – Architecture globale

### — Couche Physique :

À la base se trouve la Couche Physique, comprenant des serveurs standard, des commutateurs et des routeurs. Ces dispositifs fournissent les ressources brutes de calcul, de stockage et de réseau.

### — Couche de Virtualisation :

Située directement au-dessus du substrat physique se trouve la Couche de Virtualisation, construite sur des clusters d'hyperviseurs. Ici, chaque nœud bare-metal héberge plusieurs machines virtuelles (VM), abstrayant le CPU, la mémoire et les E/S en partitions définies par logiciel. En parallèle, un contrôleur de réseau défini par logiciel (SDN) et un moteur de stockage défini par logiciel (SDS) fournissent une configuration programmatique des overlays réseau et des pools de stockage distribués. Ces abstractions découpent l'orchestration d'infrastructure des particularités matérielles et sous-tendent l'élasticité des couches en aval.

### — Couche d'Orchestration de Conteneurs :

Au-dessus des VM virtualisées réside la Couche d'Orchestration de Conteneurs. Le plan de contrôle comprenant le Serveur API, le Planificateur, le Gestionnaire de Contrôleurs et la base de données etcd traduit les manifestes d'état souhaité en actions de cluster. Les nœuds de travail hébergent des runtimes de conteneurs et des plugins réseau, exécutant des pods et facilitant la découverte de services. Cette couche gère le placement des charges de travail, les vérifications de santé et les mises à jour progressives, tout en restant agnostique à la logique de prévision ou de provisionnement proactif.

### — Couche de Surveillance, Gestion et Automatisation :

La strate suivante collecte la télémétrie à travers les hyperviseurs, les VM et Kubernetes. Un Système de Surveillance agrège les métriques et les journaux, alimentant à la fois les tableaux de bord en temps réel et les archives historiques. Un Système de Gestion consolide ces données pour appliquer les quotas de ressources. Lorsque les seuils de politique ou de capacité sont franchis, un Moteur d'Automatisation - piloté par des outils tels que Terraform, Ansible - ajuste programmatiquement les VM ou les répliques de conteneurs via une configuration déclarative. Cette couche fait ainsi le pont entre l'observabilité et l'exécution, mais sans intelligence intrinsèque pour anticiper la demande future.

### — Couche IA Agentique :

Le sommet de l'architecture est la Couche IA Agentique, où l'intelligence proactive est concentrée dans un système multi-agents. Trois agents autonomes : Validation, Analyse et Prévision, Provisionnement, et Amélioration - collaborent pour transformer la télémétrie brute et les entrées utilisateur en plans de provisionnement optimisés.

L'Agent de Validation examine d'abord les requêtes entrantes et l'état du système pour la cohérence. L'Agent d'Analyse et de Prévision emploie des modèles de séries temporelles basés sur des transformers pour prédire les tendances de charge de travail. Guidé par ces prévisions, l'Agent de Provisionnement génère des manifestes IaC précis. Enfin, l'Agent d'Amélioration affine ces manifestes via de grands modèles de langage, injectant des paramètres optimisés pour les performances et des motifs de meilleures pratiques. Toute communication inter-agents, ainsi que les interactions avec les couches de Surveillance et d'Automatisation, se déroulent via des interfaces RESTful et de files de messages bien définies, assurant un couplage lâche et une scalabilité horizontale.

### 3.3 Couches fondamentales de l'architecture proposée

#### 3.3.1 Couche Physique

La Couche Physique constitue le substrat fondamental de l'infrastructure, fournissant les capacités essentielles de calcul, de stockage et de réseau requises pour les opérations de niveau supérieur. Cette couche inclut les serveurs physiques, les commutateurs montés en rack et les routeurs passerelles, qui établissent ensemble la plateforme de base sur laquelle toute la logique de virtualisation et d'orchestration est construite.

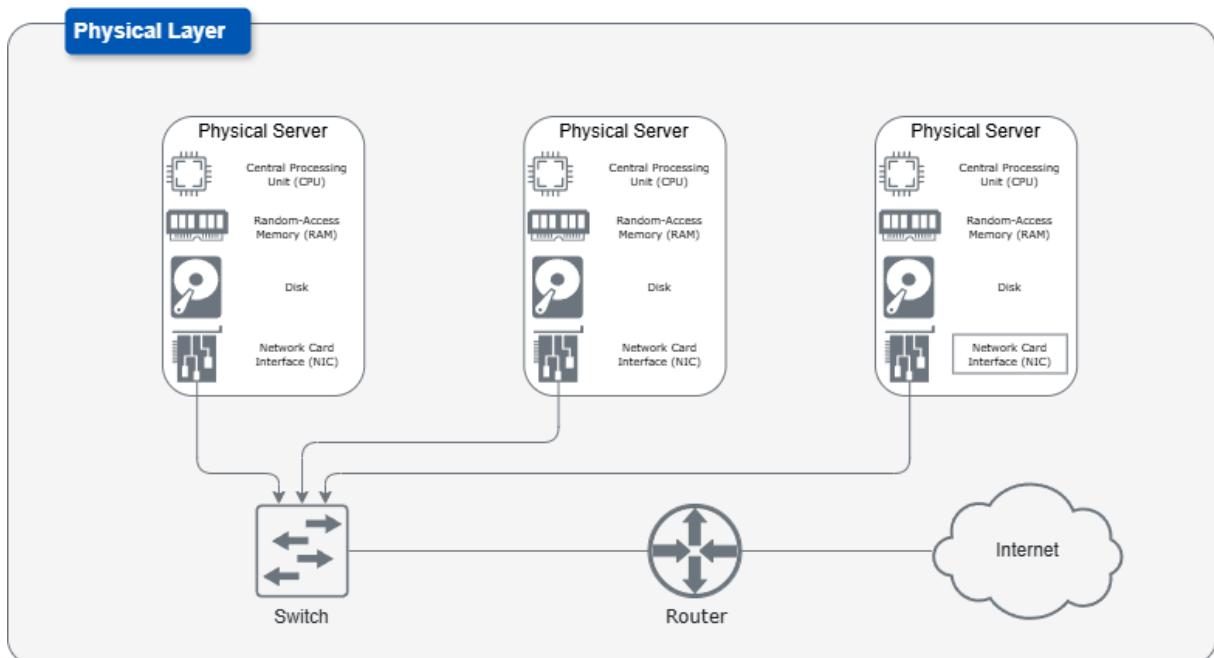


FIGURE 3.2 – Couche-Physique

Comme illustré dans la Figure 3.2, la couche physique est composée des composants

fondamentaux suivants :

- **Serveurs Physiques** : Chaque nœud serveur est équipé de :
  - Une Unité Centrale de Traitement (CPU) multi-cœurs pour exécuter les instructions et gérer les charges de travail.
  - Une Mémoire Vive Dynamique (RAM) pour fournir un stockage rapide et volatile aux processus actifs.
  - Des disques de stockage persistant (HDD ou SSD) pour les fichiers système, les images de VM et les journaux.
  - Une Carte d'Interface Réseau (NIC) pour la connectivité haute vitesse au réseau local (LAN).
- **Commutateur Réseau** : Les serveurs physiques sont interconnectés via un commutateur réseau de Couche 2/3. Ce commutateur forme l'épine dorsale de la communication intra-cluster, permettant aux nœuds de partager les charges de travail, répliquer les données et communiquer avec la couche de virtualisation.
- **Routeur et Passerelle** : Le commutateur se connecte à un routeur en amont, qui sert de passerelle entre le réseau local du centre de données et l'Internet externe. Le routeur gère la traduction d'adresses, les politiques de routage et les configurations de pare-feu.

Cette couche est volontairement conçue pour être agnostique au matériel et basée sur des composants standard, assurant la rentabilité et la scalabilité. Les limites d'abstraction définies ici permettent à la couche de virtualisation d'allouer dynamiquement les ressources (CPU, RAM, stockage, bande passante réseau) sans être liée à des topologies physiques spécifiques ou à des micrologiciels spécifiques aux fournisseurs.

### 3.3.2 Couche de Virtualisation

La Couche de Virtualisation fournit une abstraction critique entre le matériel physique sous-jacent et les couches supérieures d'orchestration de conteneurs et d'automatisation pilotée par l'IA. En employant un logiciel hyperviseur au-dessus des serveurs physiques, cette couche permet la consolidation des charges de travail, la mutualisation des ressources et l'isolation stricte des environnements de locataires ou d'applications. La Figure 3.3 présente un schéma détaillé de la Couche de Virtualisation, illustrant plusieurs nœuds hyperviseurs, des constructions de réseau virtuel et l'intégration de stockage partagé. Dans l'exposition suivante, chaque élément du diagramme fourni est décrit à son tour, avec un accent sur la communication inter-hyperviseurs, les liens de réseau virtuel et le stockage défini par logiciel (SDS).

## Chapitre 3. Conception du système et vue d'ensemble architecturale

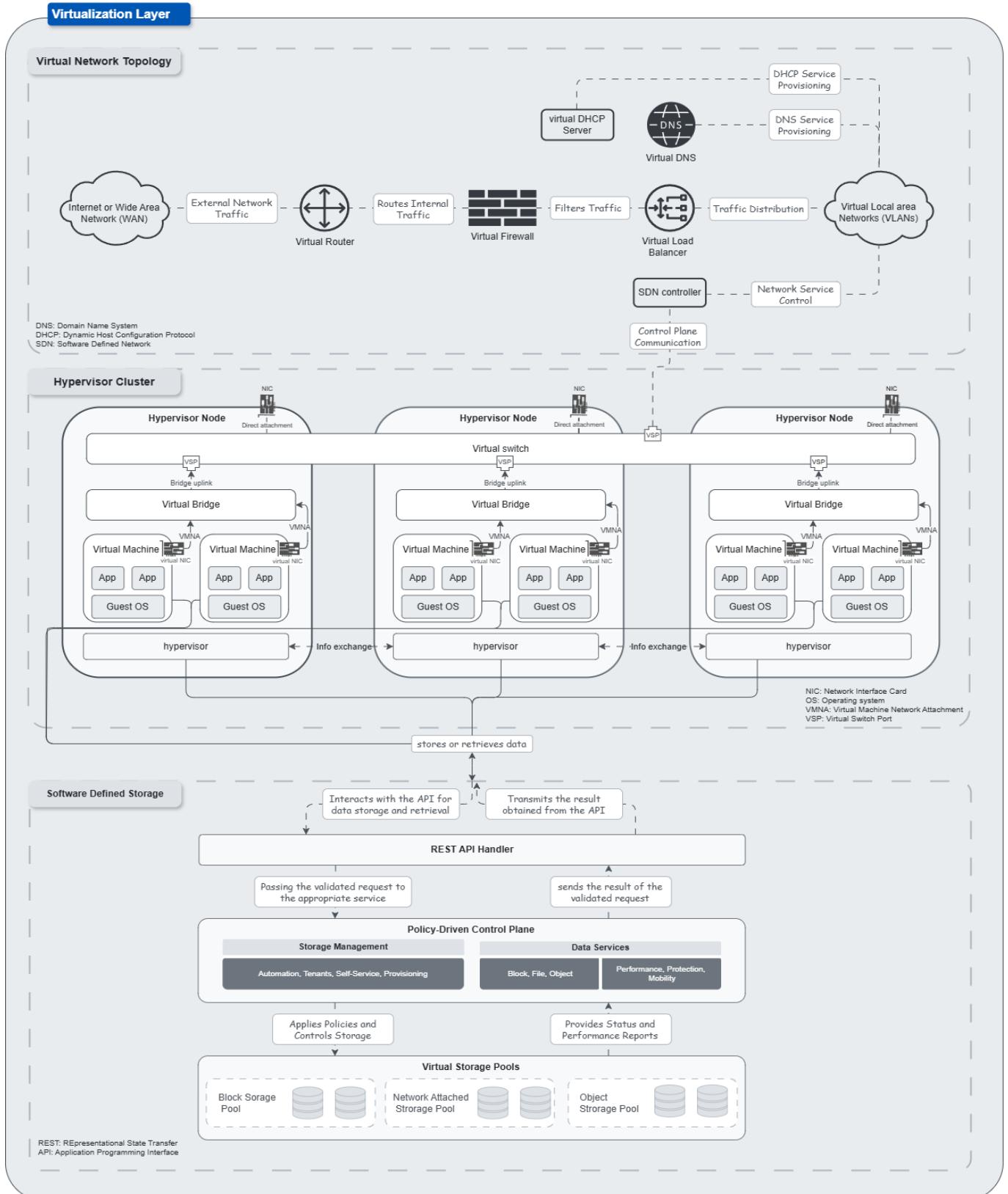


FIGURE 3.3 – Couche-de-Virtualisation

Comme illustré dans la Figure 3.3, la couche de virtualisation est composée des composants fondamentaux suivants :

- **Nœuds Hyperviseurs** : Chaque nœud hyperviseur représente un serveur physique exécutant un hyperviseur de Type-1. Au sein de chaque nœud :
  - **Logiciel Hyperviseur** :
    - Abstrait les ressources CPU, RAM et E/S en partitions virtuelles (vCPU, vRAM, disques virtuels, vNIC).
    - Gère les opérations de cycle de vie des VM (création, suppression, suspension, reprise, migration à chaud, capture d'instantané).
    - Applique l'isolation et les contextes de sécurité afin que la mémoire et les E/S de chaque VM restent séparées des VM co-résidentes.
    - Implémente des algorithmes de planification pour multiplexer les cœurs CPU physiques et le ballooning mémoire pour l'allocation dynamique de RAM.
  - **Instances de Machines Virtuelles (VM)** : Chaque VM s'exécute comme un serveur logiquement isolé, comprenant :
    - **Système d'Exploitation Invité** : Une instance OS complète s'exécutant au-dessus du matériel virtualisé.
    - **Charges de Travail d'Applications** : Une ou plusieurs applications ou services (blocs "App") qui s'exécutent sur l'OS Invité et génèrent des demandes CPU, mémoire et E/S.
    - **Dispositifs Virtuels** :
      - *vCPU(s)* : Threads CPU virtuels mappés par l'hyperviseur sur les cœurs CPU physiques.
      - *vRAM* : Mémoire virtuelle allouée à la VM, soutenue soit par la RAM physique soit par des fichiers d'échange sur le pool SDS.
      - *Disque Virtuel* : Un dispositif bloc présenté comme un fichier ou volume logique sur le moteur SDS.
      - *vNIC* : Une ou plusieurs interfaces réseau virtuelles qui connectent la VM au Pont Virtuel local.
  - **Communication de Cluster d'Hyperviseurs** :
    - *Bus de Cluster pour Battements de Cœur et Adhésion* : Les hyperviseurs échangent des messages de battement de cœur sur un réseau de gestion dédié pour suivre la santé des nœuds et la disponibilité des ressources.
    - *Canal de Migration à Chaud* : Lors de la migration d'une VM, l'hyperviseur source diffuse les pages mémoire et l'état CPU vers l'hyperviseur cible via ce canal, permettant une relocalisation avec un temps d'arrêt quasi-nul.

- *Métriques d'Usage de Ressources Partagées* : Les hyperviseurs partagent périodiquement les statistiques agrégées de charge CPU, pression mémoire et E/S réseau pour faciliter l'équilibrage de charge global et la haute disponibilité.
- **Composants de Réseau Virtuel** : Chaque nœud hyperviseur héberge un Pont Virtuel local qui se connecte à un Commutateur Virtuel à l'échelle du cluster :
  - **Pont Virtuel** :
    - Agit comme un commutateur de Couche 2 au sein d'un seul nœud hyperviseur, interconnectant toutes les vNIC des VM locales.
    - Transmet les trames entre les VM co-localisées sans quitter l'hôte, minimisant la latence intra-nœud.
    - Encapsule les paquets destinés aux VM distantes dans des paquets de tunnel overlay et les transmet au Commutateur Virtuel.
  - **Commutateur Virtuel (Réseau Overlay)** :
    - Représente un overlay de Couche 2 à l'échelle du cluster qui unifie les Ponts Virtuels à travers les hyperviseurs.
    - Maintient un mappage distribué MAC/IP-vers-VNI, géré par un contrôleur SDN ou un plan de contrôle distribué.
    - Permet une communication inter-nœuds transparente des VM :
      1. La vNIC de la VM A transmet une trame à son Pont Virtuel local.
      2. Le Pont Virtuel encapsule la trame dans un paquet tunnel et l'envoie à l'overlay du Commutateur Virtuel.
      3. Le Commutateur Virtuel désencapsule sur l'hyperviseur de destination et la livre via le Pont Virtuel de cet hôte à la vNIC de la VM B.
    - Supporte l'application de politiques réseau, la segmentation d'espaces de noms et le chiffrement overlay sans modifier les configurations de commutateurs physiques.
  - **NIC Virtuelles** :
    - Implémentées via des pilotes paravirtualisés ou des NIC émulées à l'intérieur de chaque VM.
    - Chemin sortant : Application dans l'OS Invité → pilote vNIC → émulation hyperviseur → Pont Virtuel.
    - Chemin entrant : Paquet overlay arrive au Commutateur Virtuel → transmis au Pont Virtuel → livré à la vNIC de la VM → l'OS Invité le traite.
- **Communication de Cluster Inter-Hyperviseurs** : Dans un environnement cloud, la communication entre hyperviseurs est cruciale pour des tâches comme

la migration de machines virtuelles, l'équilibrage de charge et l'assurance de haute disponibilité. Elle permet aux hyperviseurs de partager des informations sur l'utilisation des ressources et le statut des machines virtuelles pour maintenir la stabilité du système et optimiser les performances. Spécifiquement :

- *Gestion des Battements de Cœur et de l'Adhésion* : Un Démon d'Adhésion de Cluster sur chaque hyperviseur échange des messages de battement de cœur périodiques sur le réseau de gestion. L'échec de réception d'un battement de cœur dans un intervalle spécifié déclenche un basculement HA automatique pour les VM hébergées.
- *Coordination de Migration à Chaud* : Pendant une migration à chaud, l'hyperviseur source transfère les pages mémoire et l'état CPU de manière incrémentale vers l'hyperviseur cible à travers le canal inter-hyperviseurs. Cela minimise le temps d'arrêt en ne mettant en pause la VM que pour une copie d'état finale petite.
- *Échange d'Informations d'Équilibrage de Charge* : Les hyperviseurs transmettent des métriques en temps réel (par exemple, utilisation CPU, usage mémoire, débit réseau) sur ce canal, permettant aux planificateurs de couches supérieures (par exemple, Kubernetes ou agents de provisionnement IA) de redistribuer les charges de travail et d'éviter les points chauds.
- *Synchronisation des Métadonnées SDS* : Parce que le stockage partagé est essentiel pour un environnement cloud, les hyperviseurs répliquent et synchronisent les métadonnées SDS (par exemple, allocations de volumes, statut de réPLICATION, catalogues d'instantanés) via le réseau inter-hyperviseurs. Cette coordination assure la cohérence des données et supporte la mobilité des VM et la protection des données.
- **Intégration du Stockage Défini par Logiciel (SDS)** : Le stockage partagé est essentiel pour un environnement cloud. Il permet à plusieurs hyperviseurs d'accéder aux mêmes données, ce qui est nécessaire pour des fonctionnalités comme la mobilité des machines virtuelles et la protection des données. Il simplifie la gestion et assure la cohérence des données à travers l'infrastructure. La couche SDS comprend :
  - *Pool de Stockage Bloc Distribué* : Chaque hyperviseur contribue des disques locaux (HDD / SSD) à un cluster SDS distribué (par exemple, Ceph, GlusterFS, ou Proxmox ZFS). Le pool collectif apparaît comme un seul volume de stockage logique.
  - *RéPLICATION et Redondance* : Les données sont répliquées à travers plusieurs nœuds hyperviseurs selon un facteur de réPLICATION configurable garantissant la durabilité et facilitant la récupération rapide des pannes de nœuds.
  - *Provisionnement Mince et Instantanés* : Les disques virtuels des VM sont

alloués à la demande (provisionnement mince), et les instantanés au niveau bloc permettent des sauvegardes VM instantanées et la création rapide de clones pour des fins de test ou de mise à l'échelle.

- *QoS et Planification E/S* : Le moteur SDS applique des limites de taux E/S par VM pour atténuer les effets de voisins bruyants et assure que les charges de travail haute priorité reçoivent une bande passante suffisante et une faible latence.
- *Chemin d'Accès au Stockage Partagé* :
  1. Quand une VM émet une opération de lecture/écriture, la couche E/S de l'hyperviseur transmet la requête au moteur SDS sur le réseau de stockage.
  2. Le moteur SDS récupère ou valide les données de la réplique appropriée sur les disques physiques et les retourne à l'hyperviseur, qui à son tour les livre au pilote de disque virtuel de la VM.
  3. Parce que tous les hyperviseurs puisent du même pool SDS, une image VM peut rester cohérente lors de la migration.

Ce substrat de virtualisation détaillé comprenant les noeuds hyperviseurs, les VM, les composants de réseau virtuel, la communication inter-hyperviseurs et le SDS partagé permet :

- **Partitionnement Élastique des Ressources** : Les VM peuvent être dimensionnées correctement pour les CPU, la mémoire et le stockage, et de nouvelles VM peuvent être instanciées à la demande.
- **Isolation Stricte et Sécurité** : Les contextes de calcul, mémoire, réseau et stockage de chaque VM sont isolés des VM co-résidentes, assurant le confinement des pannes et la sécurité multi-locataires.
- **Haute Disponibilité et Migration à Chaud** : Les noeuds hyperviseurs peuvent migrer à chaud les VM avec un temps d'arrêt minimal. La réplication SDS assure que les disques VM restent disponibles même si un hyperviseur échoue.
- **Topologie Réseau Programmable** : Les Ponts Virtuels et l'overlay du Commutateur Virtuel à l'échelle du cluster découpent la communication VM de la topologie réseau physique, facilitant la mobilité transparente des VM et l'application de politiques réseau.
- **Gestion Unifiée du Stockage** : Le pool SDS fournit une interface de stockage cohérente pour tous les hyperviseurs, simplifiant le provisionnement VM, le clonage et les flux de travail d'instantanés.

En implémentant cette Couche de Virtualisation complète reflétant fidèlement le diagramme d'architecture fourni, le système établit une fondation robuste, flexible et programmable sur laquelle la Couche d'Orchestration de Conteneurs et les couches subsé-

quentes de Surveillance et Automatisation et IA Agentique construisent des capacités de gestion proactive et intelligente des ressources.

### **3.3.3 Couche d'Orchestration**

La Couche d'Orchestration représente le niveau de contrôle stratégique de l'infrastructure cloud-native. Elle est chargée de la gestion déclarative des charges de travail, de la découverte de services, de la planification de conteneurs, de la récupération de pannes et de l'automatisation de la scalabilité. Cette couche abstrait la complexité de la gestion du cycle de vie des conteneurs et de la coordination multi-hôtes en fournissant un plan de contrôle distribué et une interface uniforme pour les utilisateurs et opérateurs. Elle se situe au-dessus du substrat de virtualisation et gouverne l'exécution des applications conteneurisées à travers les ressources de calcul distribuées.

## Chapitre 3. Conception du système et vue d'ensemble architecturale

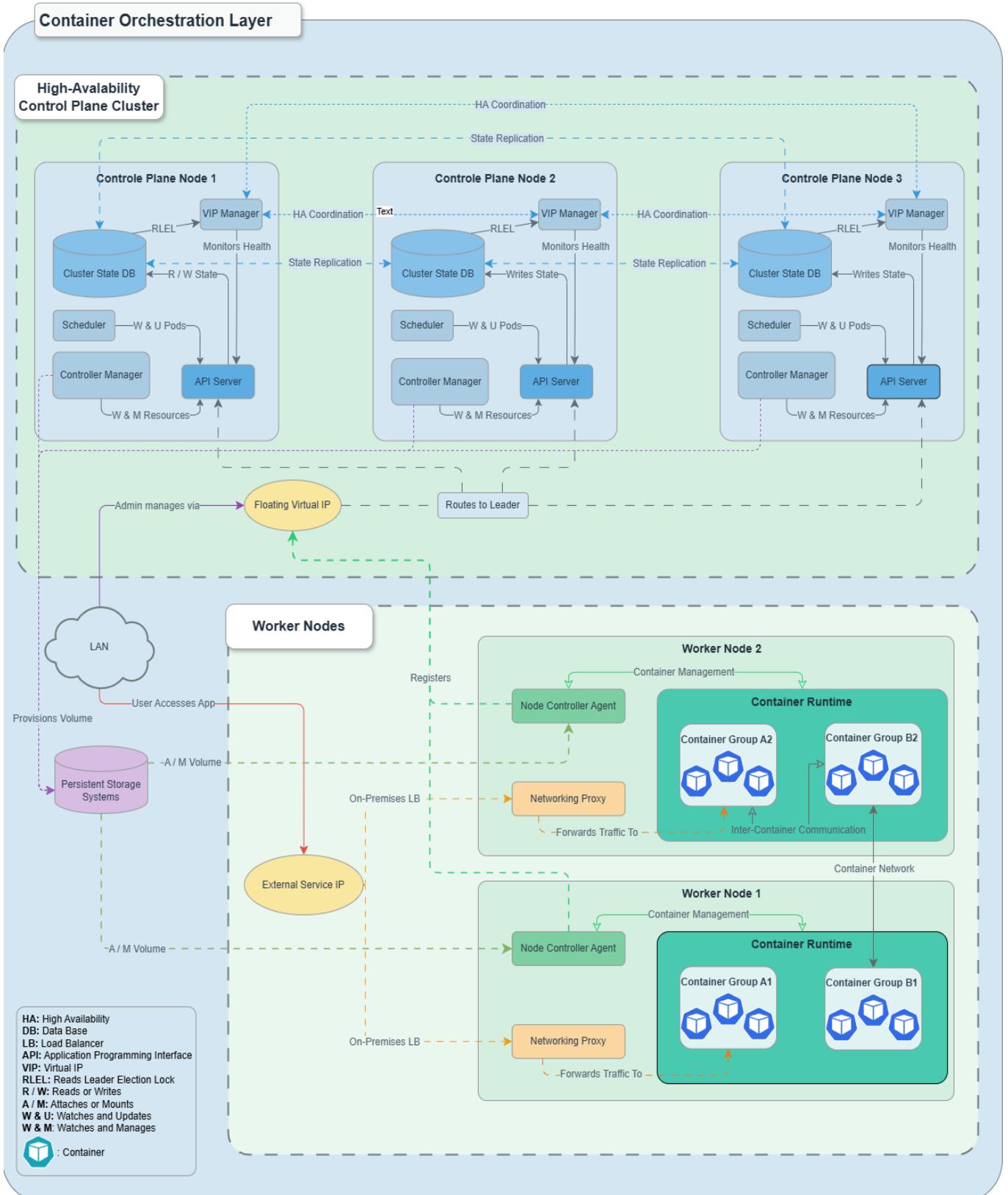


FIGURE 3.4 – Couche-d’Orchestration-de-Conteneurs

Comme illustré dans la Figure 3.4, cette couche est composée de plusieurs modules logiquement interconnectés distribués à travers les nœuds de plan de contrôle et de travail :

- **Cluster de Plan de Contrôle Haute Disponibilité (HA)** : Le plan de contrôle gouverne l'état entier du cluster et assure une prise de décision cohérente en utilisant un consensus distribué. Il comprend trois composants principaux par nœud :
  - **Base de Données d'État de Cluster (par exemple, etcd)** : Un magasin clé-valeur hautement cohérent et tolérant aux pannes utilisé pour persister la configuration entière du cluster, incluant les métadonnées, les états de ressources, les définitions de charges de travail et les secrets.
  - **Serveur API** : Sert de point d'entrée au plan de contrôle. Il authentifie les requêtes, valide les schémas et expose des interfaces RESTful pour les interactions de cluster. Tous les composants internes interagissent via le serveur API.
  - **Planificateur et Gestionnaire de Contrôleurs** : Le planificateur assigne les pods aux nœuds appropriés basé sur la disponibilité actuelle des ressources et les politiques de placement. Le gestionnaire de contrôleurs exécute des boucles de contrôle qui assurent que l'état actuel du cluster converge vers l'état désiré défini dans la base de contrôle.
  - **Gestionnaire VIP et Mécanisme d'Élection de Leader** : Parmi les nœuds du plan de contrôle, un est élu comme leader en utilisant un algorithme d'élection de leader distribué (par exemple, basé sur des baux dans etcd). Seul le nœud leader élu a l'autorité de provisionner des volumes, assurant la cohérence des données et évitant les écritures concurrentes aux backends de stockage.
- **IP Virtuelle Flottante (VIP)** : Une IP virtuelle hautement disponible abstrait l'emplacement physique du serveur API. Elle redirige dynamiquement le trafic administratif vers le leader actuel, maintenant une accessibilité transparente en cas de basculement de nœud de contrôle. Les routes VIP sont automatiquement mises à jour pour refléter les changements de leadership.
- **Nœuds de Travail** : Chaque nœud de travail est un hôte d'exécution responsable d'exécuter les charges de travail conteneurisées. Il inclut :
  - **Agent Contrôleur de Nœud (par exemple, kubelet)** : Ce démon enregistre le nœud avec le plan de contrôle et rapporte périodiquement les métriques de santé. Il assure que les pods assignés au nœud se conforment aux spécifications fournies par le plan de contrôle.
  - **Runtime de Conteneurs** : Ce module (par exemple, containerd ou CRI-O) gère le cycle de vie de bas niveau des conteneurs, incluant la récupération d'images, l'instanciation, l'exécution et la démolition.
  - **Proxy de Réseau (par exemple, kube-proxy)** : Gère les règles de réseau virtuel pour chaque nœud, permettant l'équilibrage de charge à travers les

services et maintenant des politiques réseau cohérentes.

- **Groupes de Conteneurs (Pods)** : Les plus petites unités déployables dans la couche d'orchestration, les pods encapsulent un ou plusieurs conteneurs étroitement couplés qui partagent le même espace de noms IP et les volumes persistants. Ces groupes sont orchestrés comme des unités uniques pour la mise à l'échelle, les vérifications de santé et l'isolation réseau.
- **Chemins de Communication et Relations** :
  - **Coordination du Plan de Contrôle** : Chaque nœud de contrôle maintient une coordination continue avec ses pairs via des vérifications de santé, la réplication d'état et les protocoles de leadership HA.
  - **Communication Inter-Nœuds** : Comme montré dans le diagramme, un sous-ensemble de groupes de conteneurs (pods) s'engage dans la communication inter-nœuds, essentielle pour les microservices distribués. Ce n'est pas une exigence universelle ; de nombreux pods opèrent indépendamment sur leurs propres nœuds.
  - **Communication Inter-Conteneurs** : Au sein de chaque pod, les conteneurs communiquent via des interfaces localhost et des volumes partagés, fournissant une intégration étroite et une latence minimale.
  - **Réseau de Conteneurs** : Les conteneurs qui s'étendent à travers les pods ou nœuds utilisent le réseau overlay de conteneurs (par exemple, Flannel, Calico) pour la communication, supportant la découverte de services et la résolution DNS.
  - **IP de Service Externe** : Pour permettre l'accès depuis l'extérieur du cluster (par exemple, trafic utilisateur ou requêtes API), les services peuvent être exposés via des IP externes ou des contrôleurs d'ingress, soutenus par un équilibrEUR de charge.
- **Systèmes de Stockage Persistant** : Ces systèmes permettent aux pods de conserver l'état à travers les redémarrages et fournissent des montages de volumes abstraits des backends de stockage physiques. Le provisionnement de tels volumes est coordonné exclusivement par le leader élu du plan de contrôle pour prévenir les conditions de course et assurer l'intégrité dans les environnements concurrents. Les classes de stockage et les plugins de volumes (par exemple, CSI) abstraient les détails de stockage physique et fournissent des capacités de provisionnement dynamique.

**Note de Conception** : Seul le leader dans le cluster de plan de contrôle est autorisé à provisionner des volumes persistants. Cette contrainte est imposée par l'architecture des systèmes de consensus distribués, qui visent à maintenir une forte cohérence dans l'état partagé à travers les acteurs concurrents. De plus, la figure illustre sélectivement un seul canal de communication inter-nœuds et inter-pods pour souligner que tous les groupes

de conteneurs ne participent pas à de telles relations. Chaque groupe de conteneurs doit s’engager dans au moins une des suivantes :

- Communication Inter-Conteneurs (au sein du même pod).
- Communication Réseau de Conteneurs (à travers les pods ou noeuds).
- Exposition via une IP de Service Externe (pour la communication avec les clients extérieurs au cluster).

Cette architecture en couches applique la haute disponibilité, l’élasticité et la gouvernance déclarative. Elle découpe les applications de l’infrastructure, facilite la récupération autonome et supporte les mises à jour sans temps d’arrêt. En tant que telle, elle est une pierre angulaire des systèmes cloud-natifs modernes.

### **3.3.4 Couche de Surveillance, Gestion et Automatisation**

La Couche de Surveillance, Gestion et Automatisation représente le niveau logique supérieur de l’architecture, dédiée à assurer la visibilité, l’opérabilité et le contrôle adaptatif sur l’infrastructure virtualisée et orchestrée. Elle intègre la collecte de télémétrie, l’évaluation basée sur des règles, l’automatisation déclenchée par l’utilisateur ou la machine, et les pipelines de prise de décision dynamique. Cette couche permet aux administrateurs d’observer les métriques en temps réel, de corrélérer les tendances historiques, d’exécuter des stratégies d’atténuation et d’automatiser les flux de travail récurrents, formant l’épine dorsale de la gestion intelligente d’infrastructure.

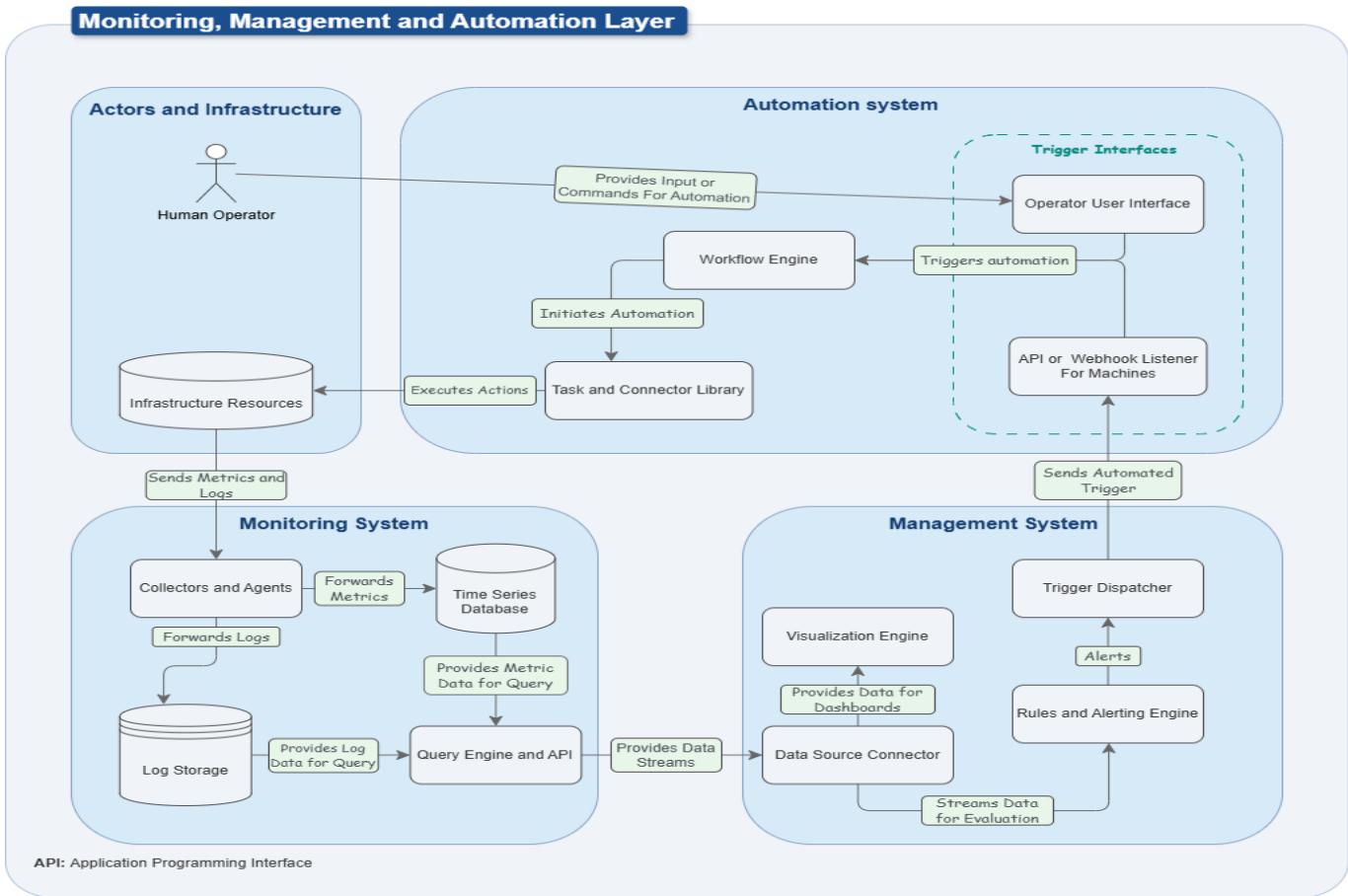


FIGURE 3.5 – Couche de Surveillance, Gestion et Automatisation

Comme illustré dans la Figure 3.5, cette couche est composée de quatre sous-systèmes interconnectés, chacun remplissant des responsabilités distinctes :

— **Acteurs et Infrastructure :**

- **Ressources d'Infrastructure** : Représente les cibles surveillées, incluant les nœuds de calcul, conteneurs, VM, sous-systèmes de stockage et composants réseau. Ces ressources émettent de la télémétrie opérationnelle et des journaux, qui servent d'entrée brute pour les systèmes en aval.

— **Système de Surveillance :**

- **Collecteurs et Agents** : Démons légers ou sidecars déployés à travers les nœuds d'infrastructure qui collectent les métriques (par exemple, usage CPU, consommation mémoire) et les journaux (par exemple, systemd, journaux d'applications) de l'environnement sous-jacent.
- **Stockage de Journaux** : Un système centralisé pour ingérer et persister les données de journaux. Ce backend de stockage supporte les requêtes basées sur des index et le découpage par plage temporelle pour l'analyse de cause racine et les audits de conformité.

- **Base de Données de Séries Temporelles (TSDB)** : Une base de données hautement performante optimisée pour gérer les flux de métriques avec une cardinalité élevée et une granularité temporelle. Elle permet la rétention à long terme des métriques de performance.
- **Moteur de Requête et API** : Fournit des API pour récupérer les métriques ou journaux de leurs backends respectifs. Ces interfaces sont souvent consommées par les règles d'alerte, les tableaux de bord de visualisation et les outils d'analyse externes.
- **Système de Gestion** :
  - **Connecteur de Source de Données** : S'intègre avec des sources externes telles que les TSDB, agrégateurs de journaux ou API cloud. Il diffuse la télé-métrie structurée dans le sous-système de gestion pour un traitement ultérieur.
  - **Moteur de Règles et d'Alertes** : Évalue les seuils définis par l'utilisateur et les conditions comportementales sur les données entrantes. Lorsque les critères prédéfinis sont remplis, il émet des alertes pour déclencher des actions réactives.
  - **Répartiteur de Déclencheurs** : Agit comme un courtier qui interprète et route les alertes en formats actionnables - soit vers des interfaces humaines (par exemple, tableaux de bord email) soit vers des pipelines d'automatisation.
  - **Moteur de Visualisation** : Convertit les données interrogées en tableaux de bord graphiques, jauge et graphiques de séries temporelles pour l'inspection humaine et le support de décision.
- **Système d'Automatisation** :
  - **Interfaces de Déclenchement** :
    - **Interface Utilisateur Opérateur** : Fournit une GUI pour l'interaction manuelle, telle que déclencher des flux de travail, injecter des paramètres ou acquitter des alertes.
    - **Écouteur API ou Webhook pour Machines** : Offre des points de terminaison programmables pour recevoir des événements des systèmes en amont ou des pipelines CI/CD, permettant l'automatisation pilotée par les événements.
  - **Moteur de Flux de Travail** : Sert d'exécuteur de logique central. Il consomme les déclencheurs, valide les préconditions et orchestre les tâches définies dans des flux de travail structurés (par exemple, YAML ou DSL).
  - **Bibliothèque de Tâches et Connecteurs** : Un référentiel de scripts prédéfinis, modules ou API capables d'interfacer avec les ressources d'infrastructure. Ces tâches incluent le provisionnement de ressources, le redémarrage de pods, la mise à l'échelle de nœuds ou la notification de systèmes externes.

### **Sémantique Opérationnelle et Flux :**

1. Les ressources d'infrastructure émettent des métriques et journaux vers les collecteurs, qui les transmettent aux backends de stockage dédiés (stockage de journaux et TSDB).
2. Le système de surveillance expose ces flux de données à l'API de requête, que le système de gestion utilise pour évaluer continuellement les règles et seuils.
3. Quand une règle est déclenchée (par exemple, usage CPU élevé sur un nœud de travail), le moteur d'alerte transmet un événement au répartiteur de déclencheurs.
4. Le répartiteur invoque le système d'automatisation, où soit un opérateur humain soit un écouteur API fournit des entrées contextuelles.
5. Le moteur de flux de travail initie une réponse prédéfinie (par exemple, mise à l'échelle d'un groupe de conteneurs), utilisant sa bibliothèque de tâches pour communiquer en retour avec les ressources d'infrastructure.

Cette conception de contrôle en boucle fermée assure un retour et une correction rapides dans les environnements dynamiques, permettant la mise à l'échelle prédictive, l'auto-guérison des pannes et un fardeau minimal pour l'opérateur. Elle supporte également les meilleures pratiques d'observabilité, telles que les "quatre signaux dorés" (latence, trafic, erreurs, saturation), tout en s'alignant avec les paradigmes GitOps et SRE.

### **3.3.5 Couche Agentique**

La Couche Agentique introduit une abstraction novatrice dans l'architecture : un sous-système de Provisionnement et de Prédition de Ressources Intelligent construit sur un paradigme de Système Multi-Agents (SMA). Dans ce contexte, les agents sont des entités logicielles autonomes et collaboratives dotées de capacités spécialisées telles que la perception, le raisonnement et l'actuation. Ces agents orchestrent collectivement la traduction d'intentions utilisateur de haut niveau en artefacts d'infrastructure déployables à travers une coordination et une décomposition distribuée des tâches. Cette couche exploite des agents avancés basés sur des LLM qui allient la compréhension du langage à l'intégration d'outils. Comme détaillé dans la Figure 3.6, le système utilise trois agents pour la collecte, l'analyse et la synthèse des informations. Ce flux de travail intelligent traduit les instructions de l'utilisateur en actions automatisées en ancrant les requêtes dans l'état actuel de l'infrastructure. Pour codifier les bonnes pratiques et garantir l'exactitude, le système puise dans des sources de connaissances distribuées, en utilisant notamment la technique RAG (Retrieval-Augmented Generation) pour interroger une base de données de documentation interne. Cela permet aux agents de compléter leurs connaissances avec des normes vérifiées provenant de documents de confiance et de code public, produisant ainsi des artefacts fiables et contextualisés. L'inclusion de cette couche permet au système

### **Chapitre 3. Conception du système et vue d'ensemble architecturale**

---

d'opérer avec un haut degré de proactivité et d'adaptabilité, permettant le provisionnement sensible au contexte, la génération automatisée de configuration et la validation contre à la fois l'état d'exécution et les directives de meilleures pratiques. Les sous-sections suivantes détaillent les responsabilités et mécanismes internes de chaque agent composant le SMA. Le flux de travail, comme montré dans la Figure 3.6, commence avec une invite utilisateur, suivie d'un traitement séquentiel à travers trois agents :

- Agent 1 : Validateur de Chat et Collecteur de Données (Collector)
- Agent 2 : Agent d'Analyse et de Stratégie (Analyser)
- Agent 3 : Agent de Construction et de Raffinement (Builder)

Ensemble, ils livrent une transformation complète d'intention en langage naturel de haut niveau en manifestes d'infrastructure déployables et sensibles au contexte.

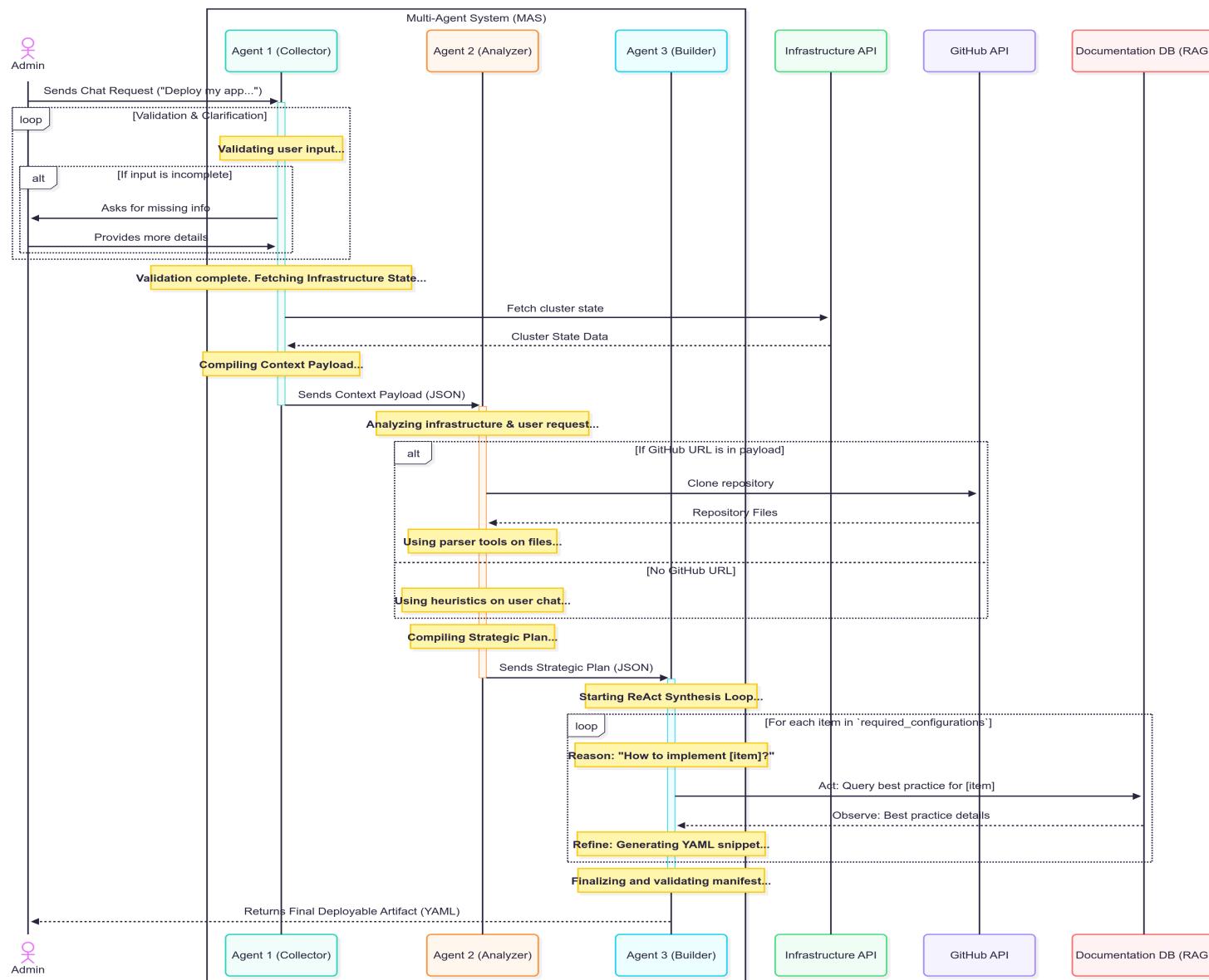


FIGURE 3.6 – Coordination du Flux de Travail Agentique dans le Système Multi-Agents (SMA)

Les sections ci-dessous présentent le rôle, l'architecture interne et la logique opérationnelle de chaque agent.

#### Agent 1 : Agent Validateur de Chat et Collecteur de Données (Collector)

Le premier composant du Système Multi-Agents (SMA) est l'Agent Validateur de Chat et Collecteur de Données. Cet agent fonctionne comme l'interface de première ligne du système pour l'interaction utilisateur et l'acquisition de contexte. Il joue un double rôle : (i) vérifier la validité et la faisabilité de la requête d'infrastructure de l'utilisateur, et (ii) enrichir la compréhension contextuelle de cette requête en collectant des informations spécifiques à la plateforme. Son architecture incarne un pipeline de raisonnement-validation-enrichissement et sert d'initiateur de l'orchestration autonome des tâches au sein du sous-système agentique.

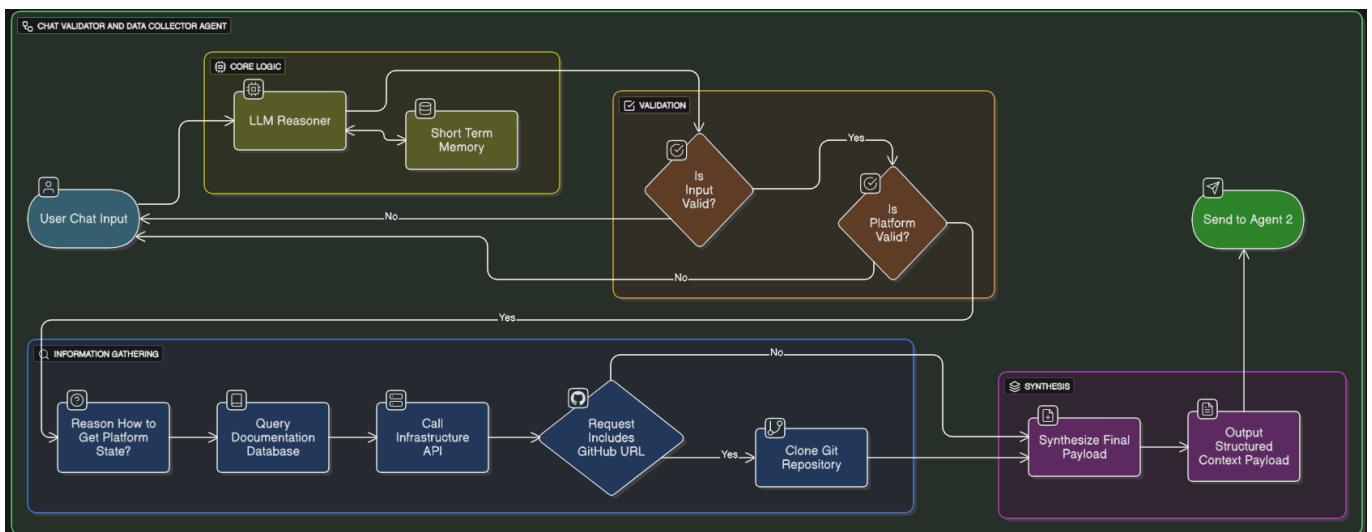


FIGURE 3.7 – Agent 1 - Flux de Travail du Validateur de Chat et Collecteur de Données

Comme illustré dans la Figure 3.7, l'agent comprend quatre modules étroitement intégrés : Logique Centrale, Validation, Collecte d'Informations et Synthèse.

##### — Module de Logique Centrale :

- L'interaction commence avec l'utilisateur soumettant une requête liée à l'infrastructure en forme libre via le langage naturel.
- Le *LLM Reasoner*, guidé par l'ingénierie des invites et des modèles d'instruction en few-shot, analyse et interprète l'intention de l'entrée utilisateur.
- Ce processus de raisonnement est enrichi par un magasin de **Mémoire à Court Terme** qui conserve le contexte temporaire tel que les identifiants de plateforme précédemment résolus ou les identifiants API et les interactions humaines.

- La sortie de raisonnement est transmise en aval pour validation structurelle.
- **Module de Validation :**
  - Ce module effectue deux vérifications de validation indépendantes :
    1. **Vérification de Validité d'Entrée** : Assure que la requête utilisateur contient suffisamment de détails structurels et sémantiques pour être traitée de manière significative (par exemple, spécifie une plateforme, un objectif ou une cible opérationnelle).
    2. **Vérification de Validité de Plateforme** : Vérifie si la plateforme référencée existe, est supportée ou peut être interrogée programmatiquement en utilisant les API disponibles.
  - Les entrées invalides ou ambiguës sont renvoyées à l'utilisateur avec des invites de clarification, permettant une boucle de rétroaction qui améliore la résilience aux requêtes incomplètes.
- **Module de Collecte d'Informations :**
  - Une fois l'entrée validée, l'agent tente d'enrichir le contexte d'entrée à travers plusieurs sources de données :
    1. **Raisonnement d'État de Plateforme** : L'agent interroge son module de raisonnement interne pour inférer la meilleure méthode pour récupérer l'état actuel de l'infrastructure (par exemple, en utilisant un appel API ou en inspectant un référentiel).
    2. **Interrogation de Base de Données de Documentation** : Une recherche sémantique est menée à travers la documentation d'infrastructure indexée pour extraire les exigences de déploiement, contraintes et comportements par défaut.
    3. **Appel d'API d'Infrastructure** : L'agent récupère les métadonnées en temps réel telles que les charges de travail en cours d'exécution, quotas de ressources ou statuts de nœuds depuis l'orchestrateur ou l'API cloud.
    4. **Clonage de Référentiel GitHub** : Si la requête de l'utilisateur inclut une URL GitHub, l'agent initie une opération de clonage de référentiel et analyse le contenu (par exemple, Dockerfiles, manifestes, plans Terraform) pour extraire les descripteurs de déploiement pertinents.
  - Ces étapes sont activées conditionnellement basées sur la présence de signaux dans l'entrée utilisateur et les métadonnées de plateforme.
- **Module de Synthèse :**
  - Toutes les informations récupérées et inférées sont agrégées dans un format cohésif et structuré.

- L'agent synthétise ces données en une **Charge Utile Finale**, qui inclut :
  - L'intention utilisateur extraite.
  - La cible de plateforme validée.
  - Les métadonnées d'infrastructure pertinentes.
  - Les extraits de configuration ou de code analysés (si applicable).
- La charge utile est sérialisée en une **Charge Utile de Contexte Structurée**, qui se conforme à un schéma interprétable par l'Agent 2 (par exemple, JSON ou YAML).
- Cette charge utile est émise via le déclencheur "Envoyer à l'Agent 2", transitionnant formellement l'exécution vers l'agent suivant dans le pipeline SMA.

Cet agent encapsule les principes de planification réactive et d'inférence contextuelle. Sa structure modulaire assure l'adaptabilité à travers les domaines, lui permettant de généraliser la logique de gestion d'entrée et de supporter de nouvelles intégrations de plateforme. À travers la validation en couches, l'enrichissement structuré et le raisonnement systématique, l'Agent 1 assure que les agents subséquents opèrent sur une fondation robuste et sémantiquement cohérente.

### **Agent 2 : Agent d'Analyse et de Stratégie (Analyser)**

La deuxième étape du Système Multi-Agents (SMA) est l'Agent d'Analyse et de Stratégie, qui fonctionne comme le noyau analytique. L'agent est responsable de :

- Interpréter la charge utile structurée et formuler une stratégie d'exécution basée sur les caractéristiques de l'application et le statut de l'infrastructure.
- Synthétiser les objectifs de déploiement en un plan de provisionnement lisible par machine qui fait le pont entre le contexte descriptif et l'orchestration actionnable.

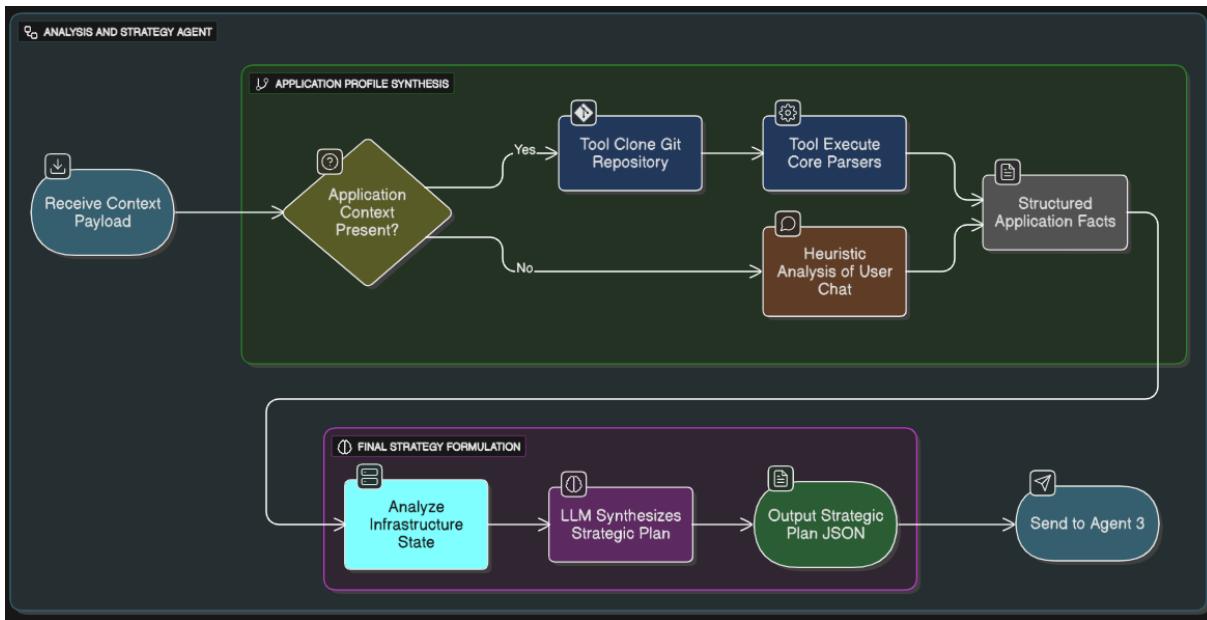


FIGURE 3.8 – Agent 2 - Flux de Travail de Formulation d'Analyse et de Stratégie

Comme montré dans la Figure 3.8, le flux de travail pour cet agent se déroule à travers deux modules fonctionnels majeurs : Synthèse de Profil d'Application et Formulation de Stratégie Finale.

#### — Synthèse de Profil d'Application :

- En recevant la charge utile de contexte structurée de l'Agent 1, l'agent détermine si la charge utile contient des artefacts d'application explicites ou des références de configuration - appelés *contexte d'application*.
- Si un tel contexte est présent :
  - Un référentiel Git référencé dans la charge utile est cloné en utilisant une interface de contrôle de version intégrée.
  - Une série d'**Analyseurs Centraux** spécifiques au domaine sont ensuite exécutés sur le contenu du référentiel. Ceux-ci peuvent inclure des analyseurs pour les manifestes Kubernetes, les charts Helm, les Dockerfiles, les plans Terraform ou les métadonnées spécifiques à l'application (par exemple, liaisons de ports, variables d'environnement).
  - Le résultat de ce pipeline est une carte consolidée de **Faits d'Application Structurés**, qui décrivent les caractéristiques, dépendances et topologie de l'unité de déploiement.
- Si aucun contexte d'application n'est explicitement présent :
  - Une **Analyse Heuristique** est effectuée sur la transcription de chat utilisateur originale intégrée dans la charge utile.
  - Ce mécanisme de repli utilise la classification basée sur des prompts et l'extraction d'entités pour inférer les objectifs de déploiement probables

- (par exemple, "conteneuriser une API web", "provisionner un nœud GPU" ou "configurer l'auto-scaling pour une charge de travail").
- Les faits extraits sont normalisés dans le même format utilisé par le pipeline d'analyseur, assurant l'uniformité dans la formulation de stratégie en aval.
  - **Formulation de Stratégie Finale :**
    - Avec les faits d'application synthétisés, l'agent transite vers la génération d'une stratégie d'exécution optimale basée sur les conditions environnementales.
    - Il initie une **Analyse d'État d'Infrastructure** en interfaçant avec les couches de virtualisation et d'orchestration pour évaluer :
      - Les capacités de nœuds et la distribution des rôles.
      - Les tendances d'utilisation des ressources (CPU, mémoire, GPU, E/S disque).
      - La disponibilité des fonctionnalités (par exemple, auto-scaling, classes de stockage, règles d'affinité).
    - L'état système dérivé, en conjonction avec les faits d'application, est alimenté dans un **Planificateur Stratégique** alimenté par LLM.
      - Le planificateur génère une stratégie de déploiement qui aligne les objectifs fonctionnels avec les contraintes système et les meilleures pratiques.
      - Par exemple, il pourrait recommander de placer l'application sur des nœuds avec des volumes soutenus par SSD pour les performances, ou d'activer l'autoscaling horizontal de pods basé sur les seuils CPU.
    - La sortie est un **JSON de Plan Stratégique** interprétable par machine, qui encode la logique de déploiement, les mappages de ressources et les politiques cibles.
    - Ce plan est ensuite transmis à l'Agent 3 pour transformation en code d'infrastructure exécutable ou manifestes déclaratifs.

Cet agent incarne le principe de raisonnement stratégique en adaptant l'intention de déploiement aux environnements d'exécution du monde réel. Il équilibre l'intention statique avec le contexte dynamique, permettant des décisions qui sont à la fois orientées objectifs et conscientes du système. La séparation entre l'analyse d'application et le diagnostic d'infrastructure assure la modularité, tandis que l'utilisation des LLM permet la flexibilité dans la gestion des spécifications explicites et inférées.

### Agent 3 : Agent de Construction et de Raffinement (Builder)

Le troisième et dernier composant du pipeline du Système Multi-Agents (SMA) est l'Agent de Construction et de Raffinement. Cet agent reçoit la stratégie de déploiement

de haut niveau de l'agent précédent et la transforme en un artefact d'infrastructure déployable - typiquement un manifeste déclaratif structuré. Il emploie un processus de raffinement itératif pour résoudre les détails de configuration manquants ou ambigus, exploitant le raisonnement LLM et les sources de documentation externes. Cet agent incarne les principes du framework ReAct (Raisonnement + Action), effectuant des cycles de synthèse et validation incrémentaux avant de livrer un plan d'infrastructure complet à l'utilisateur.

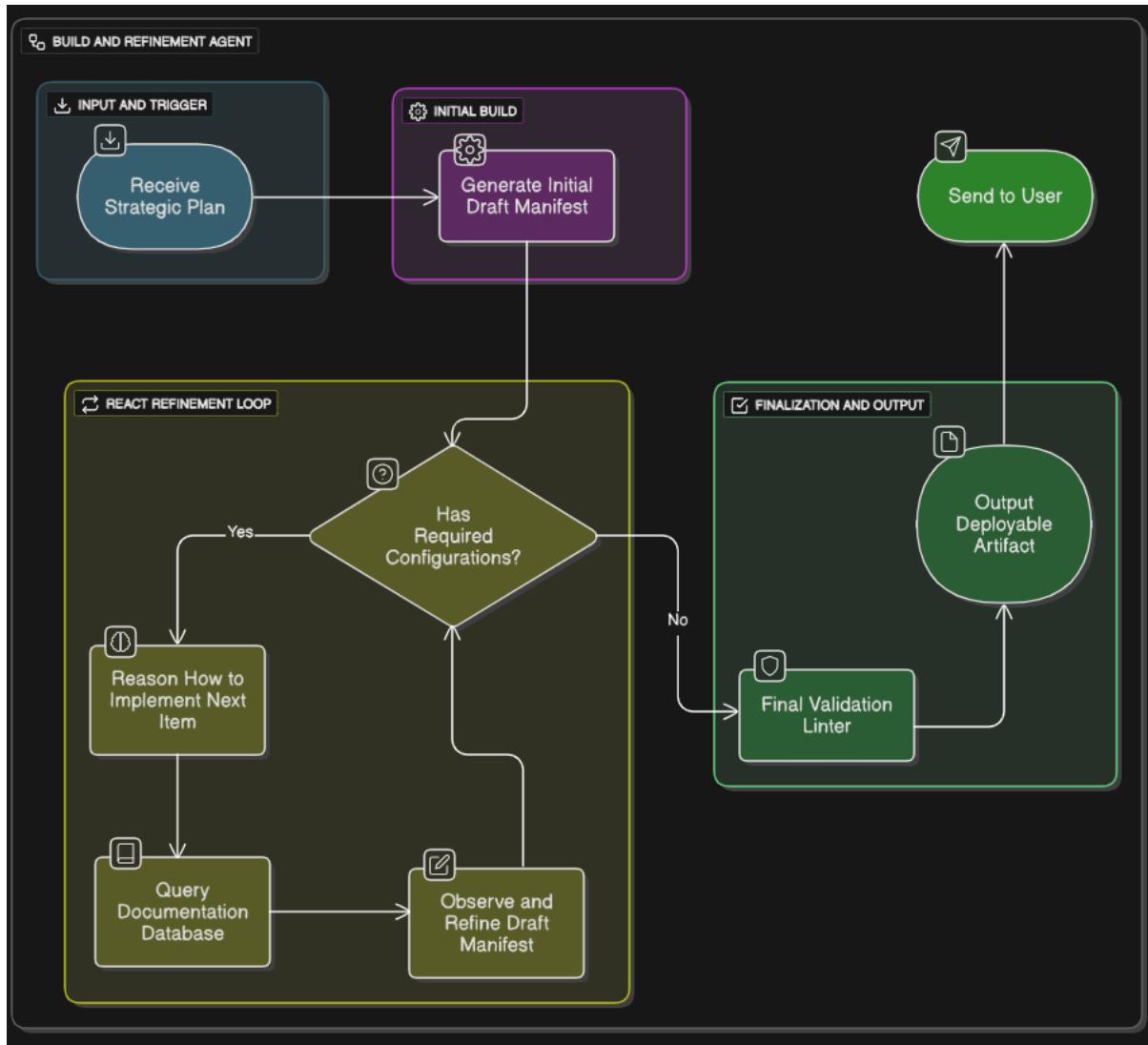


FIGURE 3.9 – Agent 3 - Flux de Travail de Construction et de Raffinement

Comme montré dans la Figure 3.9, l'agent opère à travers quatre étapes principales : Entrée et Déclenchement, Construction Initiale, Boucle de Raffinement ReAct, et Finalisation et Sortie.

#### — Entrée et Déclenchement :

- L'agent commence en recevant un **JSON de Plan Stratégique** produit par l'Agent 2. Ce document encode la structure logique du déploiement prévu,

incluant les exigences de ressources, règles de mise à l'échelle, politiques d'accès et indices de topologie.

- L'entrée sert de spécification déclarative qui guide la génération de manifeste.
- **Construction Initiale :**
  - Un artefact de configuration préliminaire est généré par le module LLM utilisant des modèles de prompts conçus pour la synthèse YAML ou JSON. Cela inclut :
    - Manifestes Kubernetes (par exemple, Deployment, Service, Ingress).
    - Modules Terraform pour le provisionnement d'infrastructure.
    - Charts Helm ou fichiers Docker Compose, selon la plateforme cible.
  - Le **Manifeste de Brouillon Initial** résultant est structurellement solide mais peut contenir des lacunes, champs TODO ou paramètres ambigus.
- **Boucle de Raffinement ReAct :**
  - L'agent entre dans une boucle itérative conçue pour résoudre les exigences de configuration en suspens.
  - Il vérifie d'abord si le manifeste contient toutes les entrées de configuration requises (par exemple, chemins d'images, montages de volumes, limites CPU, secrets).
  - Si incomplet :
    - Le LLM invoque une étape de raisonnement ("Raisonner Comment Implémenter l'Élément Suivant"), exploitant le contexte antérieur et les échaudages de prompts prédéfinis.
    - Si une clarification externe est nécessaire, l'agent interroge une **Base de Données de Documentation** (par exemple, docs Kubernetes, Registre Terraform, Hub Helm) utilisant la recherche sémantique.
    - Basé sur les informations récupérées, le manifeste de brouillon est mis à jour avec des configurations raffinées et sensibles au contexte.
  - Cette boucle se répète jusqu'à ce que la configuration satisfasse les contraintes de complétude ou atteigne une limite d'itération prédéfinie.
- **Finalisation et Sortie :**
  - Une fois le manifeste complet, l'agent invoque un **Linter de Validation Finale** pour :
    - Vérifier la validité du schéma YAML/JSON.
    - Vérifier la compatibilité de plateforme et les références de ressources.
    - Détecter les anti-patterns de configuration ou champs dépréciés.

- Lors d'une validation réussie, l'agent émet un **Artefact Déployable de Sortie** - un fichier prêt pour la production adapté à la soumission directe à un orchestrateur ou pipeline IaC.
- Cet artefact final est ensuite relayé à l'interface utilisateur, complétant le pipeline SMA.

Cet agent exemplifie le principe de synthèse assistée à travers l'auto-correction et l'ancre sémantique. Son intégration de raisonnement, utilisation d'outils et correction réactive assure que même les spécifications stratégiques incomplètes peuvent être étendues itérativement en manifestes de déploiement robustes et exécutables. En intégrant la validation spécifique au domaine et l'accès à la documentation dans la boucle de génération, le système atteint une haute confiance dans la correction et l'applicabilité de la sortie.

### **3.4 Flux de Charges de Travail et de Communication à Travers les Couches**

#### **3.4.1 Synergie Entre les Couches**

L'architecture cloud pilotée par l'IA à cinq couches opère à travers des flux de communication *verticaux* continus qui intègrent les directives de contrôle et les retours d'information à travers tous les niveaux. Cette conception établit un système en boucle fermée : les événements de charge de travail déclenchent des signaux de contrôle descendants pour la gestion des ressources, tandis que les données d'observation se propagent vers le haut pour l'analyse. À travers ces interactions bidirectionnelles, la pile équilibre dynamiquement la demande et la capacité, assurant un comportement réactif et adaptatif. Lorsqu'une nouvelle charge de travail utilisateur ou d'application entre dans le système - par exemple, une poussée de requêtes clients ou le déploiement d'un nouveau service - la couche d'orchestration médie son accommodation en émettant des décisions de planification et de placement. Le plan de contrôle de la Couche d'Orchestration de Conteneurs traduit les intentions de haut niveau (telles que les répliques d'application désirées ou les objectifs de niveau de service) en actions concrètes sur l'infrastructure. En réponse à une charge accrue, l'orchestrateur peut lancer des instances de conteneurs supplémentaires ou ajuster le placement des charges de travail en cours d'exécution pour maintenir les performances.

#### **3.4.2 Flux de Communication**

Ces directives de contrôle s'écoulent vers le bas vers les couches de Virtualisation et Physique sous forme de requêtes d'allocation de ressources. Par exemple, si le planifi-

cateur détermine que les VM actuelles manquent de capacité suffisante pour héberger de nouveaux conteneurs, il peut déclencher le provisionnement d'une nouvelle VM (via l'interface d'automatisation) sur la Couche Physique. Chaque action de ce type traduit la demande abstraite en changements d'infrastructure concrets : les conteneurs sont instantanés sur des VM sélectionnées, de nouvelles VM sont lancées sur des hôtes physiques, et les routes réseau sont configurées en conséquence. En essence, les couches inférieures exécutent les commandes de *déclenchement* qui viennent d'en haut - qu'il s'agisse de déployer un conteneur, de mettre à l'échelle un cluster de VM, ou de reconfigurer un chemin réseau - assurant ainsi que les charges de travail entrantes sont placées et servies par des ressources adéquates en temps réel. Simultanément, chaque couche génère des informations d'état qui remontent sous forme de retour de surveillance. Les serveurs physiques rapportent la santé matérielle au plan de gestion de virtualisation. Les hyperviseurs et VM contribuent des statistiques sur l'utilisation et les performances des ressources virtuelles. Au niveau d'orchestration, les métriques de service à granularité fine et les événements (tels que l'utilisation des ressources de conteneurs et les statuts de cycle de vie des pods) sont continuellement enregistrés. La couche de Surveillance et d'Automatisation agrège et analyse ces flux de télémétrie multi-couches, synthétisant une vue holistique de l'état du système. En corrélant les signaux de bas niveau avec les indicateurs de service de haut niveau, le système de surveillance peut détecter les goulots d'étranglement émergents ou les anomalies. Dans un scénario réactif, les déclencheurs pourraient invoquer des scripts pour mettre à l'échelle un niveau de service (ajoutant des répliques de conteneurs) ou allouer une infrastructure supplémentaire pour soulager le stress détecté. De cette façon, le flux ascendant de télémétrie ferme la boucle en informant les couches supérieures des conditions actuelles et en permettant des actions correctives.

### 3.4.3 Intégration de l'IA Prédictive

Au-delà de ces mesures réactives, la *Couche IA Agentique* introduit un contrôle intelligent et prédictif qui élève l'architecture vers un régime d'auto-optimisation. Cette couche supérieure ingère le flux de télémétrie consolidé et applique des analyses avancées pour anticiper les conditions futures, incarnant le changement industriel vers des opérations cloud autonomes qui met l'accent sur la gestion préventive plutôt que sur les corrections réactives. Toutes les couches restent intégralement connectées à travers des interfaces bien définies qui portent ces échanges de contrôle et de données. Cette intégration verticale assure que chaque strate est à la fois consciente et réactive aux conditions dans les autres : les couches supérieures maintiennent une vue globale du système, tandis que les couches inférieures exécutent des actions localisées sous supervision guidée. Le résultat est un système de contrôle de rétroaction en boucle fermée s'étendant du matériel physique jusqu'à la logique de décision pilotée par l'IA.

## 3.5 Conclusion

Ce chapitre a présenté une conception complète et modulaire pour un système de provisionnement intelligent augmenté par l'IA, structuré à travers cinq couches synergiques. En commençant par une Couche Physique robuste, la conception exploite le matériel standard et le réseau programmable comme substrat pour toutes les abstractions supérieures. La Couche de Virtualisation exploite ensuite les clusters d'hyperviseurs pour découpler les charges de travail des spécificités matérielles, incorporant des mécanismes de stockage partagé et de communication inter-nœuds critiques pour la haute disponibilité, la migration à chaud et la scalabilité. Sur cette fondation, la Couche d'Orchestration de Conteneurs introduit un plan de contrôle distribué, des mécanismes de découverte de services et une planification de charges de travail pilotée par politiques - le tout orchestré sous un cluster haute disponibilité avec élection de leader pour assurer la cohérence dans le provisionnement de volumes et l'état opérationnel. Les chemins de communication clés (tels que les liens inter-nœuds, inter-conteneurs et de service externe) sont sélectivement illustrés pour souligner les relations modulaires sans imposer un couplage architectural inutile. Pour compléter cette architecture de contrôle, la Couche de Surveillance, Gestion et Automatisation ferme la boucle en observant la santé de l'infrastructure, évaluant les seuils opérationnels et déclenchant des flux de travail d'automatisation réactifs ou programmés. Ce circuit de rétroaction fermé, bien que puissant, est intrinsèquement réactif - dépendant de règles prédéfinies ou d'entrées utilisateur. Pour transcender la gestion réactive et embrasser l'optimisation proactive et sensible au contexte, la Couche Agentique introduit un Système Multi-Agents (SMA) construit autour d'agents alimentés par LLM. Cette couche finale transforme fondamentalement l'infrastructure en un système adaptatif capable de comprendre les intentions utilisateur, d'analyser les états actuels, de prédire les charges de travail futures et de synthétiser des artefacts de déploiement exécutables. Ensemble, ces cinq couches représentent un changement de la gestion d'infrastructure statique vers l'orchestration dynamique et intelligente - guidée par les principes de modularité, scalabilité, automatisation et intelligence contextuelle. L'architecture n'est pas seulement un plan pour la conception de système mais aussi un paradigme tourné vers l'avenir pour opérer des environnements cloud modernes sous une complexité et des demandes de performance croissantes.

# Chapitre 4

## Implémentation Pratique

### 4.1 Introduction

Ce chapitre détaille l'implémentation pratique de l'architecture conceptuelle, construisant un véritable environnement de **cloud privé autonome**. Au cœur de celui-ci se trouve un **framework d'automatisation cognitive** piloté par un **Système Multi-Agents (SMA)** coopératif, conçu pour traduire l'intention humaine de haut niveau en **code d'infrastructure** optimisé et exécutable par machine. Cette implémentation sert de **fondement pratique pour les arguments centraux de la thèse**.

Cette implémentation priorise la **haute disponibilité**, l'**exécution automatisée**, et la **mise à l'échelle dynamique**. Pour maintenir cette focalisation, l'implémentation de la sécurité n'est pas incluse dans ce chapitre. Notre innovation principale est plutôt un **Système Multi-Agents (SMA)** personnalisé, composé de plusieurs agents spécialisés, incluant un **Manager**, un **DataCollectionAgent**, un **AnalysesAndStrategyAgent**, et une **ManifestRefinementLoop**. Ce système va au-delà des scripts statiques en travaillant de concert pour interpréter les requêtes des utilisateurs, analyser l'environnement et générer des artefacts de déploiement adaptatifs et conscients du contexte.

Pour présenter ce système complexe, le chapitre est structuré de manière à refléter son **architecture en couches**. Il commence par détailler la **couche fondamentale**, incluant le cluster Proxmox VE et le stockage Ceph. Par la suite, il décrit la **plateforme d'orchestration de conteneurs et d'automatisation**, où Terraform et Ansible sont utilisés pour déployer un cluster k3s en haute disponibilité, complété par MetallLB et Helm. Le chapitre détaille ensuite l'implémentation de la **Couche d'IA Agentique**, la clé de voûte de l'architecture, en décomposant sa structure multi-agents.

Enfin, une série de démonstrations de flux de travail pratiques, incluant un **déploiement de bout en bout**, un **test de tolérance aux pannes**, et un **scénario de provisionnement intelligent**, synthétise ces composants pour valider la résilience et les capacités opérationnelles du système.

### 4.2 Pile Technologique et Justification

La pile technologique pour ce projet a été sélectionnée pour répondre aux objectifs de recherche fondamentaux de construction d'un cloud sur site (on-premises) automatisé, résilient et observable. Les principaux moteurs de décision incluaient une préférence pour les solutions open-source, un engagement envers l'**Infrastructure as Code (IaC)**, et l'utilisation d'outils standards de l'industrie. Cette section présente les technologies choisies et la justification de leur sélection.

### 4.2.1 Sélection des Technologies de Base

L'architecture de la plateforme est composée de plusieurs couches distinctes mais interconnectées, avec des technologies spécifiques choisies pour répondre aux exigences de chaque couche. La pile technologique complète est résumée dans le Tableau 4.1.

L'architecture de la plateforme est composée de plusieurs couches distinctes mais interconnectées, qui sont détaillées visuellement dans le diagramme d'implémentation globale en **Annexe C.1**. Des technologies spécifiques ont été choisies pour répondre aux exigences de chaque couche. L'ensemble de la pile technologique est résumé dans le Tableau 4.1.

TABLE 4.1 – Pile Technologique de Base

Couche Ar-chitecturale	Technologie	Rôle Principal	Description
<b>Virtualisat-ion &amp; Infra-structure</b>	<b>Proxmox Virtual Environnement (VE)</b>	Hyperviseur, gestion de clus-ter, et Haute Disponibilité (HA).	Une plateforme open-source tout-en-un pour la virtualisa-tion d'entreprise, gérant les VMs, les hôtes et la HA.
<b>Stockage</b>	<b>Ceph (RBD &amp; CephFS)</b>	Stockage hyper-convergé, distri-bué et résilient pour les VMs et les sauvegardes.	Un système de stockage dé-finie par logiciel qui fournit un stockage bloc (RBD) et fichier (CephFS) tolérant aux pannes directement depuis les noeuds de l'hyperviseur.
<b>Automatisa-tion &amp; Configura-tion</b>	<b>Terraform &amp; Ansible</b>	IaC pour le provisionne-ment de VMs (Terraform) et la configura-tion système (Ansible).	Une approche à double outil où Terraform crée l'infrastruc-ture de manière déclarative et Ansible configure de manière procédurale les logiciels par-dessus.
<b>Orchestrat-ion de Conteneurs</b>	<b>k3s</b>	Distribution Kubernetes lé-gère et à haute disponibilité.	Une distribution Kubernetes certifiée et légère, optimisée pour les environnements sur site et à ressources limitées, avec une HA simplifiée.

TABLE 4.1 – Pile Technologique de Base

Couche Ar-chitecturale	Technologie	Rôle Principal	Description
<b>Services Réseau</b>	Proxmox SDN, OVS, OPNsense, MetalLB, Kube-vip	Définition de réseau virtuel (SDN) et commutation (OVS), routage/pare-feu (OPNsense), et exposition de services Kubernetes.	Une pile multi-composants fournissant la topologie de réseau virtuel (Proxmox SDN), la structure sous-jacente (OVS), les services réseau de base (OPNsense), et l'équilibrage de charge pour l'API/applications Kubernetes (Kube-vip/MetalLB).
<b>Application &amp; Observabilité</b>	<b>Helm, Prometheus, Grafana</b>	Gestion de paquets Kubernetes (Helm) et surveillance cloud-native.	Helm simplifie le déploiement d'applications, tandis que Prometheus et Grafana fournissent une pile complète pour la collecte et la visualisation de métriques.
<b>Couche d'IA Agentique</b>	Agent Development Kit(ADK) et Google Gemini	Framework pour la construction d'agents (Kit) et modèle d'intelligence principal (Gemini).	Le modèle Gemini fournit les capacités de raisonnement et de planification de base. Les agents, construits à l'aide du kit de développement, exécutent ces plans en interagissant avec les API de la plate-forme.

#### 4.2.2 Justification des Choix

La sélection de chaque technologie de base a été faite après une évaluation assez minutieuse de ses fonctionnalités, de son alignement avec les principes open-source du projet, et de son adéquation à un environnement sur site et hyper-convergé. Les sous-sections suivantes fournissent une justification détaillée pour chaque composant majeur.

### a. Hyperviseur

Pour justifier la sélection de **Proxmox VE** comme hyperviseur fondamental, ses attributs clés sont comparés aux alternatives de premier plan de l'industrie dans le Tableau 4.2. Cette analyse met en évidence l'adéquation unique de la plateforme aux exigences spécifiques du projet.

TABLE 4.2 – Analyse Comparative des Plateformes d’Hyperviseurs

Fonctionnalité / Aspect	Proxmox VE	VMware vSphere	XCP-ng	Microsoft Hyper-V
<b>Modèle de Licence</b>	Open-source (AGPLv3), avec abonnements de support optionnels.	Propriétaire, nécessite une licence par CPU.	Open-source (GPL), avec support professionnel optionnel.	Inclus avec Windows Server ; version autonome gratuite.
<b>Gestion</b>	Interface web intégrée pour toutes les fonctions.	Nécessite vCenter Server pour la plupart des fonctionnalités avancées.	Nécessite Xen Orchestra (interface web) pour une fonctionnalité complète.	Nécessite Hyper-V Manager ou Windows Admin Center.
<b>Fonctionnalités Clés</b>	HA, sauvegarde, Ceph/ZFS, et SDN sont tous intégrés.	Fonctionnalités de pointe vMotion, DRS, et HA.	Capacités robustes de migration à chaud et de HA.	Forte intégration avec l'écosystème Windows.
<b>Écosystème</b>	Ouvert, pas de dépendance vis-à-vis d'un fournisseur, fort soutien communautaire.	Écosystème étendu et mature mais avec une forte dépendance vis-à-vis du fournisseur.	Ouvert, axé sur la communauté, centré sur l'open-source.	Étroitement intégré avec Microsoft Azure et ses services.
<b>Justification pour le Projet</b>	Optimal pour les projets de recherche hyper-convergés et économiques grâce à son ensemble de fonctionnalités intégrées et open-source.	Standard pour les grandes entreprises avec des budgets importants.	Une alternative open-source solide.	Idéal pour les environnements majoritairement basés sur Windows.

Finalement, l'analyse confirme la sélection de **Proxmox VE**. Sa combinaison d'un modèle

open-source économique avec un riche ensemble de fonctionnalités intégrées correspond directement aux exigences du projet pour un environnement hyper-convergé.

## b. Orchestration de Conteneurs

La sélection de **Kubernetes** (via la distribution **k3s**) est contextualisée par l'analyse comparative du Tableau 4.3. Cette comparaison justifie le choix en évaluant ses capacités par rapport à d'autres plateformes d'orchestration de premier plan.

TABLE 4.3 – Analyse Comparative des Plateformes d'Orchestration de Conteneurs

Fonctionnalité / Aspect	Kubernetes (k3s)	Docker Swarm	OpenShift	Nomad
<b>Complexité</b>	Élevée, mais la variante k3s réduit considérablement la surcharge.	Faible, très facile à apprendre et à déployer.	Très élevée ; ajoute des couches de sécurité et de CI/CD à Kubernetes.	Faible à moyenne ; se concentre sur la simplicité.
<b>Écosystème</b>	Vaste et dominant ; le standard plus utilisé de l'industrie.	Limité ; principalement lié à l'écosystème Docker.	Écosystème Red Hat organisé et axé sur l'entreprise.	En croissance ; s'intègre parfaitement avec les outils HashiCorp.
<b>Flexibilité</b>	Extrêmement flexible et extensible via son API et ses CRDs.	Opinionné et moins flexible.	Opinionné pour la sécurité et les flux de travail des développeurs.	Très flexible ; peut orchestrer des applications non conteneurisées.
<b>Force Principale</b>	Communauté, extensibilité et support d'outils inégalés.	Simplicité et rapidité pour les cas d'utilisation de base.	Sécurité de niveau entreprise et outillage pour les développeurs.	Simplicité, flexibilité et support multichargés de travail.
<b>Justification pour le Projet</b>	Essentiel pour accéder à la plus large gamme d'outils cloud-native (Helm, Prometheus, MetalLB) nécessaires pour cette recherche.	Convient aux applications simples sans besoins complexes.	Un PaaS complet, excessif pour la portée de ce projet.	Une alternative solide, mais avec un écosystème plus petit.

Cette analyse valide la sélection de **Kubernetes (k3s)** comme le choix optimal. Son

écosystème dominant était une exigence non négociable pour intégrer les outils cloud-native essentiels nécessaires pour atteindre les objectifs du projet.

### c. Stockage Hyper-Convergé et Système d'Exploitation

L'objectif de créer une véritable **infrastructure hyper-convergée (HCI)** a guidé la décision en matière de stockage. **Ceph** a été choisi car il permet de distribuer le stockage sur les noeuds Proxmox eux-mêmes, évitant ainsi le point de défaillance unique et le goulot d'étranglement de performance d'un périphérique NAS séparé. Son intégration étroite avec Proxmox et sa capacité à fournir à la fois un stockage bloc résilient (**RBD**) pour les VMs et un stockage de fichiers partagé (**CephFS**) pour les sauvegardes en ont fait un choix exceptionnellement polyvalent et efficace.

Pour le système d'exploitation de base des machines virtuelles, **Ubuntu Server LTS** a été sélectionné. Bien qu'il ne s'agisse pas d'un composant architectural de base, c'est un choix d'implémentation critique. Sa sélection a été basée sur son immense soutien communautaire, sa stabilité à long terme et, plus important encore, son support de premier ordre pour **cloud-init**. Cette fonctionnalité était indispensable pour le flux de travail d'automatisation Terraform, permettant la configuration transparente et programmatique des nouvelles VMs au premier démarrage.

### d. Outils d'Automatisation et de Configuration

Un principe fondamental de ce projet était l'automatisation de bout en bout. Une approche puissante à deux outils a été adoptée, tirant parti de **Terraform** et **Ansible** pour leurs forces complémentaires.

**Terraform** a été choisi pour le provisionnement de l'infrastructure. En tant qu'outil **IaC** déclaratif, Terraform excelle dans la gestion du cycle de vie des ressources avec un état défini, telles que les machines virtuelles. Il permet de définir toute l'infrastructure VM dans du code, garantissant que `terraform apply` fera toujours converger le système vers l'état souhaité. Son écosystème de fournisseurs, y compris un fournisseur Proxmox bien supporté, a été un facteur critique dans ce choix.

**Ansible** a été sélectionné pour la gestion de la configuration. Contrairement à Terraform, Ansible est un outil d'automatisation procédural, ce qui le rend idéal pour exécuter des tâches ordonnées comme l'installation de logiciels, la configuration de fichiers système et l'orchestration de flux de déploiement complexes. Son architecture sans agent, qui communique via SSH standard, simplifie l'installation et réduit la surface d'attaque par rapport aux alternatives basées sur des agents comme **Puppet** ou **Chef**. Pour ce projet, Ansible a été indispensable pour transformer les VMs vierges provisionnées par Terraform en un cluster k3s entièrement configuré et à haute disponibilité.

### e. Services Réseau et Applicatifs

Une approche flexible et définie par logiciel pour le réseau était essentielle. **OPNsense** a été déployé en tant qu’appliance virtuelle pour servir de routeur central, de pare-feu et de serveur DHCP/DNS pour la plateforme. Le déployer en tant que VM offrait une plus grande flexibilité qu’une appliance physique et proposait une alternative plus riche en fonctionnalités et plus activement développée que d’autres pare-feux open-source comme **pfSense**.

Au sein du cluster Kubernetes, deux lacunes critiques en matière de réseau devaient être comblées pour un déploiement sur site :

1. **HA du Plan de Contrôle** : **Kube-vip** a été utilisé pour fournir une adresse IP virtuelle (VIP) stable pour le **serveur d’API Kubernetes**. Il utilise ARP pour annoncer la VIP sur le réseau et s’assure qu’elle est toujours détenue par un nœud maître sain, ce qui est essentiel pour maintenir l’accès au plan de contrôle en cas de défaillance d’un nœud.
2. **Exposition des Services Applicatifs** : **MetalLB** a été implémenté pour fournir le type de service **LoadBalancer**, qui n’est pas nativement disponible dans les clusters sur site. MetalLB écoute les objets Service de ce type et leur attribue une adresse IP à partir d’un pool préconfiguré, rendant les applications conteneurisées accessibles depuis le réseau externe.

### f. Gestion des Applications et Observabilité

Pour gérer le cycle de vie des applications sur Kubernetes, **Helm** a été choisi. En tant que gestionnaire de paquets standard pour Kubernetes, Helm permet de packager des applications complexes à plusieurs composants dans des charts réutilisables, simplifiant le déploiement, les mises à jour et la configuration. Son moteur de templating est bien plus puissant que l’utilisation de fichiers YAML bruts pour gérer des versions d’applications complexes.

Pour l’observabilité, la pile standard de l’industrie composée de **Prometheus** et **Grafana** a été sélectionnée. Le modèle pull-based de Prometheus et ses puissantes capacités de découverte de services s’intègrent parfaitement à Kubernetes, lui permettant de trouver et de collecter automatiquement les métriques des nouveaux pods et services. Grafana fournit une interface puissante et flexible pour visualiser ces données. Cette combinaison a été choisie de préférence à des outils de surveillance plus anciens comme **Nagios** ou **Zabbix** car elle est spécialement conçue pour la nature dynamique et éphémère des environnements cloud-native.

### j. La Couche d'IA Agentique

La couche d'IA Agentique, clé de voûte de l'architecture, a été implémentée en utilisant l'écosystème de Google. Ce choix a été motivé par l'engagement fort de Google envers les communautés de l'IA et de l'open-source, ce qui offre une flexibilité significative et est soutenu par un support étendu. La couche exploite le **Google Agent Development Kit (ADK)** pour construire les agents et le **modèle Google Gemini** comme intelligence centrale. Cette combinaison permet au système de traduire des commandes en langage naturel en actions concrètes en faisant appel de manière sécurisée aux outils sous-jacents de la plateforme.

En fin de compte, cette pile technologique forme une **plateforme cohésive, open-source et hautement automatisée**. Chaque composant a été délibérément choisi pour sa stabilité et son adéquation à un environnement **sur site** et **hyper-convergé**, garantissant que le système final répond directement aux objectifs de recherche de **résilience, de scalabilité et de provisionnement intelligent**.

## 4.3 Couche Fondamentale : Infrastructure Physique et Virtualisation

Le fondement de l'ensemble de l'environnement cloud autonome repose sur une couche physique et de virtualisation robuste et bien configurée. Cette section détaille la sélection du matériel, la configuration du réseau et l'installation de l'hyperviseur **Proxmox Virtual Environment (VE)**, qui sert de socle à toutes les couches ultérieures.

### 4.3.1 Configuration Matérielle et Réseau

Le fondement de tout cluster stable, en particulier celui conçu pour une haute disponibilité et des performances prévisibles, est l' **homogénéité matérielle**. En utilisant des composants de serveur identiques sur tous les noeuds, nous assurons des performances constantes, une gestion simplifiée et des opérations de basculement fiables pour des fonctionnalités comme la migration à chaud. L'infrastructure physique pour cette recherche se compose de trois serveurs identiques interconnectés via un commutateur réseau physique. Les spécifications de chaque serveur sont détaillées dans le Tableau 4.4, et leurs configurations réseau correspondantes sont décrites dans le Tableau 4.5.

Une fois les fondations physiques et réseau établies, l'étape suivante a été l'installation de l'hyperviseur Proxmox VE sur chaque noeud.

TABLE 4.4 – Spécifications des Serveurs Physiques

Composant	Spécification
CPU	12th Gen Intel Core i7-12700 (1 Socket, 12 Coeurs, 20 Threads)
Mémoire (RAM)	32 Go
Stockage	1 To NVMe SSD
Interface Réseau	1x Carte d'interface réseau (NIC) 1GbE
Connectivité	Connecté via un commutateur physique partagé 1GbE

TABLE 4.5 – Configuration Réseau des Nœuds

Nom d'hôte (FQDN)	Adresse IP	Masque de sous-réseau	Passerelle	Serveur DNS
pmox01.lab.local	172.25.5.201	255.255.255.0	172.25.5.1	8.8.8.8
pmox02.lab.local	172.25.5.202	255.255.255.0	172.25.5.1	8.8.8.8
pmox03.lab.local	172.25.5.203	255.255.255.0	172.25.5.1	8.8.8.8

### 4.3.2 Installation de Proxmox VE

Le processus d'installation de **Proxmox VE (version 8.3)** a été lancé sur chacun des trois serveurs. Cela a commencé par la création d'une clé USB amorçable à l'aide de l'image ISO officielle de Proxmox VE et de l'utilitaire **Rufus**. Au démarrage à partir du support d'installation, l'option *Install Proxmox VE (Graphical)* a été sélectionnée dans le menu de démarrage principal.

L'assistant d'installation a progressé à travers le Contrat de Licence Utilisateur Final (EULA). Une étape critique de ce processus a été le partitionnement stratégique du stockage. En accédant au menu *Options* sur l'écran '**Target Harddisk**' (Figure 4.1), le paramètre **hdszie** a été explicitement défini à 100 Go. Cette action a alloué une partition de 100 Go pour le système d'exploitation Proxmox et son stockage local par défaut, laissant intentionnellement les ~850 Go d'espace disque restants non alloués. Cet espace réservé est essentiel pour la création ultérieure du cluster de stockage distribué **Ceph**, qui est détaillé dans la Section 4.3.5.



FIGURE 4.1 – Options du disque dur Proxmox pour le dimensionnement des partitions

Par la suite, l'écran de configuration du réseau de gestion a été utilisé pour attribuer l'adresse IP statique, la passerelle, le serveur DNS et le **Nom de Domaine Entièrement Qualifié (FQDN)** pour chaque nœud, comme spécifié dans le Tableau 4.5. Tous les paramètres ont été confirmés sur l'écran de résumé, et l'installation a été lancée. À des fins de réplicabilité dans ce contexte académique, le mot de passe administrateur pour tous les nœuds a été défini sur `adminproxmox`.

Après une installation réussie, le système a redémarré automatiquement. La console a ensuite affiché l'URL pour accéder à l'interface web via HTTPS sur le port 8006. L'accès à cette URL et la connexion avec l'utilisateur root et le mot de passe préconfiguré ont confirmé l'installation réussie du nœud. Le tableau de bord web de Proxmox VE (Figure 4.2) affichait le nœud unique et opérationnel, maintenant prêt pour la phase suivante de configuration du cluster.

## Chapitre 4. Implémentation Pratique

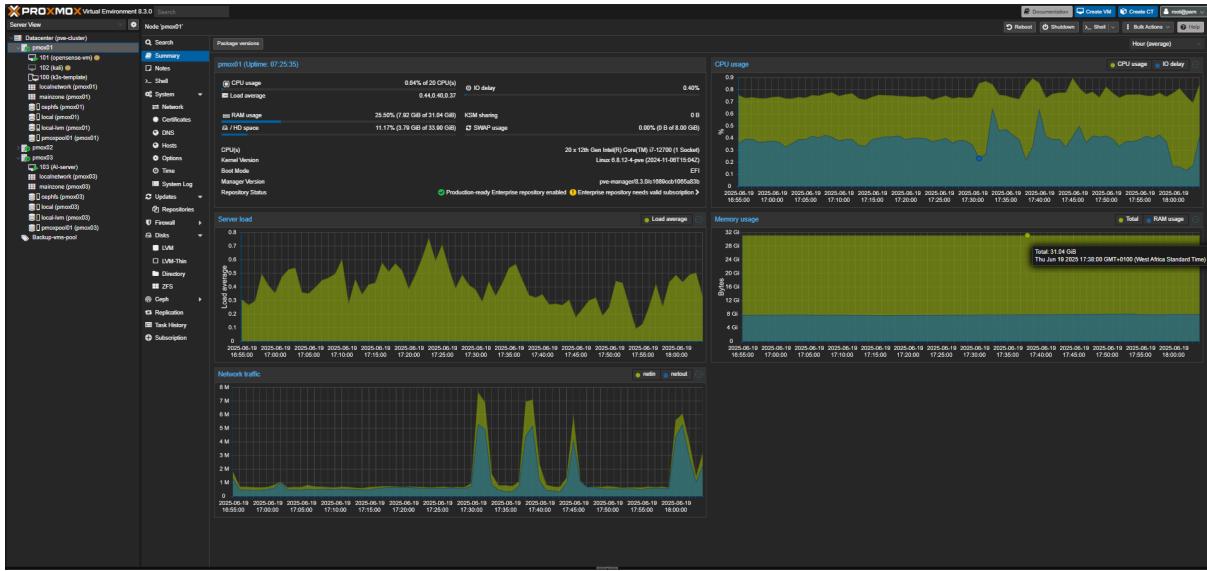


FIGURE 4.2 – Exemple de tableau de bord de l’interface web Proxmox après l’installation

### 4.3.3 Configuration Réseau avec Open vSwitch

Pour activer des capacités avancées de **Réseautage Défini par Logiciel (SDN)** au sein de Proxmox VE, le pont Linux par défaut sur chaque nœud a été remplacé par **Open vSwitch (OVS)**. Cette transition critique a été automatisée pour la cohérence et la répétabilité à travers le cluster à l'aide d'un **playbook Ansible**.

Le playbook a orchestré la reconfiguration du réseau en s'assurant d'abord que le paquet `openvswitch-switch` était installé, puis en déployant un nouveau fichier `/etc/network/interfaces`. Cette nouvelle configuration établit une nouvelle topologie réseau en créant un pont OVS nommé `vmbr0`. L'interface réseau physique (`enp1s0`) est attachée à ce pont en tant que port, et l'adresse IP statique du nœud est assignée directement au pont lui-même. La structure logique du réseau résultante est illustrée à la Figure 4.3.

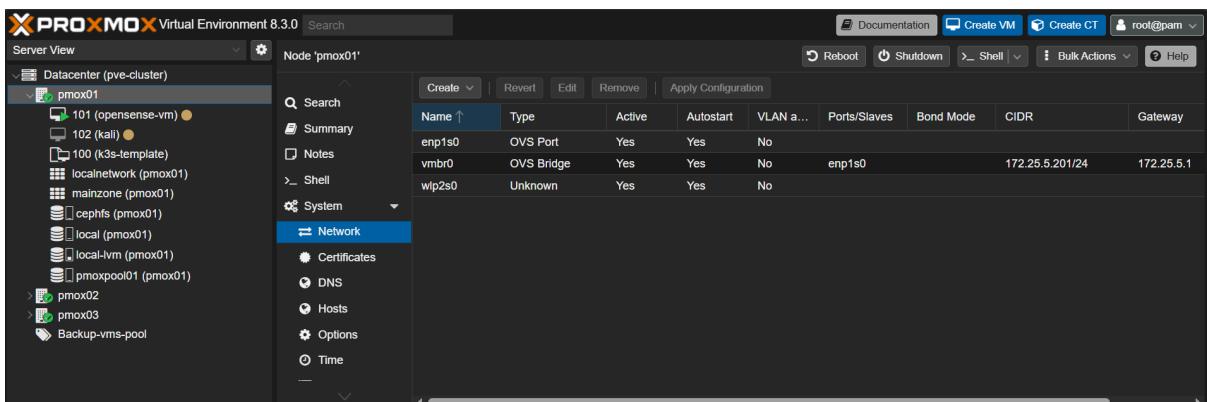


FIGURE 4.3 – Aperçu de la configuration réseau avec le pont OVS

Pour gérer les adresses IP uniques de chaque nœud (172.25.5.201, 172.25.5.202 et

172.25.5.203), l'automatisation Ansible a utilisé un mécanisme de templating. Cela garantit que, bien que la topologie réseau soit identique à travers le cluster, chaque noeud conserve son adresse IP de gestion correcte. Avec OVS maintenant configuré sur tous les noeuds, le système est prêt pour la création du cluster Proxmox et l'implémentation de zones SDN avancées.

### 4.3.4 Configuration du Cluster d'Hyperviseurs Proxmox VE

Une fois les noeuds Proxmox VE individuels installés, ils ont été unifiés en une seule entité gérée de manière centralisée. C'est une étape critique pour permettre la **haute disponibilité** et tirer parti du stockage distribué. Le processus a commencé par la création d'un nouveau cluster nommé `pve-cluster` sur le noeud principal, `pmax01`. Cette action a généré des informations d'identification sécurisées qui ont ensuite été utilisées sur les noeuds `pmax02` et `pmax03` pour les intégrer au cluster.

La formation réussie du cluster aboutit à un environnement cohésif à trois noeuds géré depuis une seule interface web. Comme le montre la Figure 4.4, la vue récapitulative du datacenter confirme maintenant que les trois noeuds sont en ligne et font partie d'un cluster avec quorum, offrant un aperçu consolidé de toutes les ressources.

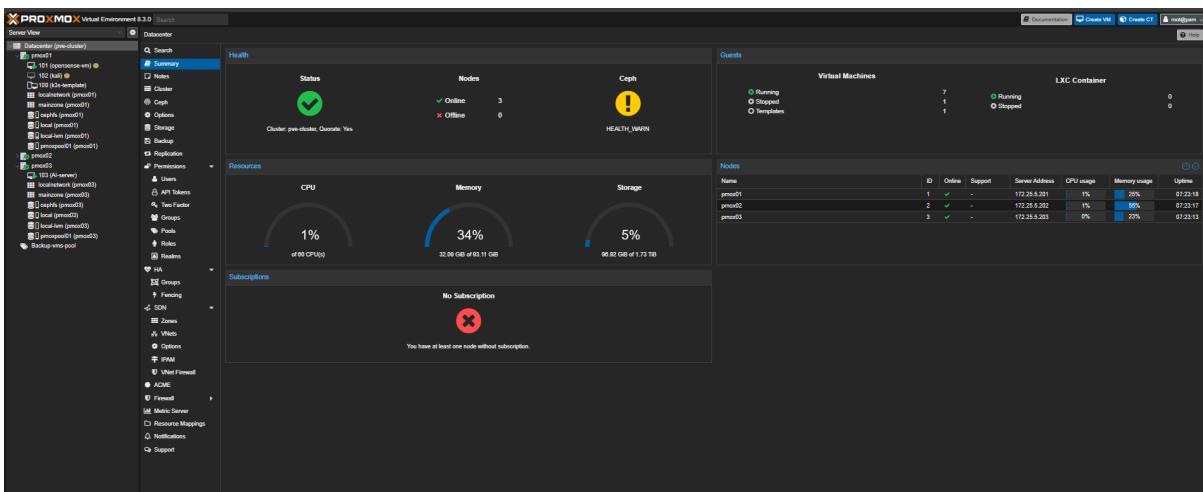


FIGURE 4.4 – Résumé du Datacenter Proxmox après la formation du cluster

Cette structure unifiée est la condition préalable fondamentale pour déployer le système de stockage résilient Ceph à l'étape suivante.

### 4.3.5 Stockage Défini par Logiciel avec Ceph

Une fois le cluster Proxmox établi, l'étape critique suivante a été d'implémenter une couche de stockage hyper-convergée en utilisant **Ceph**. Cela fournit une plateforme de

stockage résiliente, évolutive et unifiée. Le processus a commencé par la préparation des disques physiques sur chaque noeud, suivie de l'installation et de la configuration du cluster Ceph lui-même.

Comme détaillé dans l'installation de Proxmox (Section X.3.2), le paramètre `hdsize` a été utilisé pour réserver la majorité de l'espace disque sur chaque noeud. Avant que cet espace puisse être utilisé par Ceph, il devait être formellement partitionné. La procédure détaillée en ligne de commande pour cette tâche est fournie en [Annexe A.1](#).

- Le résultat de cette opération est une nouvelle grande partition (`/dev/sda4`) dédiée à Ceph, dont la présence a été confirmée à l'aide de la commande `lsblk`, comme le montre la Liste 4.5.

```
sda      8:0    0  953.9G  0 disk
| -sda1   8:1    0   1007K  0 part
| -sda2   8:2    0     1G  0 part /boot/efi
| -sda3   8:3    0   99G  0 part
| | -pve-swap 252:0  0     8G  0 lvm  [SWAP]
| | -pve-root 252:1  0   34.7G 0 lvm  /
...
`-sda4   8:5    0  853.9G  0 part
```

FIGURE 4.5 – Disposition finale du disque avec la nouvelle partition pour Ceph

Cette étape de partitionnement manuel est une condition préalable cruciale pour la création de **Démons de Stockage d'Objets (OSD) Ceph** sur le même disque physique que le système d'exploitation Proxmox.

Avec les disques préparés, le cluster Ceph a été configuré directement depuis l'interface web de Proxmox. Le processus impliquait l'établissement d'un quorum de trois noeuds **Monitor (MON)** pour la haute disponibilité, la création d'un **Démon de Stockage d'Objets (OSD)** sur la partition préparée de chaque hôte, puis le provisionnement de deux types de stockage distincts : un **pool de stockage bloc répliqué** (`poxpool01`) pour les disques de VM résilients et un **système de fichiers CephFS** (`cephfs`) pour le stockage partagé. La procédure détaillée, étape par étape, pour cette configuration, y compris la création des moniteurs, des OSDs et des pools de stockage, est documentée en [Annexe A.3](#).

L'aboutissement de ces étapes est une plateforme de stockage unifiée et hyper-convergée, intégrée directement au sein du cluster Proxmox. La santé globale et l'état final du cluster sont surveillés via le tableau de bord principal de Ceph, comme le montre la Figure 4.6. Cette vue récapitulative confirme un statut `HEALTH_OK`, indiquant que tous les compo-

sants sont en ligne et que le système fonctionne comme prévu, remplissant ainsi l'objectif d'une fondation de stockage stable et résiliente.

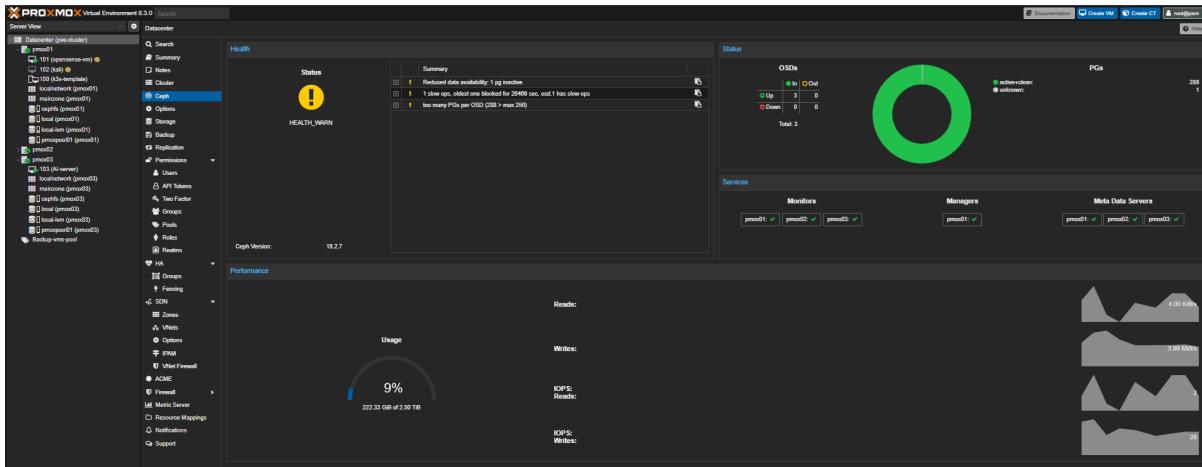


FIGURE 4.6 – Le tableau de bord du cluster Ceph

### 4.3.6 Réseautage Défini par Logiciel (SDN) et l’Appliance Virtuelle OPNsense

La fondation de stockage étant établie, l'attention se porte maintenant sur la structure du réseau virtuel. Cette section détaille la configuration du **Réseautage Défini par Logiciel (SDN)** et de l'appliance virtuelle **OPNsense** qui fournit les services de routage et DHCP.

Bien que le réseau physique fournisse la connectivité pour les hôtes Proxmox, il est insuffisant pour provisionner un environnement virtuel vaste, isolé et multi-locataire. Pour résoudre ce problème, une approche hybride a été mise en œuvre, combinant les capacités natives de **Réseautage Défini par Logiciel (SDN)** de Proxmox VE avec une appliance virtuelle dédiée pour les services réseau. Cette stratégie tire parti des forces des deux composants : Proxmox SDN est utilisé pour définir la structure du réseau virtuel et appliquer la segmentation du réseau au niveau de l'hyperviseur, tandis qu'une appliance virtuelle OPNsense fournit des services essentiels de couche 3 tels que le routage, le DHCP et le DNS. Cela crée un réseau virtuel flexible et puissant, indépendant de l'infrastructure physique sous-jacente.

#### a. Définition de la Structure du Réseau Virtuel dans Proxmox

L'étape initiale a été de configurer les composants SDN au sein du Datacenter Proxmox pour créer l'espace réseau logique.

- 1. Crédit de la Zone SDN :** Une zone VLAN nommée `mainzone` a été créée, comme le montre la Figure 4.7. Cette zone utilise le plugin `vlan`, qui demande à

## Chapitre 4. Implémentation Pratique

Proxmox de marquer le trafic pour ce réseau, assurant ainsi son isolation des autres réseaux.

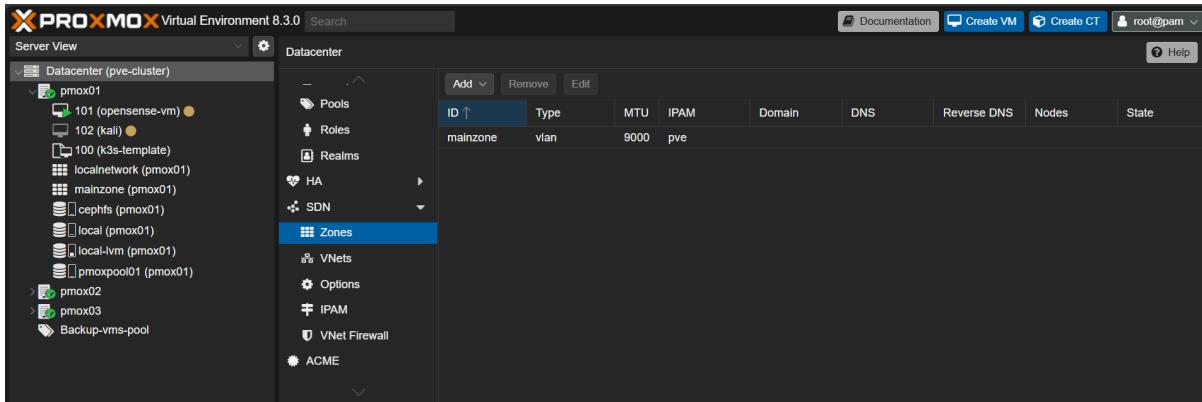


FIGURE 4.7 – Configuration de la zone SDN `mainzone`

2. **Création du VNet et du Sous-réseau :** Un **Réseau Virtuel (VNet)** nommé `mainvnet` a ensuite été créé et associé à la `mainzone`. Au sein de ce VNet, un sous-réseau a été défini pour le **Réseau Local (LAN)** interne : `192.168.0.0/16`. Cela fournit un grand espace d'adressage privé pour toutes les machines virtuelles et les pods Kubernetes ultérieurs. La passerelle pour ce sous-réseau a été désignée comme `192.168.0.1`, qui est l'adresse IP qui sera attribuée à l'interface interne de l'apppliance OPNsense (Figure 4.8).

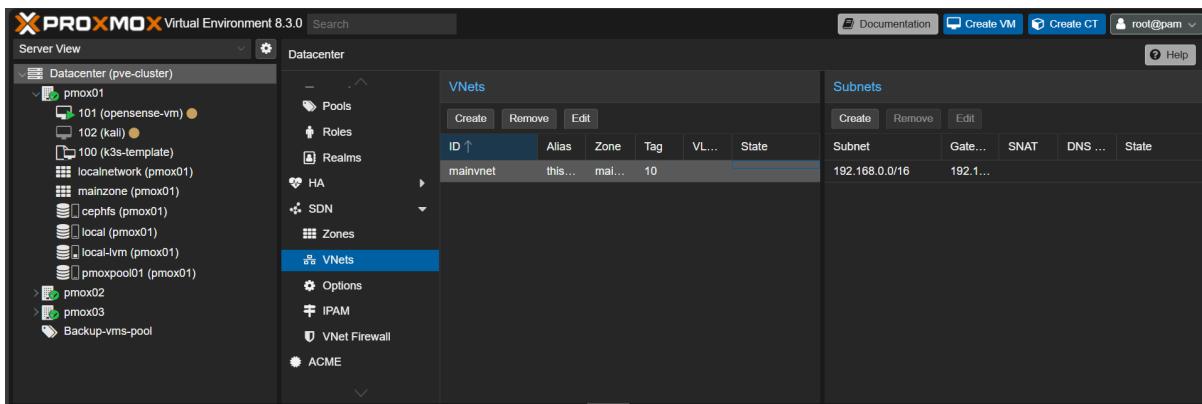


FIGURE 4.8 – Définition du `mainvnet` et de son sous-réseau associé `192.168.0.0/16`.

Avec ces composants configurés, Proxmox est maintenant conscient du réseau virtuel, et ce VNet peut être attaché aux machines virtuelles.

### b. OPNsense comme Routeur Virtuel et Fournisseur de Services

Avec la structure du réseau virtuel définie dans Proxmox SDN, une machine virtuelle OPNsense (ID 101) a été déployée pour agir comme routeur central, pare-feu et fournisseur

## Chapitre 4. Implémentation Pratique

de services pour le `mainvnet`. L’appliance OPNsense est essentielle pour relier le réseau virtuel isolé au monde extérieur et fournir des services réseau essentiels.

L’apppliance a été provisionnée avec 4 Go de RAM et 2 coeurs de CPU. Comme le montre la configuration matérielle de Proxmox (Figure 4.9), la VM a été équipée de deux interfaces réseau virtuelles distinctes :

- **net0** : Connectée au `mainvnet`, désignée pour servir d’interface LAN interne.
- **net1** : Connectée au pont physique `vmbr0`, désignée pour servir d’interface WAN externe.

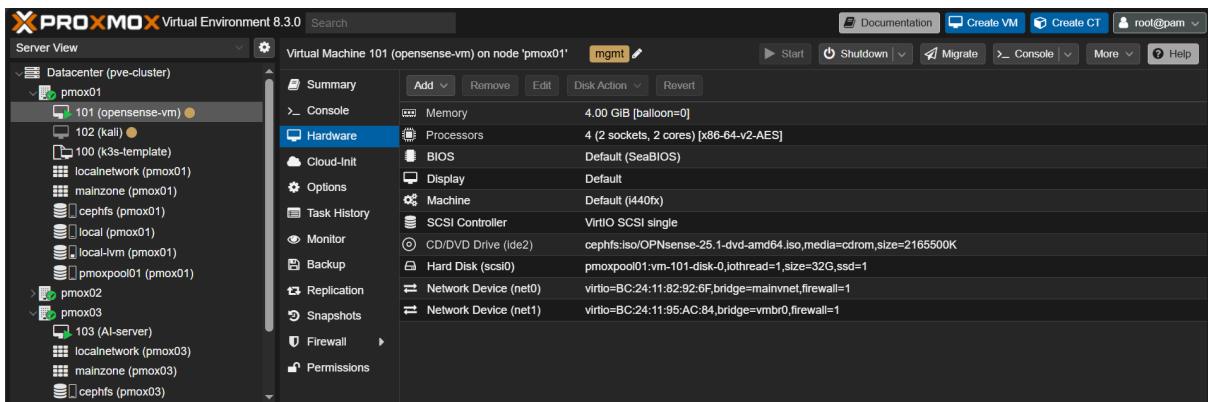


FIGURE 4.9 – Configuration matérielle Proxmox pour la VM OPNsense

Cette configuration à double interface établit un flux de trafic réseau clair et multicouches, comme illustré à la Figure 4.10. Tout le trafic provenant des machines virtuelles sur le LAN interne est acheminé via l’appliance OPNsense, qui effectue la **Traduction d’Adresse Réseau (NAT)**. Cela permet à toutes les VMs sur le réseau privé `192.168.0.0/16` de partager la seule adresse IP de l’interface WAN d’OPNsense pour l’accès sortant à Internet, tout en restant isolées du réseau physique.

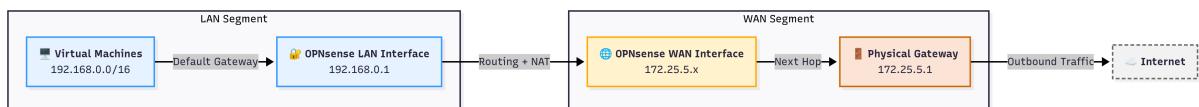


FIGURE 4.10 – Flux logique du trafic réseau

### c. Installation et Configuration Initiale d’OPNsense

Pour fournir des **services réseau de couche 3** essentiels tels que le routage, le DHCP et le DNS pour l’environnement virtualisé, une **appliance virtuelle OPNsense** a été déployée. L’apppliance a été installée depuis la console Proxmox, avec un accès initial à l’interface graphique web astucieusement établi via une VM temporaire sur le même `mainvnet` isolé. Les détails de l’installation sont documentés en **Annexe B.1**.

Pour les besoins de cette recherche, la configuration a été axée sur la **fonctionnalité**

**de base.** Les fonctionnalités avancées comme les protocoles de routage dynamique et les VPN ont été intentionnellement omises pour maintenir la simplicité, et le **pare-feu a été désactivé** pour assurer un **flux de trafic sans restriction** au sein de l'environnement de laboratoire de confiance. La configuration finale comprenait l'attribution d'IP statiques aux interfaces LAN et WAN, la définition de la passerelle par défaut et l'activation d'un serveur DHCP pour le réseau interne, comme indiqué en [Annexe B.1](#).

Le résultat réussi de ce processus est un **routeur virtuel entièrement opérationnel**, résumé par le tableau de bord principal montré à la Figure 4.11. Cette vue unique confirme l'état opérationnel du système, y compris la **passerelle WAN active (172.25.5.1)**, l'utilisation des ressources et les statistiques de trafic en direct, validant que l'apppliance **route correctement le trafic** entre les interfaces LAN interne et WAN externe.

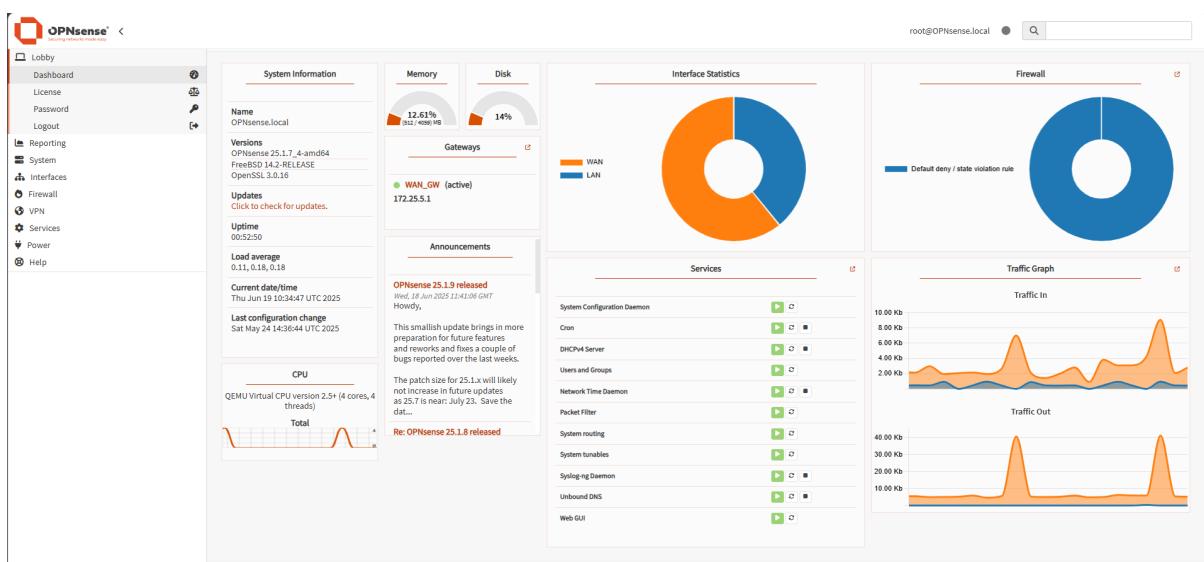


FIGURE 4.11 – Le tableau de bord d’OPNsense montrant l’état opérationnel final

Ceci a établi un **réseau virtuel robuste**, où OPNsense fournit un **routage et un DHCP critiques**, préparant l’environnement pour toutes les charges de travail ultérieures de machines virtuelles et de conteneurs.

### 4.3.7 Configuration de la Haute Disponibilité et de la Résilience du Système

Avec les couches de calcul, de stockage et de réseau fondamentales établies, l’accent est mis sur la garantie de la résilience opérationnelle contre les pannes matérielles et la perte de données. Ceci a été réalisé en mettant en œuvre une stratégie à deux volets au sein du cluster Proxmox VE : le basculement des services en temps réel à l'aide du gestionnaire de **Haute Disponibilité (HA)** et une protection robuste des données grâce à des sauvegardes automatisées et planifiées.

### a. Haute Disponibilité pour la Continuité de Service

Pour protéger les services critiques contre les temps d'arrêt imprévus des hôtes, la fonctionnalité **Proxmox HA** a été configurée. Ce système détecte automatiquement un nœud défaillant et redémarre ses machines virtuelles désignées sur d'autres noeuds disponibles dans le cluster.

L'implémentation a impliqué deux étapes clés :

- Création d'un Groupe HA** : Un groupe HA nommé `mainHA` a été créé pour définir l'ensemble des nœuds responsables de l'exécution des services à haute disponibilité. Comme le montre la Figure 4.12, ce groupe a été configuré pour inclure les trois nœuds du cluster (`pmax01`, `pmax02`, `pmax03`), assurant une capacité de basculement maximale. L'option `nofailback` a été activée, ce qui signifie que les VMs ne retourneront pas automatiquement à leur hôte d'origine une fois qu'il sera rétabli. Cela aide à prévenir les migrations inutiles et les perturbations de service potentielles causées par le déplacement de charges de travail stables vers leur nœud d'origine.

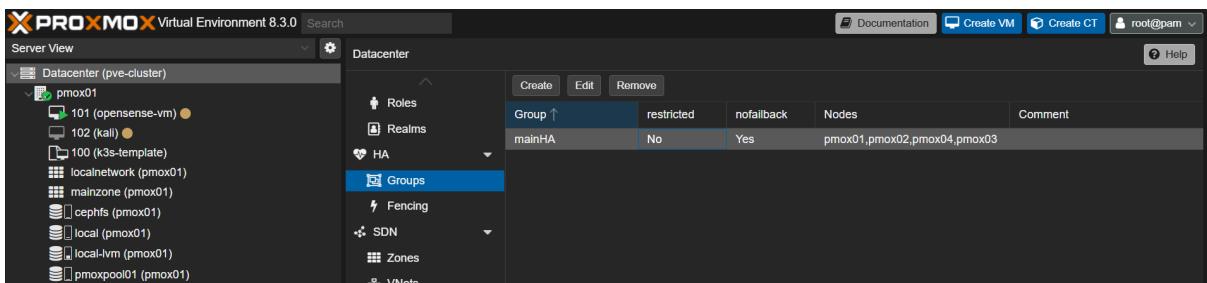


FIGURE 4.12 – La configuration du groupe `mainHA`

- Assignment des Ressources** : Le groupe `mainHA` sert de conteneur de politiques pour tous les services critiques. Initialement, l'apppliance OPNsense a été ajoutée à ce groupe pour assurer la résilience du réseau. Au fur et à mesure que les composants ultérieurs comme le serveur IA et les noeuds Kubernetes ont été provisionnés (en utilisant les méthodes automatisées détaillées dans la Section 4.4), ils ont également été assignés par programme au groupe `mainHA`. La Figure 4.13 illustre l'état final, entièrement peuplé, du panneau des ressources HA, avec tous les services critiques sous gestion. Cette configuration garantit qu'en cas de défaillance d'un hôte, le système tentera automatiquement de redémarrer toute VM gérée sur un nœud survivant.

## Chapitre 4. Implémentation Pratique

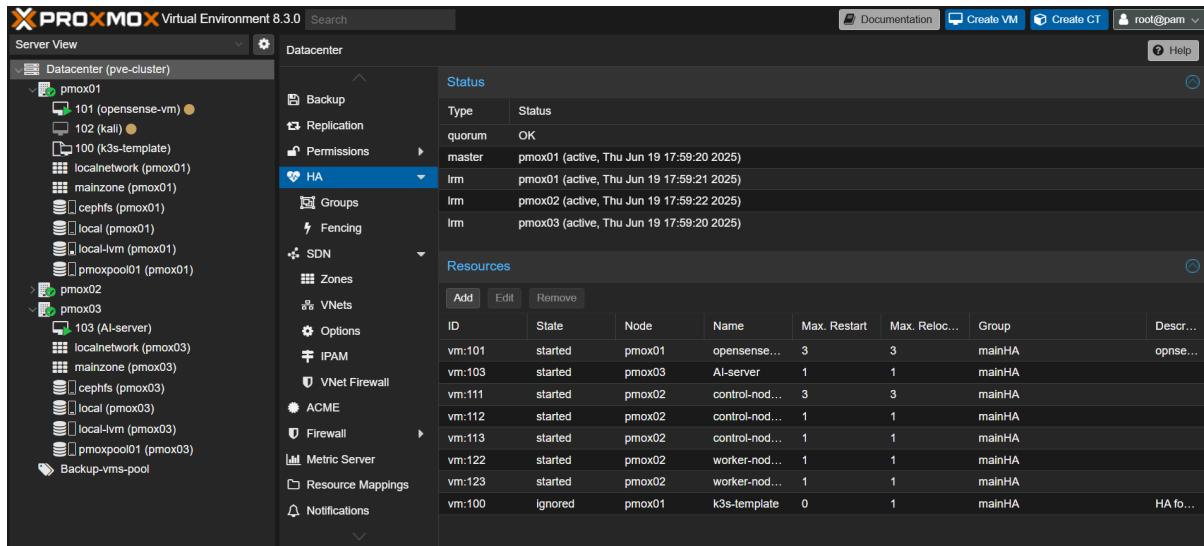


FIGURE 4.13 – L'écran d'état du gestionnaire HA

### b. Sauvegardes Automatisées pour la Protection des Données

Alors que la Haute Disponibilité protège contre la perte de service immédiate, les sauvegardes automatisées sont essentielles pour la reprise après sinistre et la protection contre la corruption des données. Le fondement de la stratégie de sauvegarde est un pool de ressources nommé **Backup-vms-pool**, créé spécifiquement pour regrouper les machines virtuelles les plus critiques, comme le montre la Figure 4.14.

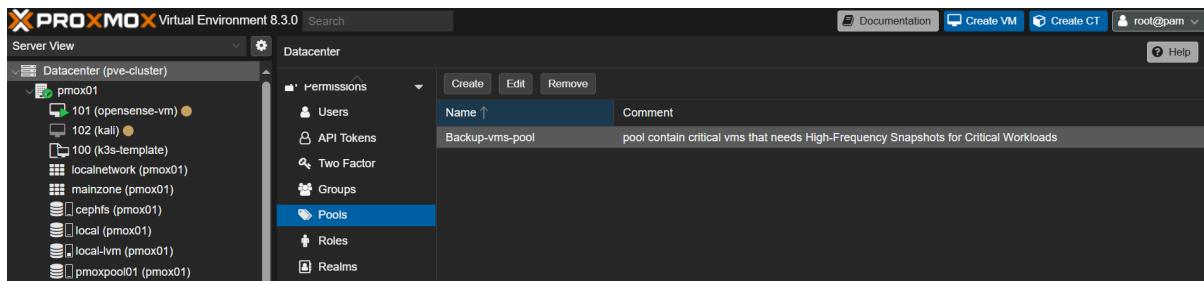


FIGURE 4.14 – Création du Backup-vms-pool

Ce pool est peuplé automatiquement pendant la phase de provisionnement de l'infrastructure, car les VMs Kubernetes et du serveur IA sont créées par Terraform (détalé dans la Section 4.4), elles sont ajoutées par programme en tant que membres. L'état final et peuplé du pool est montré pour référence à la Figure 4.15.

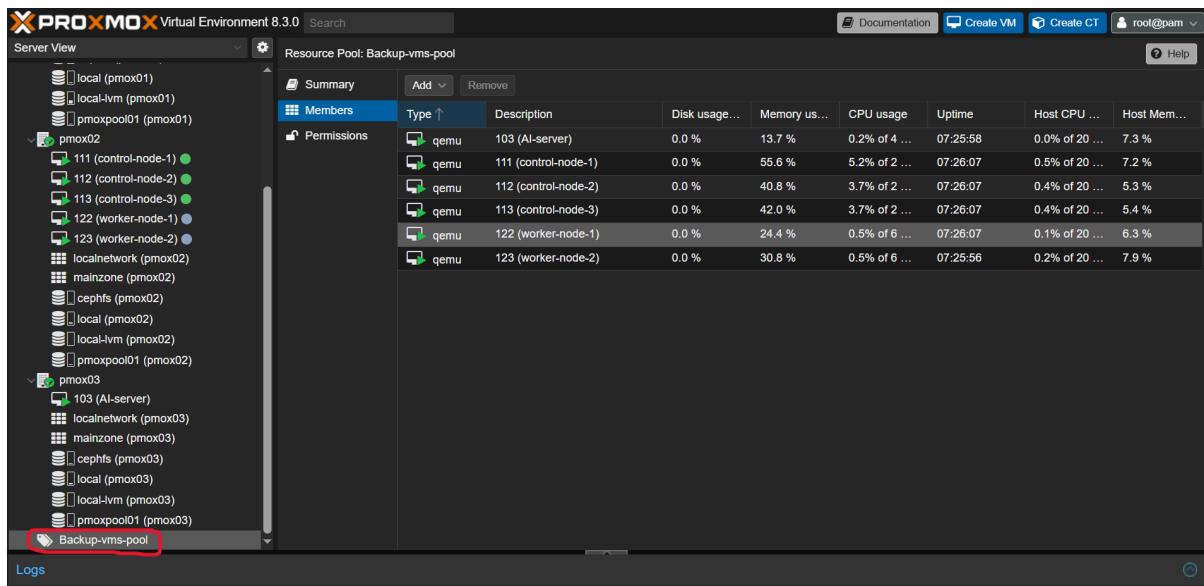


FIGURE 4.15 – Le Backup-vms-pool entièrement peuplé

Avec le **Backup-vms-pool** établi comme cible logique, une stratégie de sauvegarde à deux niveaux a été mise en œuvre pour répondre à différents besoins de récupération, en tirant parti du backend de stockage résilient cephfs pour toutes les données de sauvegarde :

### 1. Instantanés à Haute Fréquence pour les Charges de Travail Critiques :

Pour assurer un faible **Objectif de Point de Reprise (RPO)** pour les services les plus dynamiques, une tâche de sauvegarde a été créée pour cibler le **Backup-vms-pool**. Cette tâche s'exécute toutes les 30 minutes en utilisant le mode *Snapshot*, créant des sauvegardes en direct à un instant T avec un impact minimal sur les performances.

### 2. Sauvegardes Complètes Quotidiennes pour la Reprise après Sinistre :

Pour garantir une cohérence absolue des données pour l'ensemble de la plateforme, une deuxième tâche de sauvegarde plus complète a été configurée. Cette tâche cible toutes les machines virtuelles, s'exécute quotidiennement pendant une fenêtre de faible activité (21 :00), et utilise le mode *Stop*, qui assure un état parfaitement cohérent, idéal pour une reprise après sinistre complète.

La mise en œuvre de cette stratégie complète à deux niveaux est résumée à la Figure 4.16, qui montre les deux tâches de sauvegarde configurées dans la vue Datacenter de Proxmox.

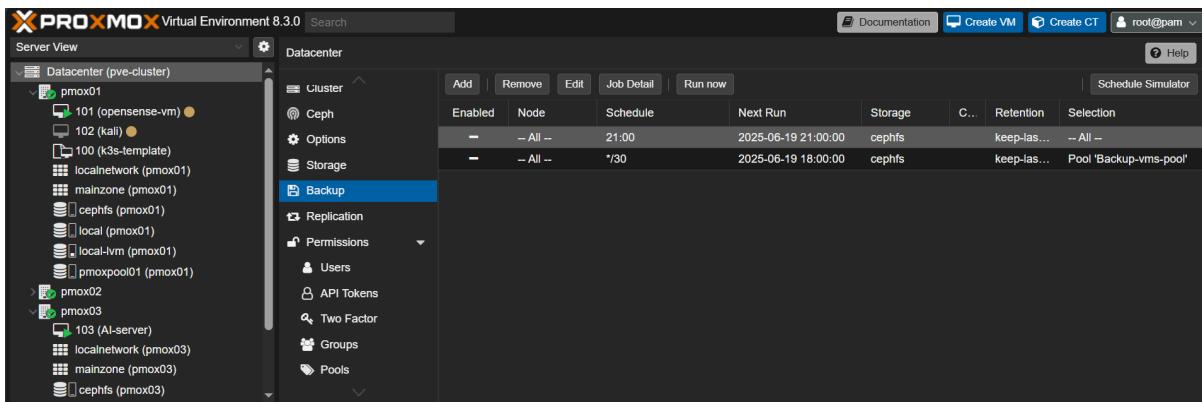


FIGURE 4.16 – Le calendrier de sauvegarde de Proxmox

Cette stratégie de sauvegarde multi-niveaux, combinée à la configuration de Haute Disponibilité, crée un cadre de résilience robuste qui protège le système contre les pannes matérielles immédiates et la perte de données à long terme.

## 4.4 Couche d’Orchestration de Conteneurs : Kubernetes Haute Disponibilité avec k3s

Avec l’infrastructure hyper-convergée sous-jacente entièrement opérationnelle, la couche suivante de l’architecture — la plateforme d’orchestration de conteneurs — a été déployée. Cette recherche utilise **k3s**, une distribution Kubernetes légère mais entièrement conforme, pour former un cluster à haute disponibilité. L’ensemble du cycle de vie des machines virtuelles hébergeant ce cluster est géré à l’aide d’une approche **Infrastructure as Code (IaC)** avec Terraform.

### 4.4.1 Préparation du Template de VM

Un prérequis pour le provisionnement automatisé est une image de base standardisée, ou template. Un template de machine virtuelle nommé **k3s-template** (ID 100) a été créé à cet effet. Le template a été préparé en installant un **OS Ubuntu Server** minimal et en le configurant pour supporter **cloud-init**. Cela permet la personnalisation automatisée des nouvelles VMs au premier démarrage, y compris la définition des noms d’hôte, des configurations réseau et des informations d’identification des utilisateurs. Le script spécifique utilisé pour créer et configurer ce template est détaillé en **Annexe A.2**.

### 4.4.2 Provisionnement Automatisé des Nœuds Kubernetes avec Terraform

Pour garantir un processus de déploiement répétable et cohérent pour les noeuds Kubernetes, **Terraform** a été utilisé. Une configuration Terraform modulaire a été développée pour définir et provisionner par programme les machines virtuelles requises en clonant le `k3s-template`. Cette approche **Infrastructure as Code (IaC)** permet à l'infrastructure d'être **versionnée, facilement mise à l'échelle** en modifiant des variables, et gérée via des **flux de travail automatisés**.

La configuration Terraform se compose de plusieurs fichiers :

- `providers.tf` : Définit le fournisseur Proxmox requis.
- `variables.tf` : Déclare tous les paramètres configurables, tels que le nombre de VMs, les noms et les allocations de ressources.
- `main.tf` : Contient la logique principale qui définit la ressource `proxmox_vm_qemu` à créer.
- `credentials.auto.tfvars` : Fournit les valeurs spécifiques pour les variables, définissant l'état souhaité de l'infrastructure.

Le fichier `credentials.auto.tfvars` a été configuré pour déployer un cluster de cinq noeuds (trois noeuds de plan de contrôle, deux noeuds de travail) avec des allocations de ressources, des paramètres réseau et des politiques de résilience spécifiques, comme décrit précédemment.

Avec tous les fichiers de configuration placés dans un seul répertoire sur un poste de travail de gestion, le déploiement a été exécuté avec deux commandes :

1. `terraform init` : Pour initialiser le répertoire de travail et télécharger le plugin du fournisseur Proxmox.
2. `terraform apply` : Pour analyser la configuration, présenter un plan d'exécution et, après confirmation, créer les ressources dans Proxmox.

L'exécution réussie de ce processus a abouti à la création de cinq nouvelles machines virtuelles, entièrement configurées et prêtes pour l'étape suivante. La Figure 4.17 montre la configuration matérielle d'un nœud de travail provisionné (`worker-node-2`), confirmant les 6 coeurs, 8 Go de RAM spécifiés, et sa connexion au `mainvnet`.

## Chapitre 4. Implémentation Pratique

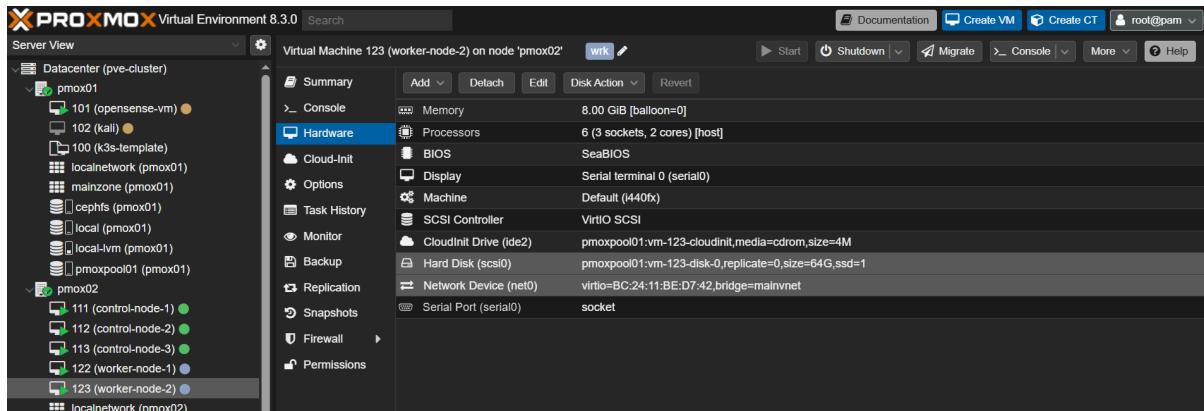


FIGURE 4.17 – La configuration matérielle de la VM worker-node-2 provisionnée

Tandis que la Figure 4.18 illustre la configuration cloud-init appliquée à une VM, montrant l'injection réussie de l'adresse IP statique, des serveurs DNS et des informations d'identification de l'utilisateur définies dans les variables Terraform.

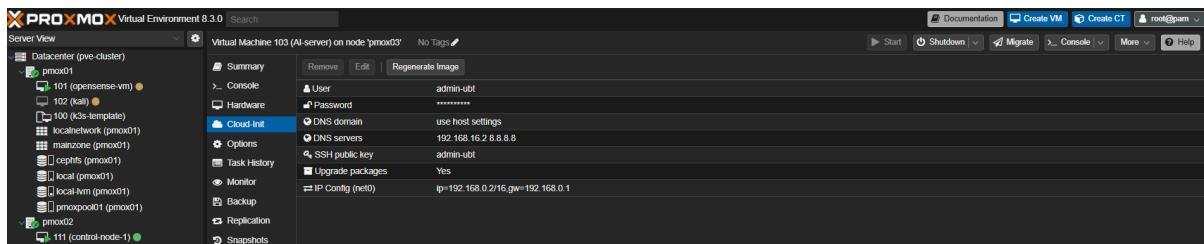


FIGURE 4.18 – La configuration Cloud-Init appliquée à une VM provisionnée

### 4.4.3 Configuration et Déploiement du Cluster k3s HA avec Ansible

Suite au provisionnement automatisé des machines virtuelles avec Terraform, la phase suivante s'est concentrée sur la tâche de gestion de configuration consistant à installer et à mettre en place un cluster k3s à haute disponibilité. Pour garantir un déploiement fiable, répétable et conforme aux meilleures pratiques, cette recherche a tiré parti du projet open-source populaire

`techno-tim/k3s-ansible`. Ce playbook Ansible automatise l'ensemble du processus, de la préparation des nœuds au démarrage d'un cluster multi-maîtres et à l'intégration de composants essentiels.

#### a. Personnalisation du Playbook Ansible

Avant l'exécution, le projet cloné a été adapté aux paramètres spécifiques de cet environnement de recherche. Cela a impliqué de renommer le fichier de configuration

ansibile.example.cfg en ansible.cfg et de préparer la structure de l'inventaire en renommant le répertoire inventory/sample en inventory/my-cluster.

Les personnalisations principales ont ensuite été effectuées dans les fichiers d'inventaire et de variables :

1. **Inventaire des Hôtes (hosts.ini)** : Le fichier d'inventaire a été rempli avec les adresses IP des cinq VMs provisionnées. Les trois noeuds du plan de contrôle ont été placés dans le groupe [master], et les deux noeuds de travail ont été placés dans le groupe [node], comme le montre la Figure 4.19.

FIGURE 4.19 – Le fichier hosts.ini

2. Connexion et Élévation de Privilèges (`k3s_cluster.yaml`) : Un fichier a été créé à l'emplacement `inventory/my-cluster/group_vars/k3s_cluster.yaml` pour définir comment Ansible se connecte aux noeuds cibles. Comme on le voit sur la Figure 4.20, ce fichier spécifie l'`ansible_user` (`admin-ubt`), le chemin vers la clé privée SSH pour l'authentification, et les paramètres pour l'élevation de privilèges (`become: true, become_method: sudo`).



The screenshot shows a text editor window titled 'Mousepad' with the file path '~/.ansible/k3s-ansible/inventory/my-cluster/group\_vars/k3s\_cluster.yaml'. The menu bar includes File, Edit, Search, View, Document, and Help. The toolbar contains icons for new, open, save, cut, copy, paste, find, and search. The main text area displays the following YAML configuration:

```
1 ansible_user: admin-ubt
2 ansible_ssh_private_key_file: ~/.ssh/id_ed25519
3 ansible_python_interpreter: /usr/bin/python3
4 ansible_connection: ssh
5 ansible_sshpass_prompt: Enter passphrase for key '/home/kali/.ssh/id_ed25519'
6 ansible_ssh_pass: admin
7 ansible_become: true
8 ansible_become_method: sudo
9 ansible_become_pass: admin
```

FIGURE 4.20 – Le fichier k3s\_cluster.yaml

3. Configuration Spécifique au Cluster (all.yaml) : Les variables clés dans le fichier group\_vars/all/all.yaml ont été modifiées pour définir l'architecture du cluster :

- `ansible_user` : Ceci a été défini sur `admin-ubt` pour correspondre à l'utilisateur créé par cloud-init.
- `apiserver_endpoint` : Ceci a été défini sur `192.168.0.9`. Le playbook utilise cette variable pour configurer **Kube-vip**, qui fournit une adresse IP virtuelle (VIP) stable pour le plan de contrôle à haute disponibilité. Cela garantit que le serveur d'API Kubernetes reste accessible même si l'un des noeuds maîtres tombe en panne.
- `metal_lb_ip_range` : Ceci a été configuré sur `192.168.0.246-192.168.0.254` pour mettre en place l'exposition des services sur site.

### b. Déploiement et Vérification

Avec la configuration adaptée à l'environnement, le déploiement a été initié en exécutant le playbook Ansible principal depuis le poste de travail de gestion : `ansible-playbook site.yml`.

Le playbook a exécuté une série de rôles qui ont préparé les noeuds, installé les binaires k3s, configuré le premier maître, joint les autres maîtres pour former un cluster etcd, et enfin joint les noeuds de travail.

Après le déploiement, la connexion à `control-node-1` et l'exécution de `kubectl get nodes` ont confirmé la création réussie du cluster. La sortie de la Figure 4.21 montre clairement un cluster de cinq noeuds avec trois noeuds de plan de contrôle, etcd, maîtres et deux noeuds de travail, tous dans un état `Ready` et exécutant une version cohérente de k3s (`v1.30.2+k3s2`).

admin-ubt@control-node-1:~\$ sudo kubectl get nodes				
NAME	STATUS	ROLES	AGE	VERSION
control-node-1	Ready	control-plane,etcd,master	3d2h	v1.30.2+k3s2
control-node-2	Ready	control-plane,etcd,master	3d2h	v1.30.2+k3s2
control-node-3	Ready	control-plane,etcd,master	3d2h	v1.30.2+k3s2
worker-node-1	Ready	<none>	3d2h	v1.30.2+k3s2
worker-node-2	Ready	<none>	3d2h	v1.30.2+k3s2

FIGURE 4.21 – La sortie de `kubectl get nodes`

### c. Exposition de Services sur Site avec MetalLB

Une fonctionnalité critique automatisée par ce playbook Ansible est l'installation de **MetalLB**. Dans les environnements sur site comme celui-ci, Kubernetes n'a pas de fournisseur natif pour les services de type **LoadBalancer**. MetalLB comble cette lacune en fournissant une solution d'équilibrage de charge réseau qui s'intègre avec l'équipement réseau standard. En répondant aux requêtes ARP, il rend une adresse IP privée de son pool configuré accessible sur le réseau local.

La variable `metal_lb_ip_range` configurée précédemment garantit que lorsqu'un service de type `LoadBalancer` est créé dans Kubernetes, MetalLB lui attribuera automatiquement une IP disponible de la plage `192.168.0.246-192.168.0.254`, rendant le service accessible depuis le `mainvnet`.

Cette approche automatisée, pilotée par un playbook, assure non seulement un déploiement initial correct et résilient, mais fournit également une voie claire pour une future scalabilité. L'ajout de nouveaux nœuds de travail, par exemple, nécessite simplement d'ajouter leurs adresses IP au fichier `hosts.ini` et de ré-exécuter le playbook. L'établissement réussi de cette plateforme Kubernetes robuste fournit la base pour le déploiement d'applications conteneurisées. Cela permet d'installer et de gérer les applications à l'aide d'outils cloud-native standard, tels que les charts Helm ou les manifestes Kubernetes déclaratifs.

## 4.5 Couche de Surveillance, de Gestion et d'Automatisation

Le déploiement réussi de l'infrastructure et de la plateforme d'orchestration de conteneurs n'est pas l'étape finale. Un système robuste nécessite des outils complets pour la surveillance, la gestion et la maintenance. Cette section détaille la couche opérationnelle de l'architecture, qui intègre les flux de travail d'automatisation, établit une observabilité à l'échelle du système et met en œuvre une stratégie de protection des données résiliente.

### 4.5.1 Le Flux de Travail Intégré IaC et d'Automatisation

L'implémentation de cette plateforme reposait sur un flux de travail cohésif d'**Infrastructure as Code (IaC)** et d'automatisation. **Terraform** a été utilisé pour gérer le provisionnement déclaratif des ressources fondamentales — les machines virtuelles. Cela a permis de s'assurer que l'état souhaité de l'infrastructure était défini dans le code. Par la suite, **Ansible** a été utilisé pour la gestion de la configuration, installant et configurant par programme le cluster k3s complexe et à haute disponibilité au-dessus des VMs provisionnées. Cette approche à deux outils offre une méthode puissante, répétable et versionnée pour déployer l'ensemble de la plateforme à partir de zéro.

### 4.5.2 Surveillance Cloud-Native avec Prometheus et Grafana

Pour obtenir une observabilité dans le cluster Kubernetes, une pile de surveillance cloud-native composée de **Prometheus** et **Grafana** a été déployée à l'aide d'un **chart**

## Chapitre 4. Implémentation Pratique

**Helm** pour fournir une observabilité du cluster. La Figure 4.22 confirme le déploiement réussi de tous les pods et services nécessaires.

NAME	READY	STATUS	RESTARTS	AGE
grafana-85b9748f55-g2hw2	1/1	Running	8 (25m ago)	3d23h
prometheus-alertmanager-0	1/1	Running	3 (25m ago)	3d23h
prometheus-kube-state-metrics-786488967b-n9mb4	1/1	Running	112 (24m ago)	3d23h
prometheus-prometheus-node-exporter-8kk12	1/1	Running	3 (26m ago)	3d23h
prometheus-prometheus-node-exporter-9kxb5	1/1	Running	3 (25m ago)	3d23h
prometheus-prometheus-node-exporter-mkk78	1/1	Running	3 (25m ago)	3d23h
prometheus-prometheus-node-exporter-wnhq2	1/1	Running	3 (25m ago)	3d23h
prometheus-prometheus-node-exporter-zlqhm	1/1	Running	3 (24m ago)	3d23h
prometheus-prometheus-pushgateway-849bb86467-t7zst	1/1	Running	3 (25m ago)	3d23h
prometheus-server-6648f86775-k82r2	2/2	Running	6 (24m ago)	3d23h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
grafana	LoadBalancer	10.43.143.51	192.168.0.247	80:30878/TCP	3d23h
kubernetes	ClusterIP	10.43.0.1	<none>	443/TCP	4d18h
prometheus-alertmanager	ClusterIP	10.43.148.189	<none>	9093/TCP	3d23h
prometheus-alertmanager-headless	ClusterIP	None	<none>	9093/TCP	3d23h
prometheus-kube-state-metrics	ClusterIP	10.43.60.182	<none>	8080/TCP	3d23h
prometheus-prometheus-node-exporter	ClusterIP	10.43.153.243	<none>	9100/TCP	3d23h
prometheus-prometheus-pushgateway	ClusterIP	10.43.19.183	<none>	9091/TCP	3d23h
prometheus-server	LoadBalancer	10.43.29.69	192.168.0.246	80:30222/TCP	3d23h

FIGURE 4.22 – la pile de surveillance Prometheus et Grafana

Après une configuration initiale pour ajouter la source de données Prometheus et importer un tableau de bord communautaire, le système a fourni un aperçu immédiat des métriques clés du cluster (Figure 4.23).

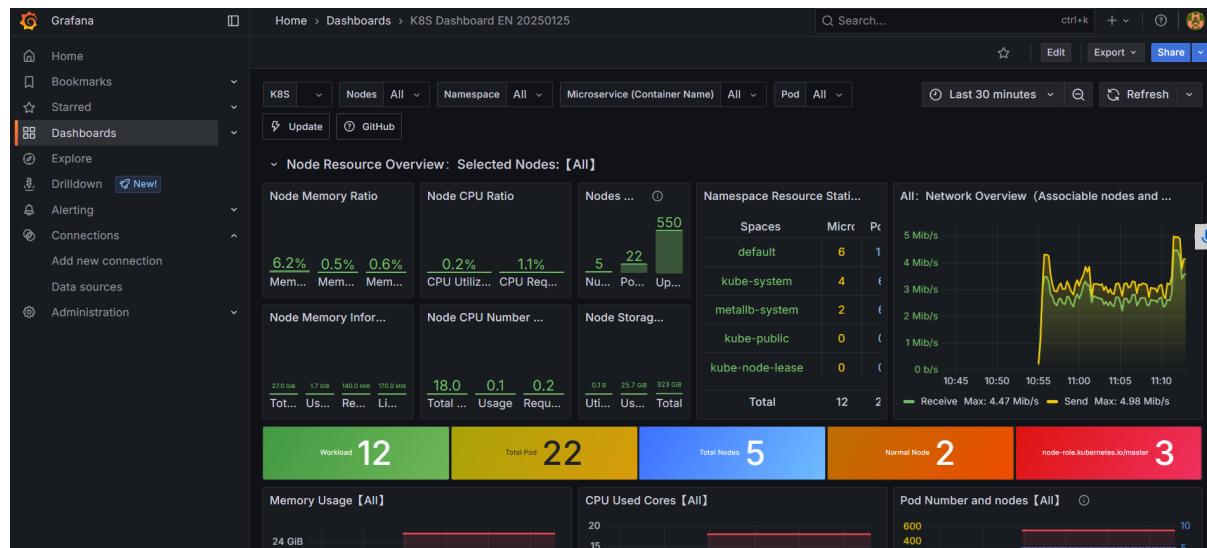


FIGURE 4.23 – Le tableau de bord Grafana pour le k3s\_cluster

### 4.5.3 Interfaces de Gestion Centralisées

La gestion du système mis en œuvre est assurée par deux interfaces principales, chacune correspondant à une couche différente de l'architecture. L' **interface web de Proxmox VE** sert de plan de gestion pour la couche physique et de virtualisation, offrant un

contrôle sur les VMs, le stockage, le réseau, la HA et les sauvegardes. Pour la couche applicative, l' **API Kubernetes**, accessible via l'outil en ligne de commande **kubectl**, offre un contrôle complet sur toutes les charges de travail conteneurisées, les services et les ressources du cluster.

Cette couche opérationnelle unifie la plateforme grâce à un flux de travail d'automatisation cohésif, une surveillance complète et des interfaces de gestion distinctes. Le système résultant est robuste, observable et entièrement préparé pour le déploiement de la couche finale d'IA Agentique, remplissant les objectifs principaux de cette phase d'implémentation.

## 4.6 La Couche d'IA Agentique

Le composant final et le plus critique de l'architecture est la **couche d'IA Agentique**. Cette couche sert de "cerveau" à la plateforme, la transformant d'un système qui peut être automatisé en un système capable de comprendre et de traiter des requêtes en langage naturel pour effectuer un **provisionnement intelligent**. Cette section détaille la mise en place du serveur IA dédié et le développement du **Système Multi-Agents (SMA)** qui alimente cette capacité.

### 4.6.1 Provisionnement du Serveur IA et Sélection du LLM

Pour héberger la charge de travail agentique, une machine virtuelle dédiée nommée **AI-server** a été provisionnée en utilisant le même **flux de travail Terraform** automatisé établi précédemment. Cela a garanti que sa configuration était cohérente avec le reste des ressources de la plateforme.

Le plan de recherche initial prévoyait d'utiliser un **Grand Modèle de Langage (LLM)** hébergé localement pour garantir que le système soit entièrement autonome. Cependant, des tests préliminaires ont révélé que l'exécution d'un modèle open-source performant localement consommait une quantité disproportionnée de ressources CPU et mémoire, l'emportant sur les avantages pour la portée de ce projet.

Par conséquent, la décision a été prise de pivoter vers une approche plus économique en ressources basée sur une API. L' **API Google Gemini** a été sélectionnée comme moteur d'intelligence principal. Cette approche a déchargé la demande de calcul significative du LLM tout en donnant accès à des capacités de raisonnement, de planification et de génération de code de pointe. Une clé API sécurisée a été générée et configurée sur le **AI-server** pour permettre la communication avec le modèle Gemini.

#### 4.6.2 Développement de l'Agent de Provisionnement Proxmox

Avec le serveur IA et l'accès au LLM établis, l'agent principal a été développé en utilisant le **Google Agent Development Kit (ADK)**. L'ADK a fourni le cadre nécessaire pour construire un **Système Multi-Agents (SMA)** sophistiqué, où différents agents collaborent pour répondre à la demande d'un utilisateur.

En raison de la complexité et des contraintes de temps de cette recherche, la portée du SMA a été exclusivement axée sur le **provisionnement intelligent de Proxmox**. L'architecture de ce système, représentée à la Figure 4.24, se compose de plusieurs agents spécialisés :

- **Manager** : L'orchestrateur qui reçoit la requête initiale de l'utilisateur et délègue les tâches aux autres agents.
- **DataCollectionAgent** : Collecte des informations en temps réel de l'environnement en se connectant à l'API Proxmox.
- **AnalysesAndStrategyAgent** : Interprète l'objectif de l'utilisateur et les données collectées pour former un plan de haut niveau.
- **ManifestRefinementLoop** : Un sous-système itératif contenant des agents qui génèrent, examinent et affinent le fichier de configuration Terraform final jusqu'à ce qu'il réponde aux normes requises.

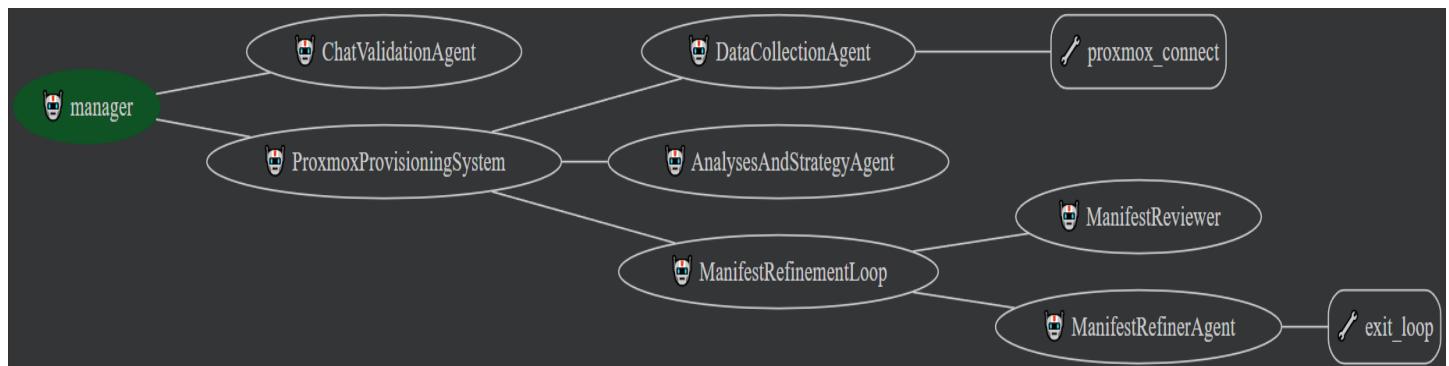


FIGURE 4.24 – L'architecture du Système de Provisionnement Intelligent.

Le déroulement détaillé de ce flux de travail collaboratif est présenté en **Annexe C.2**

#### 4.6.3 Interaction Utilisateur et Flux de Travail

Le **Système Multi-Agents** développé est exposé à l'utilisateur final via une interface web propre, hébergée localement. La Figure 4.25 capture une interaction utilisateur typique, mettant en valeur la nature conversationnelle et multi-agents du système alors que le **ValidationAgent** engage l'utilisateur pour clarifier une demande de provisionnement.

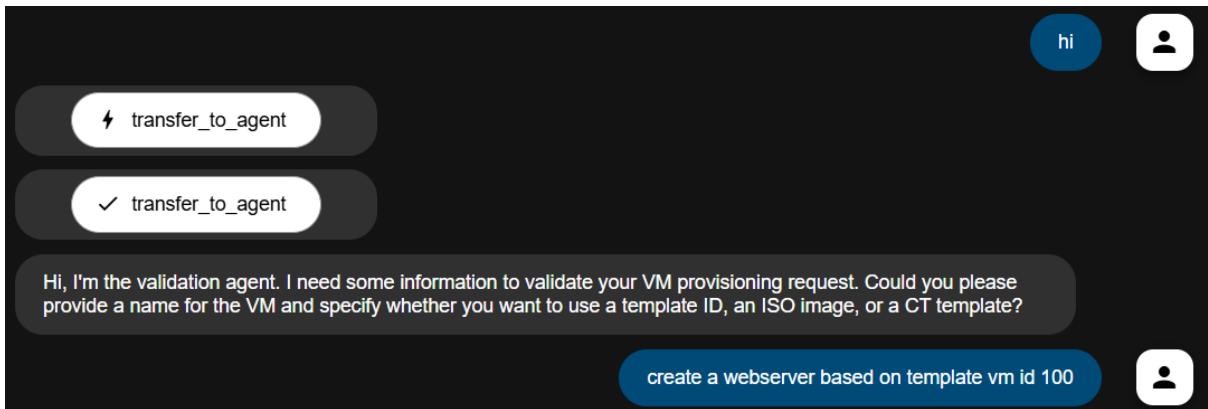


FIGURE 4.25 – Un exemple d’interaction avec l’IA Agentique

Contrairement à un simple outil en ligne de commande, le système est conçu pour gérer l’ambiguïté par le dialogue :

1. Un utilisateur peut initier une conversation avec une demande générale ou incomplète.
2. L’agent **Manager** achemine intelligemment la demande vers l’agent spécialisé approprié — dans ce cas, le **ValidationAgent**.
3. Le **ValidationAgent** engage alors un dialogue avec l’utilisateur pour recueillir les détails spécifiques requis pour le provisionnement, tels que le nom de la VM et sa source (**template**, **ISO**, etc.).
4. L’utilisateur fournit les détails clarifiés ("créer un serveur web basé sur le template **vm id 100**"), que l’agent peut ensuite utiliser pour continuer.

Cette étape de validation interactive est cruciale pour garantir que la configuration générée est exacte avant d’être transmise aux agents de provisionnement. Une fois toutes les informations validées, le flux de travail du SMA se poursuit, générant finalement un fichier Terraform (**.tf**) pour l’examen final et l’exécution par l’utilisateur. Cette approche "humain dans la boucle" garantit la sécurité tout en tirant parti de l’intelligence de l’agent pour gérer la tâche complexe d’écriture du code de configuration de l’infrastructure, atteignant ainsi l’objectif du projet de **provisionnement intelligent**.

## 4.7 Validation de la Plateforme : Scénarios Intégrés

Les sections précédentes ont détaillé l’implémentation des couches individuelles de la plateforme. Cette section vise à démontrer la synergie entre ces couches en présentant quatre scénarios pratiques. Ces tests valident la fonctionnalité de bout en bout de la plateforme, de sa capacité à une mise à l’échelle transparente et à la résilience face aux pannes matérielles, à son efficacité dans le déploiement automatisé d’applications et le provisionnement intelligent.

### 4.7.1 Scénario 1 : Mise à l’Échelle Pilotée par l’Automatisation

Ce scénario démontre la capacité de la plateforme à se mettre à l'échelle horizontalement en ajoutant un nouvel hôte physique et un nouveau nœud de travail Kubernetes en utilisant le flux de travail IaC établi.

- Mise à l'échelle de la couche Hyperviseur :** Le processus a commencé avec le cluster Proxmox initial à trois nœuds. Un nouveau serveur physique, pmox04, a été installé avec Proxmox VE et joint au cluster, augmentant instantanément les ressources de calcul et de mémoire totales disponibles du cluster, comme le montre la Figure 4.26.

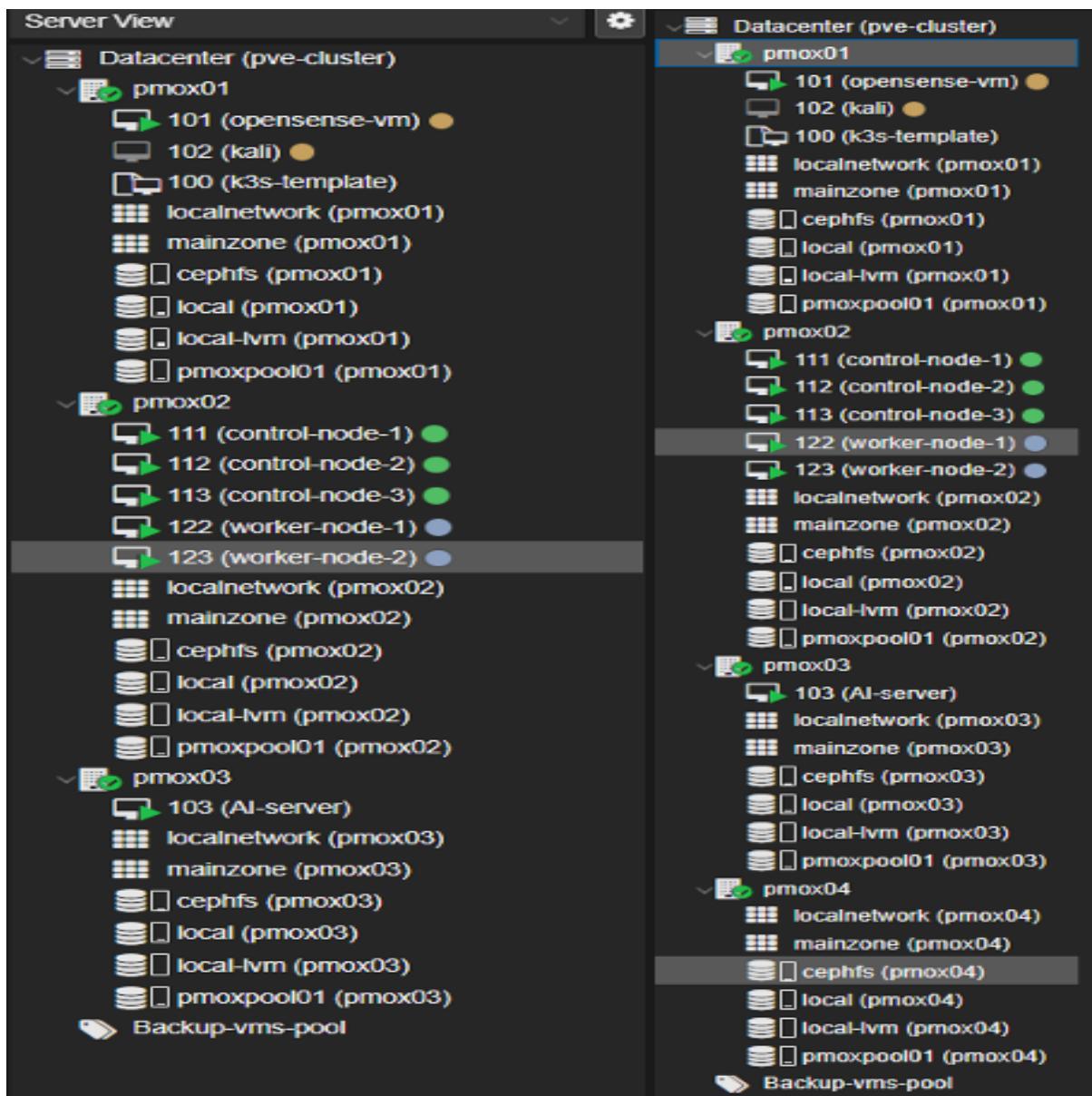


FIGURE 4.26 – La vue des serveurs du cluster Proxmox avant (gauche) et après (droite) la mise à l'échelle de trois à quatre nœuds physiques.

2. **Mise à l'échelle de la couche Kubernetes :** Pour gérer une charge applicative accrue, un nouveau nœud de travail a été provisionné et configuré à l'aide du flux de travail automatisé. Après avoir ajouté `worker-node-3` aux fichiers de configuration Terraform et Ansible, les commandes respectives `terraform apply` et `ansible-playbook site.yml` ont été exécutées.
3. **Vérification :** Le résultat réussi est confirmé à la Figure 4.27, où la commande `kubectl get nodes` montre que le nouveau `worker-node-3` a rejoint le cluster et est dans un état `Ready`, disponible pour accepter de nouvelles charges de travail seulement 91 secondes après sa création.

admin-ubt@control-node-3:~\$ sudo kubectl get nodes				
NAME	STATUS	ROLES	AGE	VERSION
control-node-1	Ready	control-plane,etcd,master	5d3h	v1.30.2+k3s2
control-node-2	Ready	control-plane,etcd,master	5d3h	v1.30.2+k3s2
control-node-3	Ready	control-plane,etcd,master	5d3h	v1.30.2+k3s2
worker-node-1	Ready	<none>	5d3h	v1.30.2+k3s2
worker-node-2	Ready	<none>	5d3h	v1.30.2+k3s2
worker-node-3	Ready	<none>	91s	v1.30.2+k3s2

FIGURE 4.27 – L'état des nœuds du cluster Kubernetes après l'opération de mise à l'échelle automatisée

Ce scénario prouve que la mise à l'échelle de la plateforme n'est pas un projet complexe et manuel, mais un processus simple, déclaratif et répétable, confirmant l'efficacité du flux de travail automatisé.

#### 4.7.2 Scénario 2 : Tolérance aux Pannes Multi-Couches

Ce scénario simule des pannes matérielles et logicielles pour tester l'architecture de haute disponibilité multi-couches.

1. **Panne de la couche Hyperviseur (Proxmox HA) :** L'hôte nouvellement ajouté, `pmax04`, a été brusquement éteint. Le gestionnaire de Haute Disponibilité de Proxmox a détecté la panne et, comme le montre la Figure 4.28, a automatiquement migré la VM `worker-node-3` (ID 124) vers un hôte survivant (`pmax01`). Cette migration transparente est possible car le disque de la VM réside sur le stockage Ceph partagé, accessible depuis tous les nœuds du cluster.

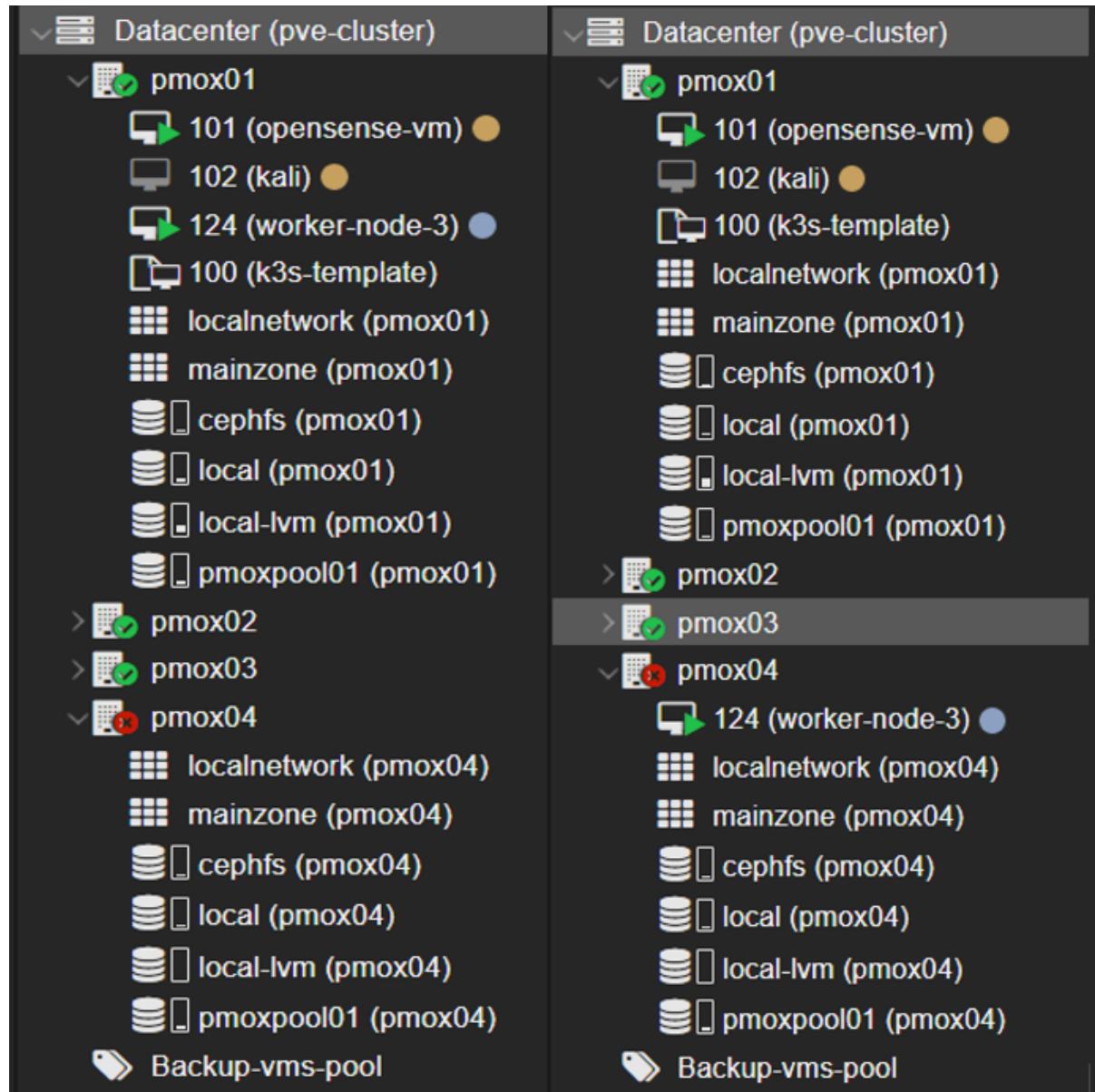


FIGURE 4.28 – Basculement de Proxmox HA en action. La VM `worker-node-3` est automatiquement migrée de l'hôte défaillant `pmox04` (gauche) vers l'hôte sain `pmox01` (droite).

2. Panne de la couche Conteneur (k3s HA) : Ensuite, un nœud critique du plan de contrôle, `control-node-1`, a été arrêté. La résilience du cluster k3s a été immédiatement évidente. La Figure 4.29 montre la sortie de `kubectl get nodes`, où `control-node-1` est correctement identifié comme `NotReady`. Malgré cela, le cluster reste entièrement opérationnel car le **cluster etcd** maintient le quorum et **Kube-vip** a automatiquement déplacé l'IP virtuelle du plan de contrôle vers un maître sain.

NAME	STATUS	ROLES	AGE	VERSION
control-node-1	NotReady	control-plane,etcd,master	5d3h	v1.30.2+k3s2
control-node-2	Ready	control-plane,etcd,master	5d3h	v1.30.2+k3s2
control-node-3	Ready	control-plane,etcd,master	5d3h	v1.30.2+k3s2
worker-node-1	Ready	<none>	5d3h	v1.30.2+k3s2
worker-node-2	NotReady	<none>	5d3h	v1.30.2+k3s2
worker-node-3	NotReady	<none>	24m	v1.30.2+k3s2

FIGURE 4.29 – État du cluster Kubernetes lors d'une panne du plan de contrôle

**3. Vérification de la Disponibilité du Service :** Le test ultime de la résilience est la disponibilité continue des applications. Comme le confirme la Figure 4.30, même avec un nœud maître en panne, tous les pods d'application pour la pile de surveillance Prometheus et Grafana restent dans un état `Running`, et leurs services `LoadBalancer` sont toujours actifs et exposent leurs IP externes. La plateforme a résisté avec succès à la panne sans perte de service applicatif.

NAME	READY	STATUS	RESTARTS	AGE
grafana-85b9748f55-g2hw2	1/1	Running	8 (25m ago)	3d23h
prometheus-alertmanager-0	1/1	Running	3 (25m ago)	3d23h
prometheus-kube-state-metrics-786488967b-n9mb4	1/1	Running	112 (24m ago)	3d23h
prometheus-prometheus-node-exporter-8kk12	1/1	Running	3 (26m ago)	3d23h
prometheus-prometheus-node-exporter-9kxb5	1/1	Running	3 (25m ago)	3d23h
prometheus-prometheus-node-exporter-mkk78	1/1	Running	3 (25m ago)	3d23h
prometheus-prometheus-node-exporter-wnhq2	1/1	Running	3 (25m ago)	3d23h
prometheus-prometheus-node-exporter-zlqhm	1/1	Running	3 (24m ago)	3d23h
prometheus-prometheus-pushgateway-849bb86467-t7zst	1/1	Running	3 (25m ago)	3d23h
prometheus-server-6648f86775-k82r2	2/2	Running	6 (24m ago)	3d23h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
grafana	LoadBalancer	10.43.143.51	192.168.0.247	80:30878/TCP	3d23h
kubernetes	ClusterIP	10.43.0.1	<none>	443/TCP	4d18h
prometheus-alertmanager	ClusterIP	10.43.148.189	<none>	9093/TCP	3d23h
prometheus-alertmanager-headless	ClusterIP	None	<none>	9093/TCP	3d23h
prometheus-kube-state-metrics	ClusterIP	10.43.60.182	<none>	8080/TCP	3d23h
prometheus-prometheus-node-exporter	ClusterIP	10.43.153.243	<none>	9100/TCP	3d23h
prometheus-prometheus-pushgateway	ClusterIP	10.43.19.183	<none>	9091/TCP	3d23h
prometheus-server	LoadBalancer	10.43.29.69	192.168.0.246	80:30222/TCP	3d23h

FIGURE 4.30 – Les pods et services applicatifs restent entièrement opérationnels pendant la panne du nœud du plan de contrôle.

Ce test valide le concept critique de résilience multi-couches. Les couches de virtualisation et d'orchestration de conteneurs ont répondu de manière autonome aux pannes, travaillant de concert pour maintenir la disponibilité de l'infrastructure et des applications.

#### 4.7.3 Scénario 3 : Déploiement d'Application de Bout en Bout

Ce scénario démontre l'objectif principal de la plateforme : déployer, exposer et servir de manière transparente une **application conteneurisée**. Pour ce test, une application

web d'exemple a été déployée à l'aide d'un fichier **manifeste Kubernetes** standard. Le manifeste définissait un **Deployment** pour s'assurer que trois répliques du pod de l'application étaient toujours en cours d'exécution pour la redondance, et un **Service** de type **LoadBalancer** pour exposer l'application au réseau local.

1. **Initiation et Déploiement** : Le déploiement a été initié en appliquant le fichier manifeste au cluster à l'aide d'une seule commande : `kubectl apply -f web-application.yaml`.
2. **Kubernetes et MetalLB en Action** : Dès réception du manifeste, le plan de contrôle de Kubernetes est entré en action. Le contrôleur de Deployment a créé un ReplicaSet, qui à son tour a planifié l'exécution de trois pods sur les noeuds de travail disponibles. Simultanément, l'objet Service a été créé. Comme son type était spécifié comme **LoadBalancer**, le **contrôleur MetalLB** l'a détecté et lui a attribué une adresse IP disponible de son pool préconfiguré.
3. **Vérification et Résultat** : Le résultat réussi de ce processus est confirmé à la Figure 4.31. La sortie de la commande `kubectl get services` montre le service nouvellement créé `learn-nginx-app-service`. MetalLB l'a provisionné avec succès avec une adresse IP externe de `192.168.0.248`, le rendant accessible sur le `mainvnet`.

<code>└\$ kubectl get services</code>					
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
grafana	LoadBalancer	10.43.143.51	192.168.0.247	80:30878/TCP	8d
kubernetes	ClusterIP	10.43.0.1	<none>	443/TCP	9d
learn-nginx-app-service	LoadBalancer	10.43.43.193	192.168.0.248	80:30087/TCP	4m35s
prometheus-alertmanager	ClusterIP	10.43.148.189	<none>	9093/TCP	8d

FIGURE 4.31 – La sortie de `kubectl get services`

Pour confirmer la fonctionnalité de bout en bout, l'adresse IP externe attribuée a été accédée depuis un navigateur web sur le même réseau. Comme le montre la Figure 4.32, l'interface de l'application web s'est chargée avec succès, prouvant que le trafic était correctement acheminé de l'utilisateur, via le point d'entrée MetalLB, vers le service, et enfin vers l'un des pods de l'application en cours d'exécution.

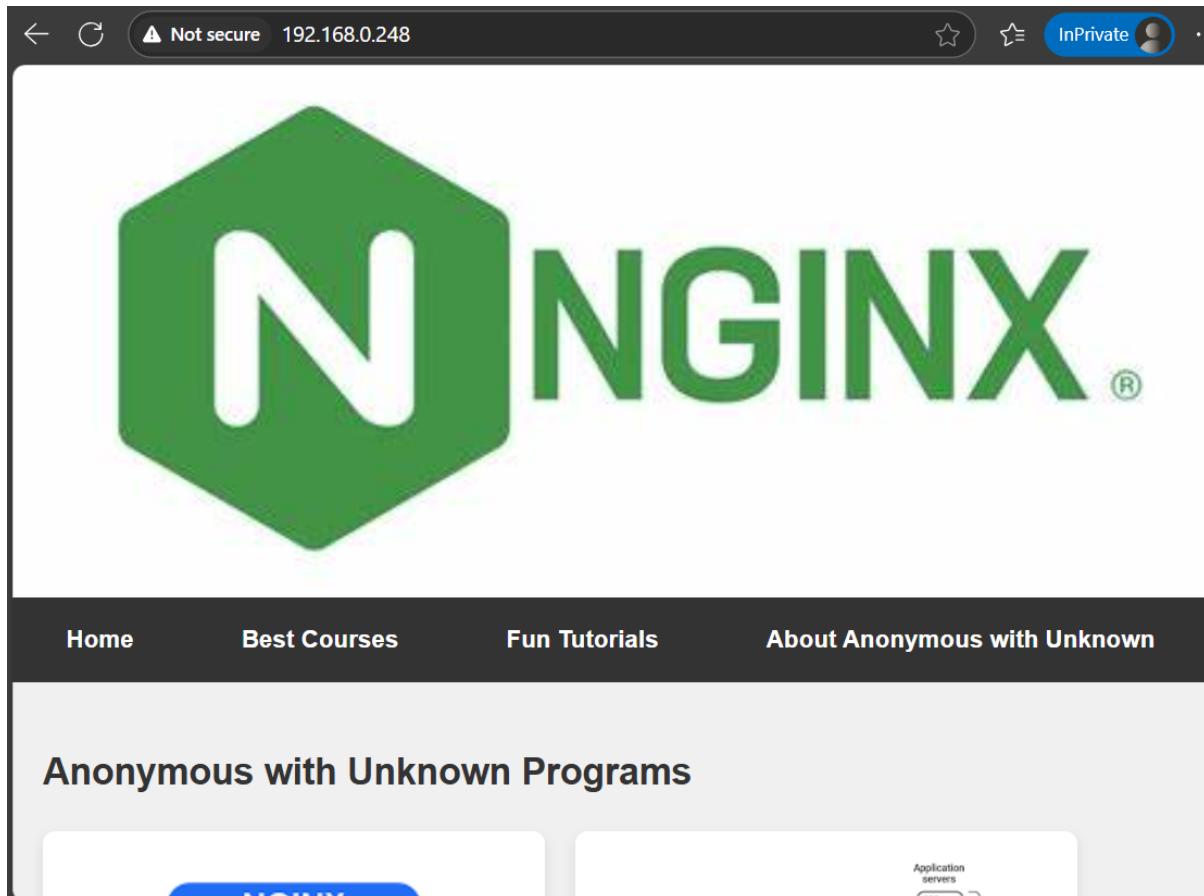


FIGURE 4.32 – Accès réussi à l’application web via son IP de LoadBalancer.

Ce scénario valide la capacité de la plateforme à abstraire les complexités de la planification des conteneurs, de la découverte de services et de l’équilibrage de charge réseau, offrant un mécanisme simple et puissant pour gérer le cycle de vie des applications cloud-native.

### 4.7.4 Scénario 4 : Provisionnement Intelligent avec la Couche d’IA Agentique

Ce scénario final démontre le flux de travail de bout en bout de la **couche d’IA Agentique**, mettant en évidence sa capacité à traduire une requête utilisateur de haut niveau en **langage naturel** en une machine virtuelle provisionnée. Ce test valide l’objectif de recherche principal de réaliser un **provisionnement intelligent**.

1. **Requête Utilisateur et Interprétation par l’Agent** : Le processus commence dans l’interface web, où l’utilisateur soumet une simple requête : "créer un serveur web basé sur le template vm id 100". Comme le montre la Figure 4.33, l’agent extrait les paramètres clés et présente cette compréhension structurée à l’utilisateur pour validation.

## Chapitre 4. Implémentation Pratique

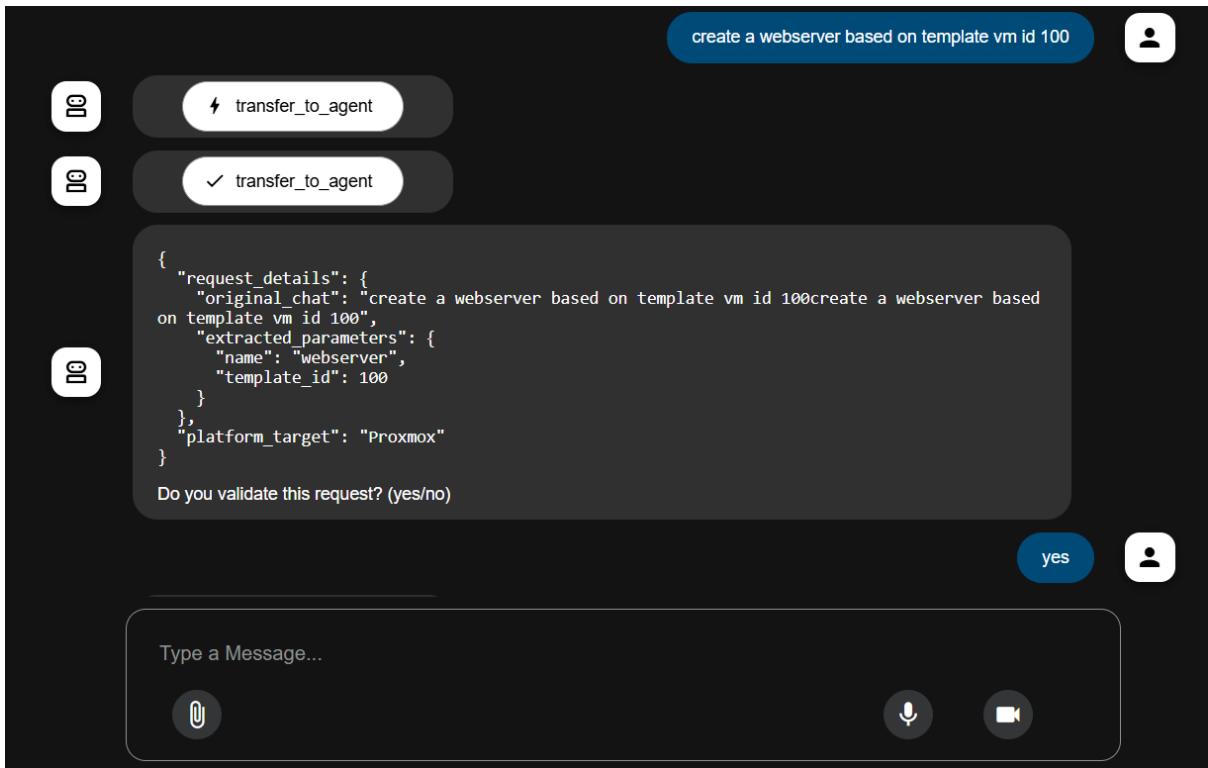


FIGURE 4.33 – L'interprétation par l'agent de la requête de l'utilisateur et la demande de validation subséquente.

2. **Raisonnement de l'Agent et Génération du Manifeste :** Sur validation de l'utilisateur ("oui"), le **Système Multi-Agents** exécute son flux de travail principal. Comme l'agent l'expliqué dans la Figure 4.34), cela implique un processus de raisonnement en plusieurs étapes, y compris la collecte de données, l'analyse stratégique et l'affinement itératif pour produire un **manifeste Terraform** conforme aux meilleures pratiques.

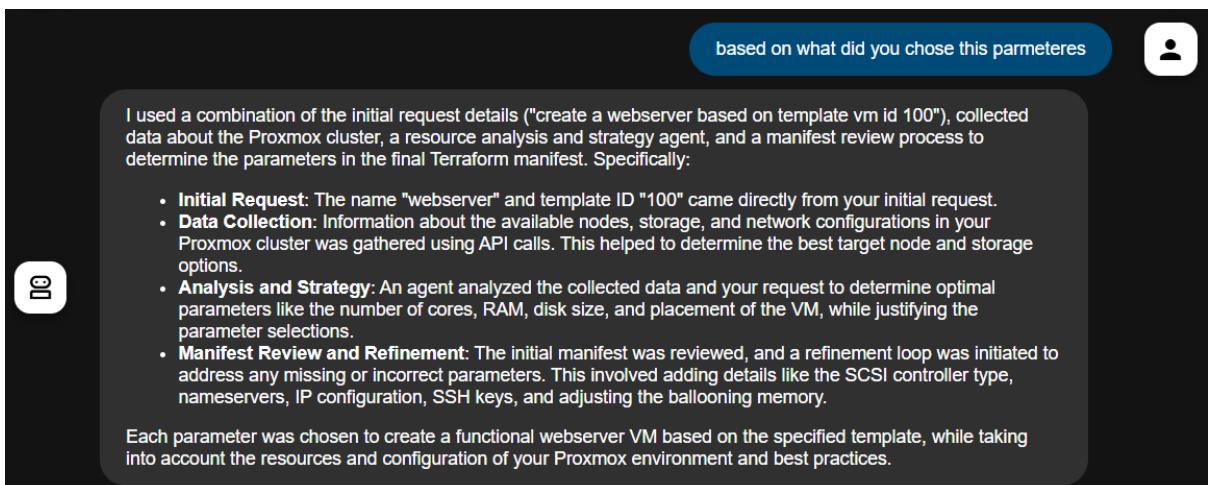


FIGURE 4.34 – L'agent expliquant son processus de raisonnement en plusieurs étapes.

3. **Exécution avec Humain dans la Boucle :** Le fichier Terraform final et validé est

fourni à l'opérateur humain pour des raisons de sécurité et de supervision. L'opérateur exécute alors `terraform apply`, la création réussie de la nouvelle ressource étant montrée à la Figure 4.35.

```

    cipassword = "admin"
}

manifest meets all requirements. Exiting the refinement loop.

exit_loop
exit_loop

proxmox_vm_qemu.vm_terraforms[0]: Still creating... [6m28s elapsed]
proxmox_vm_qemu.vm_terraforms[0]: Still creating... [6m38s elapsed]
proxmox_vm_qemu.vm_terraforms[0]: Creation complete after 6m46s [id=pmox03/qemu/125]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

```

FIGURE 4.35 – L'exécution réussie du fichier Terraform généré par l'agent.

- Vérification du Résultat :** Le résultat est immédiatement visible dans l'interface web de Proxmox. La Figure 4.36 montre l'état de l'hôte pmox03 avant et après l'opération, confirmant que la nouvelle VM `webserver` (ID 125) a été créée avec succès.

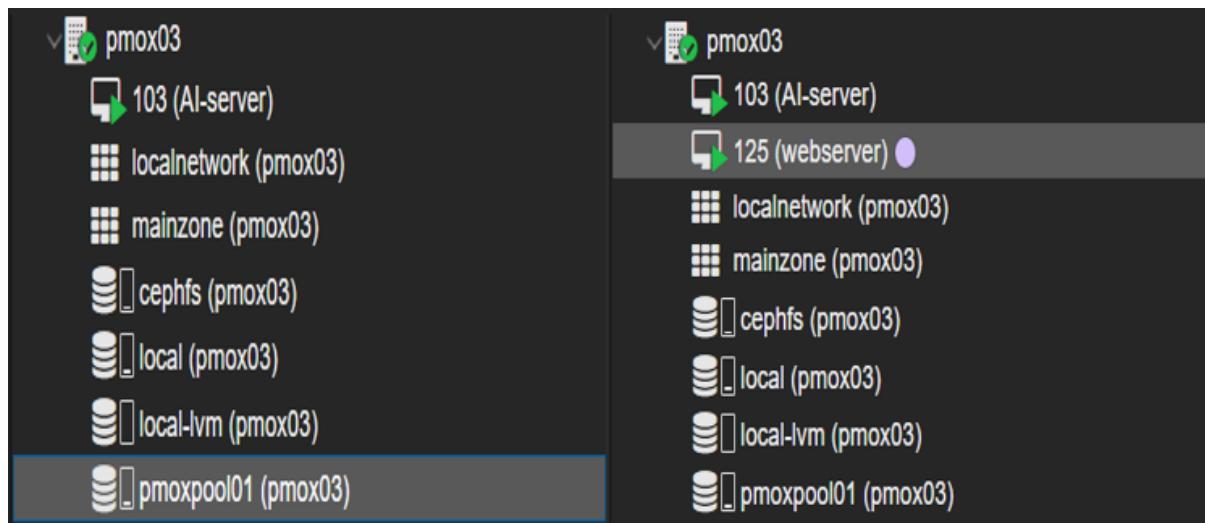


FIGURE 4.36 – La vue du serveur Proxmox avant (gauche) et après (droite) le flux de travail de provisionnement intelligent

Ce scénario valide avec succès l'ensemble du flux de travail agentique, démontrant la capacité du système à comprendre l'intention humaine, à raisonner sur l'état de l'environnement et à générer intelligemment un fichier de configuration complexe, comblant ainsi le fossé entre les commandes de haut niveau et la gestion de l'infrastructure de bas niveau.

## 4.8 Conclusion

Ce chapitre a détaillé l'implémentation complète d'une **plateforme cloud autonome et multi-couches**, traduisant l'architecture conceptuelle en un système entièrement opérationnel. En commençant par le matériel physique, une fondation résiliente a été établie en utilisant **Proxmox VE** pour la virtualisation et **Ceph** pour le stockage hyper-convergé. Sur cette fondation, un cluster **k3s** à haute disponibilité a été déployé, géré entièrement par un flux de travail sophistiqué d'**Infrastructure as Code** qui a tiré parti des forces complémentaires de **Terraform** pour le provisionnement et d'**Ansible** pour la configuration. Ceci a été soutenu par une structure de réseau virtuel robuste gérée par **OPNsense** et une pile d'observabilité complète avec **Prometheus** et **Grafana**.

La fonctionnalité de la plateforme n'a pas seulement été affirmée mais **pratiquement** validée à travers une série de scénarios intégrés. Ces tests ont démontré la capacité du système à une mise à l'échelle transparente et pilotée par l'automatisation, sa **résilience multi-couches** contre les pannes matérielles et logicielles, et son efficacité dans la gestion du cycle de vie de bout en bout des applications cloud-native. L'aboutissement de cet effort a été l'implémentation et le test réussis de la **couche d'IA Agentique**, qui a confirmé la capacité du système à traduire une **commande en langage naturel** de haut niveau en une **ressource d'infrastructure provisionnée**.

En résumé, la phase d'implémentation est terminée. Le résultat est une plateforme cloud sur site entièrement opérationnelle qui atteint avec succès les objectifs de conception architecturale de **résilience, de scalabilité et de provisionnement intelligent**. Ce travail valide avec succès la thèse selon laquelle en intégrant une infrastructure automatisée et multi-couches avec une IA agentique, il est possible de combler le fossé entre l'**intention humaine** de haut niveau et l'**exécution d'infrastructure** de bas niveau, remplissant ainsi les **objectifs principaux de cette recherche**.

# **Conclusion et perspectives**

# Conclusion générale

La complexité croissante des infrastructures informatiques modernes, alimentée par des charges de travail dynamiques, la prévalence des applications conteneurisées et le besoin croissant de scalabilité et de résilience, a rendu nécessaire le développement de solutions de provisionnement de ressources intelligentes et automatisées. Cette thèse a conçu, mis en œuvre et évalué avec succès une architecture multi-couches intelligente qui intègre la virtualisation, l'orchestration de conteneurs et un système multi-agents (SMA) piloté par l'IA pour optimiser le provisionnement et la mise à l'échelle des ressources dans des clusters virtualisés. En s'appuyant sur des technologies open-source et des modèles d'IA de pointe, le système proposé comble le fossé entre les avancées théoriques et les implémentations pratiques dans l'infrastructure cloud. Les couches fondamentales de cette architecture Proxmox VE pour la virtualisation et Ceph pour le stockage hyperconvergé fournissent une base robuste et évolutive pour l'abstraction des ressources et la haute disponibilité. La distribution Kubernetes légère (k3s) ajoute une couche d'orchestration de conteneurs flexible, permettant un déploiement et une gestion transparents des applications. De plus, l'intégration d'une pile de surveillance native pour le cloud (Prometheus et Grafana) assure une observabilité complète, tandis que l'utilisation d'outils d'Infrastructure as Code (IaC) (Terraform et Ansible) simplifie l'automatisation des flux de travail complexes. La contribution la plus significative de cette recherche réside dans la mise en œuvre de la couche d'IA agentique, qui introduit des capacités de provisionnement intelligent via un système multi-agents (SMA). En intégrant le modèle Google Gemini, le système transforme les requêtes utilisateur en langage naturel en solutions d'infrastructure déployables grâce à un raffinement et une validation itératifs. Cette approche avec intervention humaine (human-in-the-loop) garantit la précision, l'adaptabilité et un provisionnement contextuel, permettant à la plateforme de répondre dynamiquement aux changements de charge de travail, d'optimiser l'allocation des ressources et d'améliorer l'efficacité opérationnelle.

# Perspectives

Bien que cette recherche ait démontré la faisabilité et l'efficacité de l'architecture proposée, elle met également en évidence plusieurs opportunités d'exploration et d'innovation futures :

- Améliorations de la sécurité : Les considérations de sécurité, telles que l'isolation réseau, la communication sécurisée des API et la gestion des identités, ont été intentionnellement simplifiées dans ce travail.
- L'intégration de mécanismes de sécurité avancés tels que les architectures zero-

trust, la communication chiffrée entre les agents et le contrôle d'accès basé sur les rôles (RBAC) pour les environnements multi-locataires pourrait encore renforcer la robustesse du système.

- Intégration de l'IA générative : Bien que le SMA s'appuie sur des modèles d'IA avancés pour le provisionnement, les itérations futures pourraient intégrer plus profondément l'IA générative dans les tâches de prédiction de charge de travail et d'optimisation des ressources. Les modèles génératifs pourraient enrichir le système en générant des traces de charge de travail synthétiques, en améliorant la prévision de la demande et en identifiant des stratégies de mise à l'échelle optimales.
- Extensibilité Hybride et Multi-Cloud : L'architecture se concentre actuellement sur un unique environnement de cloud privé hyperconvergé. Étendre ses capacités pour orchestrer des infrastructures hybrides ou multi-cloud permettrait aux organisations d'équilibrer les coûts, les performances et la conformité en déplaçant dynamiquement les charges de travail entre les clouds publics et privés.

En conclusion, cette thèse a jeté les bases d'une infrastructure cloud hautement automatisée, intelligente et résiliente. En relevant les défis décrits et en poursuivant les pistes futures suggérées, cette architecture a le potentiel d'évoluer vers une solution complète pour la gestion des environnements cloud de nouvelle génération, permettant aux organisations de répondre aux exigences de charges de travail de plus en plus complexes, dynamiques et distribuées.

# Bibliographie

- [1] Containerized computing evolution. <https://clouddocs.f5.com/training/community/nginx/html/class12/intro.html>. Accessed : 2025-03.
- [2] Sujith Surendran. Type 1 and type 2 hypervisor. <http://vgyan.in/type-1-and-type-2-hypervisor/>, 2014. Accessed : 2025-02.
- [3] SR Shishira, A Kandasamy, and K Chandrasekaran. Survey on meta heuristic optimization techniques in cloud computing. In *2016 international conference on advances in computing, communications and informatics (ICACCI)*, pages 1434–1440. IEEE, 2016.
- [4] Ian Smalley Stephanie Susnjara. What is virtualization ? <https://www.ibm.com/think/topics/virtualization>, 2025. Accessed : 2025-02.
- [5] Ian Smalley Stephanie Susnjara. What are containers ? <https://www.ibm.com/think/topics/containers>, 2024. Accessed : 2025-04.
- [6] Awodele Oludele, Emmanuel C. Ogu, Kuyoro Shade, and Umezuruike Chinecherem. On the evolution of virtualization and cloud computing : A review. 2014.
- [7] Containers vs. virtual machines (vms). <https://www.redhat.com/en/topics/containers/containers-vs-vms>, 2023. Accessed : 2025-04-15.
- [8] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4) :50–58, 2010.
- [9] Shahir Daya, Nguyen Van Duy, Kameswara Eati, Carlos M Ferreira, Dejan Glozic, Vasfi Gucer, Manav Gupta, Sunil Joshi, Valerie Lampkin, Marcelo Martins, et al. *Microservices from theory to practice : creating applications in IBM Bluemix using the microservices approach*. IBM Redbooks, 2016.
- [10] Kelsey Hightower, Brendan Burns, and Joe Beda. *Kubernetes : Up and Running : Dive into the Future of Infrastructure*. O'Reilly Media, Sebastopol, CA, 2017.

## Bibliographie

---

- [11] Tania Lorido-Botran, Jose Miguel-Alonso, and Juan A Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 12(4) :559–592, 2014.
- [12] Sadeka Islam, Jacky Wai Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Gener. Comput. Syst.*, 28 :155–162, 2012.
- [13] Zijun Li, Linsong Guo, Jiagan Cheng, Quan Chen, Bingsheng He, and Minyi Guo. The serverless computing survey : A technical primer for design architecture. *arXiv preprint arXiv :2112.12921*, 2021. Decouples cloud architecture into virtualization, encapsulation, orchestration, and coordination layers.
- [14] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2005.
- [15] Ian Smalley Stephanie Susnjara. What are hypervisors ? <https://www.ibm.com/think/topics/hypervisors>, October 2024. Consulted on february 2025.
- [16] what-is-a-hypervisor. <https://www.redhat.com/en/topics/virtualization/what-is-a-hypervisor>, 2023.
- [17] What is a hypervisors ? <https://www.vmware.com/topics/hypervisor>. Consulted on february 2025.
- [18] Dirk Merkel. Docker : Lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239) :2, 2014.
- [19] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *Queue*, 14(1) :70–93, 2016.
- [20] Tania Lorido-Botran, Jose Miguel-Alonso, and Juan A Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *ACM Computing Surveys*, 46(1) :6, 2014.
- [21] Saleha Alharthi, Afra Alshamsi, Anoud Alseiari, and Abdulmalik Alwarafy. Auto-scaling techniques in cloud computing : Issues and research directions. *Sensors*, 24(17) :5551, 2024.
- [22] David Buchaca Prats, Josep Lluís Berral, Chen Wang, and Alaa Youssef. Proactive container auto-scaling for cloud native machine learning services. In *Proceedings of the 2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 475–479, 2020.

## Bibliographie

---

- [23] Vedran Dakić, Goran Đambić, Jurica Slovinac, and Jasmin Redžepagić. Optimizing kubernetes scheduling for web applications using machine learning. *Electronics*, 14(5) :863, 2025.
- [24] Run a large-scale workload with flex-start with queued provisioning. <https://cloud.google.com/kubernetes-engine/docs/how-to/provisioningrequest>, 2024. Accessed : 2025-05-23.
- [25] Jooyoung Kim, Abhishek Nautiyal, and Ankur Sethi. Optimize compute resources on amazon ecs with predictive scaling. <https://aws.amazon.com/blogs/containers/optimize-compute-resources-on-amazon-ecs-with-predictive-scaling/>, 2024. Accessed : 2025-05-23.
- [26] Luyao Pei, Cheng Xu, Xueli Yin, and Jinsong Zhang. Multi-agent deep reinforcement learning for cloud-based digital twins in power grid management. *Journal of Cloud Computing*, 13(1) :152, 2024.
- [27] Long Wang, Anh V. Ngu, Songqing Chen, Xiang Bai, and Manish Parashar. A dynamic resource allocation approach for scientific workflows in cloud computing. *Future Generation Computer Systems*, 27(5) :618–628, 2011.
- [28] Rokaia Rokaia. Metaheuristic optimization for scheduling in cloud computing environments : A review. *Metaheuristic Optimization Review*, 2024.
- [29] Anindya Bose, Sanjay Nag, et al. An overview of the state-of-the-art virtual machine placement algorithms for green cloud data centres. *Advancement in Management and Technology (AMT)*, 3(1) :1–12, 2022.
- [30] S Manikandan and M Chinnadurai. Virtualized load balancer for hybrid cloud using genetic algorithm. *Intell Autom Soft Comput*, 32(3) :1459–1466, 2022.
- [31] Xiong Fu, Qing Zhao, Junchang Wang, Lin Zhang, and Lei Qiao. Energy-aware vm initial placement strategy based on bpsos in cloud computing. *Scientific Programming*, 2018(1) :9471356, 2018.
- [32] L. Wang, G. von Laszewski, A. Younge, X. He, and M. Kunze. Energy-aware virtual machine consolidation for cloud data centers. *Proceedings of IEEE International Conference on Cloud Computing Technology and Science*, pages 102–109, 2013.
- [33] Z. Xiong and J. Xu. Towards energy-efficient cloud resource provisioning and scheduling using container-based virtualization. In *2014 IEEE International Conference on Cloud Computing*, pages 275–282, 2014.

## Bibliographie

---

- [34] A. Beloglazov and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5) :755–768, 2012.
- [35] K. Ferdaus, S. Abdelwahed, S. Guica, and E. Bohrer. Ant colony optimization for energy-aware virtual machine placement in data centers. *Journal of Network and Computer Applications*, 45 :117–128, 2014.
- [36] A. Subramoney and P. Nyirenda. Comparative evaluation of ga–pso hybrid for workflow scheduling in cloud computing. *arXiv preprint arXiv :2012.00176*, 2020. Accessed : 2025-05-24.
- [37] R.N. Calheiros, C.W. Tsai, C.A.F. de Rose, and M.A.S. Netto. A hybrid genetic algorithm and particle swarm optimization for workflow scheduling in cloud environments. *Future Generation Computer Systems*, 52 :242–256, 2015.
- [38] Said Nabi, Masroor Ahmad, Muhammad Ibrahim, and Habib Hamam. Adpsos : adaptive pso-based task scheduling approach for cloud computing. *Sensors*, 22(3) :920, 2022.
- [39] L. Zhang and H. Chen. Binary particle swarm optimization for initial virtual machine placement in cloud computing. *Mathematical Problems in Engineering*, 2018 :9471356, 2018. Accessed : 2025-05-24.
- [40] P. Mohammadzadeh, S. Rahati, and A. Mosavi. Chaotic hybrid multi-objective optimization algorithm for scientific workflow scheduling in multisite clouds. *Applied Soft Computing*, 125 :109466, 2023.
- [41] ASTR Journal. Multi-resource particle swarm optimization for vm allocation in openstack, 2014. Accessed : 2025-05-23.
- [42] Mohit Kumar, Subhash Chander Sharma, Shalini Goel, Sambit Kumar Mishra, and Akhtar Husain. Autonomic cloud resource provisioning and scheduling using meta-heuristic algorithm. *Neural Computing and Applications*, 32 :18285–18303, 2020.
- [43] Fahd N. Al-Wesabi, Marwa Obayya, Manar Ahmed Hamza, Jaber S. Alzahrani, Deepak Gupta, and Sachin Kumar. Energy aware resource optimization using unified metaheuristic optimization algorithm allocation for cloud computing environment. *Sustainable Computing : Informatics and Systems*, 35 :100686, 2022.
- [44] Syed Hamid Hussain Madni, Muhammed Faheem, Muhammad Younas, Maidul Hasan Masum, and Sajid Shah. Critical review on resource scheduling in iaas clouds : Taxonomy, issues, challenges, and future directions. *The Journal of Engineering*, 2024.

- [45] Adnan Abid, Muhammad Faraz Manzoor, Muhammad Shoaib Farooq, Uzma Farooq, and Muzammil Hussain. Challenges and issues of resource allocation techniques in cloud computing. *KSII Trans. Internet Inf. Syst.*, 14 :2815–2839, 2020.
- [46] Min Cao, Yaoyu Li, Xupeng Wen, Yue Zhao, and Jianghan Zhu. Energy-aware intelligent scheduling for deadline-constrained workflows in sustainable cloud computing. *Egyptian Informatics Journal*, 2023.
- [47] Mohammad Masdari and Afsane Khoshnevis. A survey and classification of the workload forecasting methods in cloud computing. *Cluster Computing*, 23 :2399 – 2424, 2019.
- [48] Jitendra Kumar, Rimsha Goomer, and Ashutosh Kumar Singh. Long short term memory recurrent neural network (lstm-rnn) based workload forecasting model for cloud datacenters. *Procedia Computer Science*, 125 :676–682, 2018.
- [49] Yonghua Zhu, Weilin Zhang, Yihai Chen, and Honghao Gao. A novel approach to workload prediction using attention-based lstm encoder-decoder network in cloud environment. *EURASIP Journal on Wireless Communications and Networking*, 2019, 2019.
- [50] Jing Bi, Shuang Li, Haitao Yuan, and Mengchu Zhou. Integrated deep learning method for workload and resource prediction in cloud systems. *Neurocomputing*, 424 :35–48, 2020.
- [51] Jitendra Kumar and Ashutosh Kumar Singh. Workload prediction in cloud using artificial neural network and adaptive differential evolution. *Future Gener. Comput. Syst.*, 81 :41–52, 2018.
- [52] Sumit Kumar. Statistical vs machine learning vs deep learning modeling for time series forecasting, December 2022. Accessed : 2025-05-23.
- [53] Martín Solís and Luis Alexánder Calvo-Valverde. Explaining when deep learning models are better for time series forecasting. *ITISE 2024*, 2024.
- [54] Angelo Casolaro, Vincenzo Capone, Gennaro Iannuzzo, and Francesco Camastrà. Deep learning for time series forecasting : Advances and open problems. *Inf.*, 14 :598, 2023.
- [55] Shereen Elsayed, Daniela Thyssens, Ahmed Rashed, Lars Schmidt-Thieme, and Hadi Samer Jomaa. Do we really need deep learning models for time series forecasting? *ArXiv*, abs/2101.02118, 2021.

## Bibliographie

---

- [56] Zahra Fatemi, Minh-Thu T. Huynh, Elena Zheleva, Zamir Syed, and Xiaojun Di. Mitigating cold-start forecasting using cold causal demand forecasting model. *ArXiv*, abs/2306.09261, 2023.
- [57] Andrea Rossi, Andrea Visentin, Diego Carraro, Steven D. Prestwich, and Kenneth N. Brown. Forecasting workload in cloud computing : towards uncertainty-aware predictions and transfer learning. *Clust. Comput.*, 28 :258, 2025.
- [58] Aya I Maiyza, Noha O Korany, Karim Banawan, Hanan A Hassan, and Walaa M Sheta. Vtgan : hybrid generative adversarial networks for cloud workload prediction. *Journal of Cloud Computing*, 12(1) :97, 2023.
- [59] Weiwei Lin, Kun Yao, Lan Zeng, Fagui Liu, Chun Shan, and Xiaobin Hong. A gan-based method for time-dependent cloud workload generation. *Journal of Parallel and Distributed Computing*, 168 :33–44, 2022.
- [60] Raymond Ang, Wai Kit Lau, and Khang Tai Le. Integrating large language models with splunk to augment enterprise logs workflows, December 8 2024. Accessed : 2025-05-24.
- [61] Shivani Arbat, Vinodh Kumaran Jayakumar, Jaewoo Lee, Wei Wang, and In Kee Kim. Wasserstein adversarial transformer for cloud workload prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 12433–12439, 2022.
- [62] Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. Cilp : Co-simulation based imitation learner for dynamic resource provisioning in cloud computing environments, 2023.
- [63] Shreshth Tuli, Giuliano Casale, and Nicholas R Jennings. Cilp : Co-simulation-based imitation learner for dynamic resource provisioning in cloud computing environments. *IEEE Transactions on Network and Service Management*, 20(4) :4448–4460, 2023.
- [64] Table 4 : Comparative analysis of load balancing techniques. <https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-023-00473-z/tables/4>, 2023. Accessed : 2025-05.
- [65] Amazon Web Services, Inc. Using amazon bedrock agents to interactively generate infrastructure as code, 2024. Accessed : 2025-05-24.
- [66] Amazon Web Services, Inc. Top analytics announcements of aws re :invent 2024, 2024. Accessed : 2025-05-24.

## Bibliographie

---

- [67] Amazon Web Services, Inc. amazon-bedrock-rag-knowledgebases-agents-cloudformation. <https://github.com/aws-samples/amazon-bedrock-rag-knowledgebases-agents-cloudformation>, 2024. Accessed : 2025-05-24.
- [68] Amazon Web Services, Inc. Aws announces support for predictive scaling for amazon ecs services, 2024. Accessed : 2025-05-02.
- [69] Célia Ghedini Ralha, Aldo HD Mendes, Luiz A Laranjeira, Aletéia PF Araújo, and Alba CMA Melo. Multiagent system for dynamic resource provisioning in cloud computing platforms. *Future generation computer systems*, 94 :80–96, 2019.
- [70] Ali Belgacem, Saïd Mahmoudi, and Maria Kihl. Intelligent multi-agent reinforcement learning model for resources allocation in cloud computing. *Journal of King Saud University-Computer and Information Sciences*, 34(6) :2391–2404, 2022.
- [71] J Octavio Gutierrez-Garcia and Kwang Mong Sim. Agents for cloud resource allocation : An amazon ec2 case study. In *International Conference on Grid and Distributed Computing*, pages 544–553. Springer, 2011.
- [72] Tong Cheng, Hang Dong, Lu Wang, Bo Qiao, Si Qin, Qingwei Lin, Dongmei Zhang, Saravan Rajmohan, and Thomas Moscibroda. Multi-agent reinforcement learning with shared policy for cloud quota management problem. In *Companion Proceedings of the ACM Web Conference 2023*, pages 391–395, 2023.
- [73] Xiangqiang Gao, Rongke Liu, and Aryan Kaushik. Hierarchical multi-agent optimization for resource allocation in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 32(3) :692–707, 2021.
- [74] Sofiane Kemchi, Abdelhafid Zitouni, and Mahieddine Djoudi. Self organization agent oriented dynamic resource allocation on open federated clouds environment. *ArXiv*, abs/2001.07496, 2016.
- [75] Fouad Jowda and Muntasir Al-Asfoor. A multi-agent system framework for cloud resources allocation. *Journal of Al-Qadisiyah for computer science and mathematics*, 13(3) :Page–78, 2021.

---

## **Annexes**

# Annexe A

## PROXMOX

### A.1 Partitionnement d'un disque Proxmox en CLI (Post-Installation)

Cette subsection explique comment partitionner un disque dans un environnement Proxmox VE à l'aide des outils en ligne de commande après l'installation initiale. Ceci est particulièrement utile lorsque vous avez intentionnellement réservé de l'espace libre sur le disque principal pour des configurations de stockage personnalisées.

#### A.1.1 Note de pré-installation

Pendant l'installation de Proxmox VE, il est fortement recommandé d'utiliser l'option d'installation `hdsize`. Cette option définit la quantité du disque principal qui doit être allouée au système Proxmox (spécifiquement, au groupe de volumes LVM `pve`), garantissant que de l'espace libre reste non alloué pour une utilisation ultérieure.

## Annexe A. PROXMOX

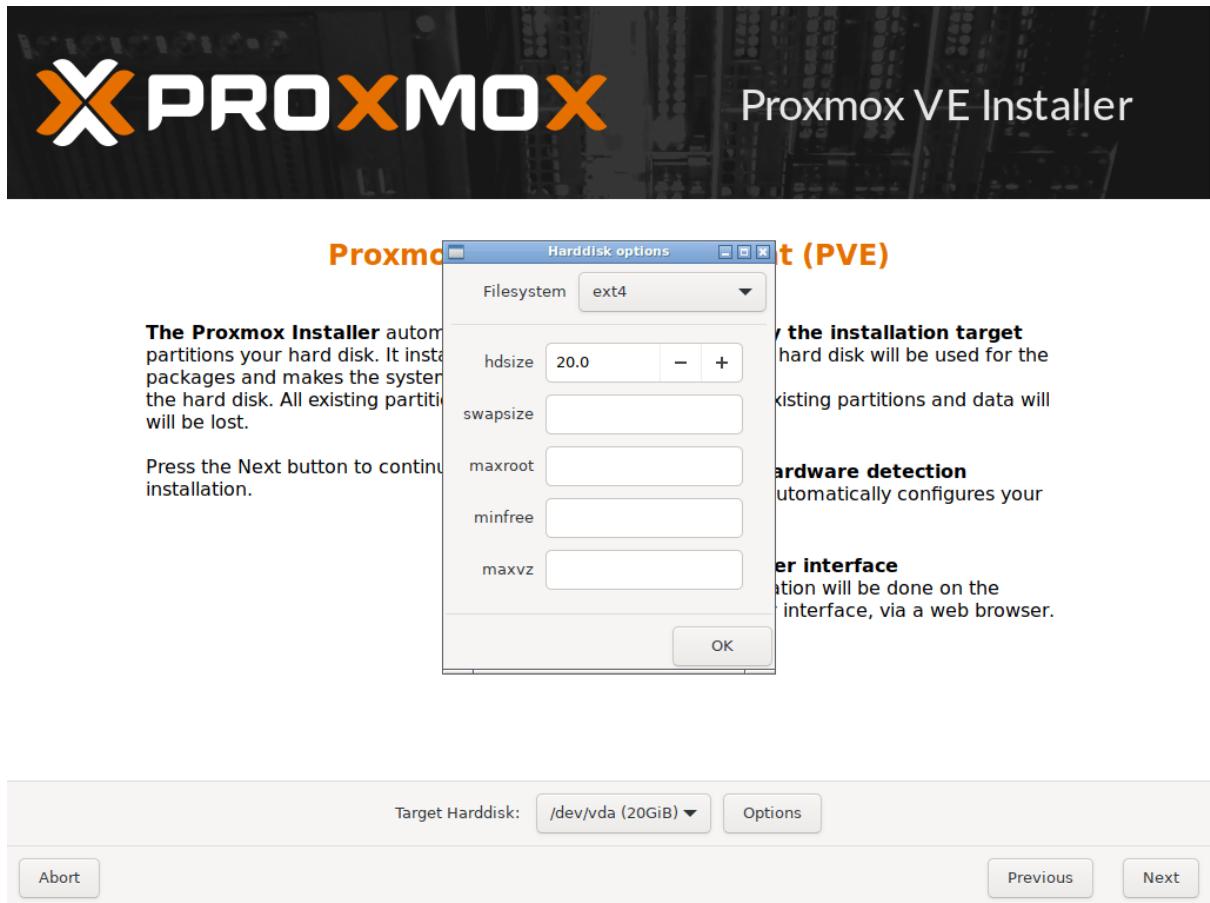


FIGURE A.1 – Définition de l’option `hdszie` dans l’installateur Proxmox.

Par exemple, pour limiter l’installation principale de Proxmox à 100 Go, vous spécifierez :

```
hdszie=100G
```

Cette commande crée une partition système Proxmox de 100 Go et laisse le reste de l’espace disque non formaté et disponible pour un partitionnement manuel.

### A.1.2 Structure initiale du disque

Après avoir installé Proxmox avec une option comme `hdszie=100G` sur un disque de 1 To, la structure initiale du disque ressemblera à ce qui suit lors de l’inspection avec `lsblk`. Notez l’espace non alloué sur `sda` qui ne fait partie d’aucune partition.

```
sda              8:0    0  953.9G  0 disk
| -sda1           8:1    0   1007K  0 part
| -sda2           8:2    0      1G  0 part /boot/efi
`-sda3           8:3    0    99G  0 part
```

## Annexe A. PROXMOX

---

```
| -pve-swap      252:0    0     8G  0 lvm  [SWAP]  
| -pve-root      252:1    0   34.7G  0 lvm  /  
| -pve-data_tmeta 252:2    0     1G  0 lvm  
| ‘-pve-data     252:4    0    42G  0 lvm  
‘-pve-data_tdata 252:3    0    42G  0 lvm  
‘-pve-data     252:4    0    42G  0 lvm
```

### A.1.3 Repartitionnement avec gdisk

Pour utiliser les ~850 Go d'espace disque libre restants, nous allons créer une nouvelle partition, `sda4`, à l'aide de l'utilitaire `gdisk`.

#### Étape 1 : Lancer gdisk

Ouvrez l'outil de partitionnement de disque GPT pour le disque cible.

```
gdisk /dev/sda
```

#### Étape 2 : Créer une nouvelle partition

Suivez les invites interactives pour créer une nouvelle partition. Utilisez les valeurs par défaut pour allouer tout l'espace libre restant à cette nouvelle partition.

```
Command (? for help): n  
Partition number (4-128, default 4): 4  
First sector: (press Enter)  
Last sector: (Press Enter to use the remaining free space)  
Hex code or GUID (L to show codes, Enter = 8300): (press Enter)
```

Le code hexadécimal 8300 correspond à un type de partition standard "Linux filesystem".

#### Étape 3 : Afficher et vérifier la structure

Utilisez la commande `p` pour afficher la nouvelle table de partition et vérifier qu'elle correspond à vos intentions avant d'écrire les modifications sur le disque.

```
Command (? for help): p
```

La sortie devrait maintenant inclure la nouvelle partition, `sda4`.

## Annexe A. PROXMOX

---

Number	Start (sector)	End (sector)	Size	Code	Name
1	34	2047	1007.0 KiB	EF02	
2	2048	2099199	1024.0 MiB	EF00	
3	2099200	209715200	99.0 GiB	8E00	Linux LVM
4	209715208	1998585863	853.9 GiB	8300	Linux filesystem

### Étape 4 : Écrire la table de partition sur le disque

Utilisez la commande `w` pour enregistrer les modifications. Il s'agit d'une opération destructive, procédez donc avec prudence.

```
Command (? for help): w
```

```
Final checks complete. About to write GPT data. THIS WILL OVERWRITE
EXISTING PARTITIONS!!
```

```
Do you want to proceed? (Y/N): y
```

Confirmez avec `Y` pour écrire la nouvelle table de partition.

### Étape 5 : Redémarrer le système

Un redémarrage est nécessaire pour que le noyau reconnaisse la nouvelle table de partition.

```
reboot
```

#### A.1.4 Vérification après redémarrage

Une fois le système redémarré, reconnectez-vous et exécutez à nouveau `lsblk` pour vérifier que la nouvelle partition est visible par le système d'exploitation.

```
lsblk
```

Vous devriez maintenant voir la nouvelle partition (`sda4` dans cet exemple, bien que le numéro puisse varier) listée.

```
sda                  8:0      0 953.9G  0 disk
| -sda1                8:1      0 1007K  0 part
| -sda2                8:2      0     1G  0 part /boot/efi
```

```
| -sda3              8:3    0    99G  0  part
| |-pve-swap        252:0  0     8G  0  lvm   [SWAP]
| |-pve-root        252:1  0   34.7G  0  lvm   /
| |-pve-data_tmeta 252:2  0     1G  0  lvm
| | '-pve-data     252:4  0   42G  0  lvm
| |-pve-data_tdata 252:3  0   42G  0  lvm
| | '-pve-data     252:4  0   42G  0  lvm
`-sda4              8:4    0  853.9G  0  part
```

La nouvelle partition `/dev/sda4` est maintenant prête à être formatée avec un système de fichiers (par exemple, ext4, xfs) ou à être utilisée comme volume physique pour un nouveau groupe LVM, un pool ZFS, etc.

## A.2 Création d'un template de machine virtuelle dans Proxmox

Ce guide vous guidera à travers les étapes pour créer un template réutilisable pour les machines virtuelles (VMs) dans Proxmox en utilisant l'image Cloud d'Ubuntu. Les templates vous permettent de déployer rapidement de nouvelles VMs avec des paramètres pré-configurés, ce qui permet de gagner du temps et d'assurer la cohérence. Ce chapitre couvre à la fois le processus manuel étape par étape et une méthode entièrement automatisée basée sur un script.

### A.2.1 Prérequis

Avant de continuer, assurez-vous de ce qui suit :

- Vous disposez d'un environnement Proxmox VE configuré.
- Vous avez téléchargé l'image Cloud d'Ubuntu depuis [Ubuntu Cloud Images](#) et l'avez téléversée sur votre stockage ISO Proxmox.
- Un stockage et des ressources suffisants sont disponibles sur votre nœud Proxmox.

### A.2.2 Création manuelle du template (GUI & CLI)

Cette subsection détaille le processus manuel de création d'un template en utilisant une combinaison de l'interface web de Proxmox et de la ligne de commande.

## Annexe A. PROXMOX

### Étape 1 : Téléverser l'image Cloud

- Connectez-vous à l'interface web de Proxmox.
- Naviguez vers Datacenter > pmox01 > local > ISO Images.
- Pour téléverser l'image Cloud d'Ubuntu, vous avez deux options :
  1. Si l'image est déjà téléchargée sur votre machine locale, cliquez sur le bouton Upload et sélectionnez le fichier.
  2. Alternativement, téléchargez l'image directement depuis le web en cliquant sur Download from URL, en saisissant l'URL <https://cloud-images.ubuntu.com/noble/current/> en fournissant un nom pour l'image, et en cliquant sur Download.

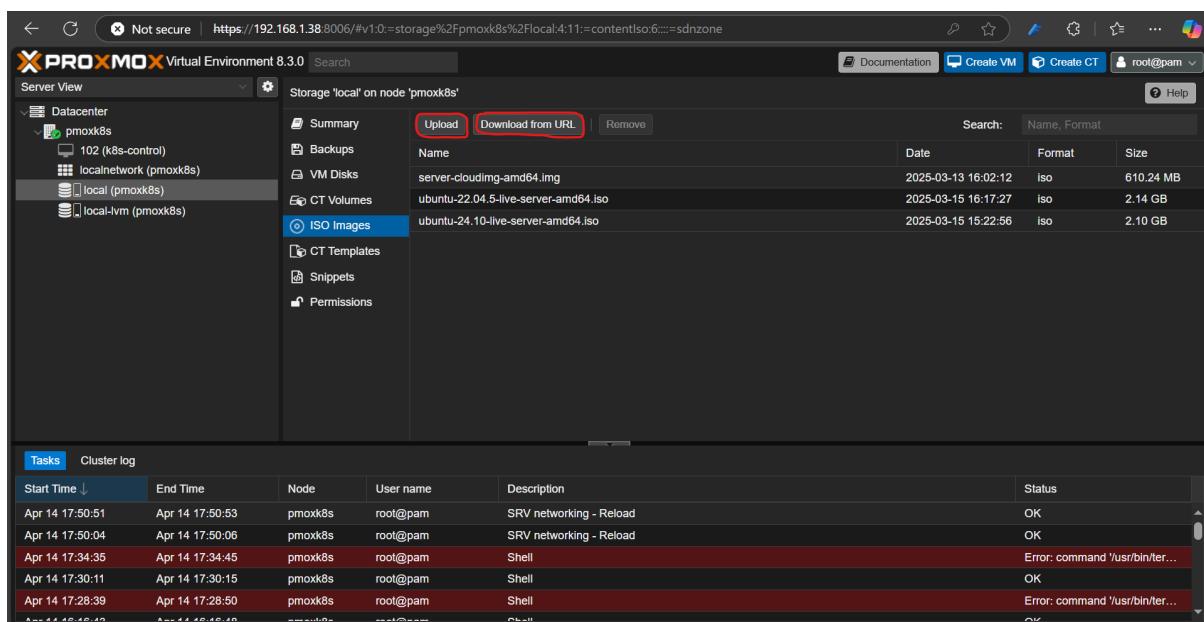


FIGURE A.2 – Téléversement de l'image Cloud Ubuntu.

### Étape 2 : Créer et configurer la VM

Il existe deux alternatives pour créer la VM de base : utiliser l'interface graphique ou la ligne de commande.

#### Alternative 1 : Utilisation de la GUI

- Sous le nœud pmox01, cliquez sur le bouton Create VM.
- **Général** : Donnez un nom et un ID au template.
- **OS** : Sélectionnez Do not use any media.
- **Système** : Cochez la case Qemu Agent.

## Annexe A. PROXMOX

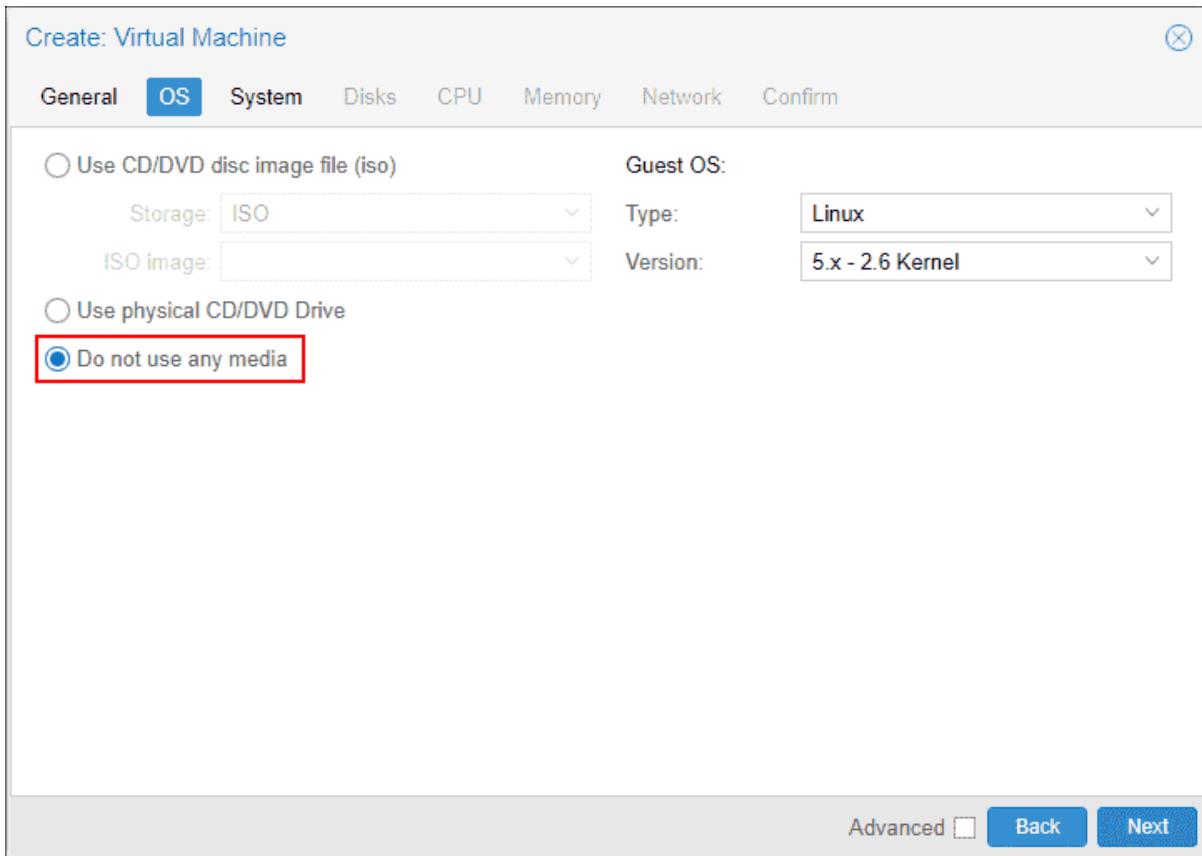


FIGURE A.3 – Créer une VM sans média ISO.

- **Disques** : Supprimez le disque dur par défaut en le sélectionnant et en cliquant sur l'icône de la corbeille. Nous importerons l'image cloud manuellement.
- **CPU, Mémoire, Réseau** : Configurez-les comme vous le souhaitez, ou conservez les valeurs par défaut.
- Terminez l'assistant pour créer la VM.

### Alternative 2 : Utilisation de la CLI

Ces commandes effectuent les mêmes actions que les étapes de l'interface graphique, ainsi que la configuration ultérieure du disque et du matériel.

- Créez la VM de base :

```
qm create 100 --name ubuntu-template --memory 2048 --cores 2 --net0 virtio,bridge
```

- Définissez les options matérielles nécessaires (agent QEMU, lecteur Cloud-Init, console série) :

```
qm set 100 --agent enabled=1
```

## Annexe A. PROXMOX

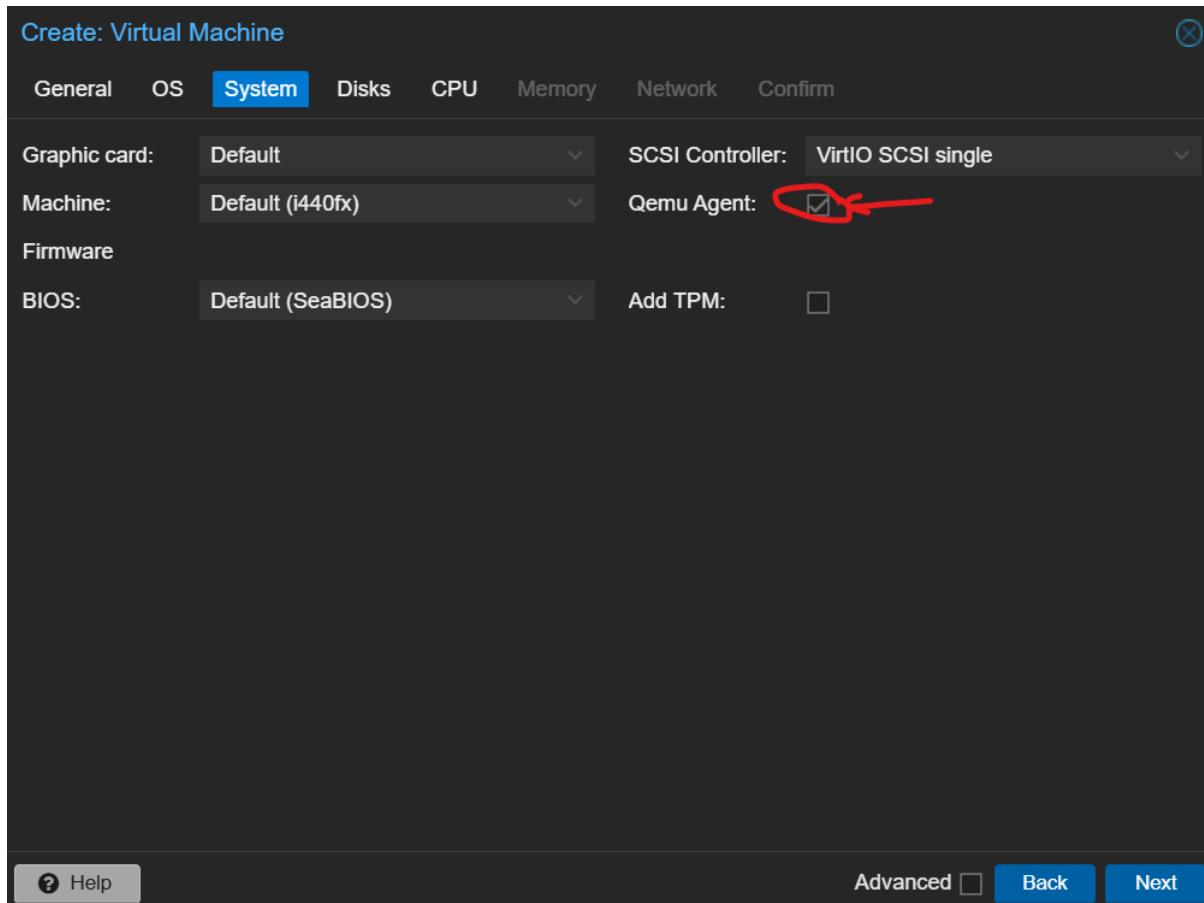


FIGURE A.4 – Activer l’agent QEMU dans la fenêtre Système.

```
qm set 100 --ide2 local-lvm:cloudinit  
qm set 100 --serial0 socket --vga serial0
```

- Copiez l’image cloud depuis le stockage ISO, redimensionnez-la et importez-la en tant que disque pour la VM (remplacez les noms d’image/disque et l’ID de la VM si nécessaire) :

```
# Copier l’image du stockage vers un répertoire de travail  
cp /var/lib/vz/template/iso/noble-server-cloudimg-amd64.img /root/noble.qcow2  
  
# Redimensionner l’image  
qemu-img resize /root/noble.qcow2 32G  
  
# Importer l’image redimensionnée comme un disque pour la VM  
qm importdisk 100 /root/noble.qcow2 local-lvm
```

- Attachez le nouveau disque et définissez l’ordre de démarrage :

## Annexe A. PROXMOX

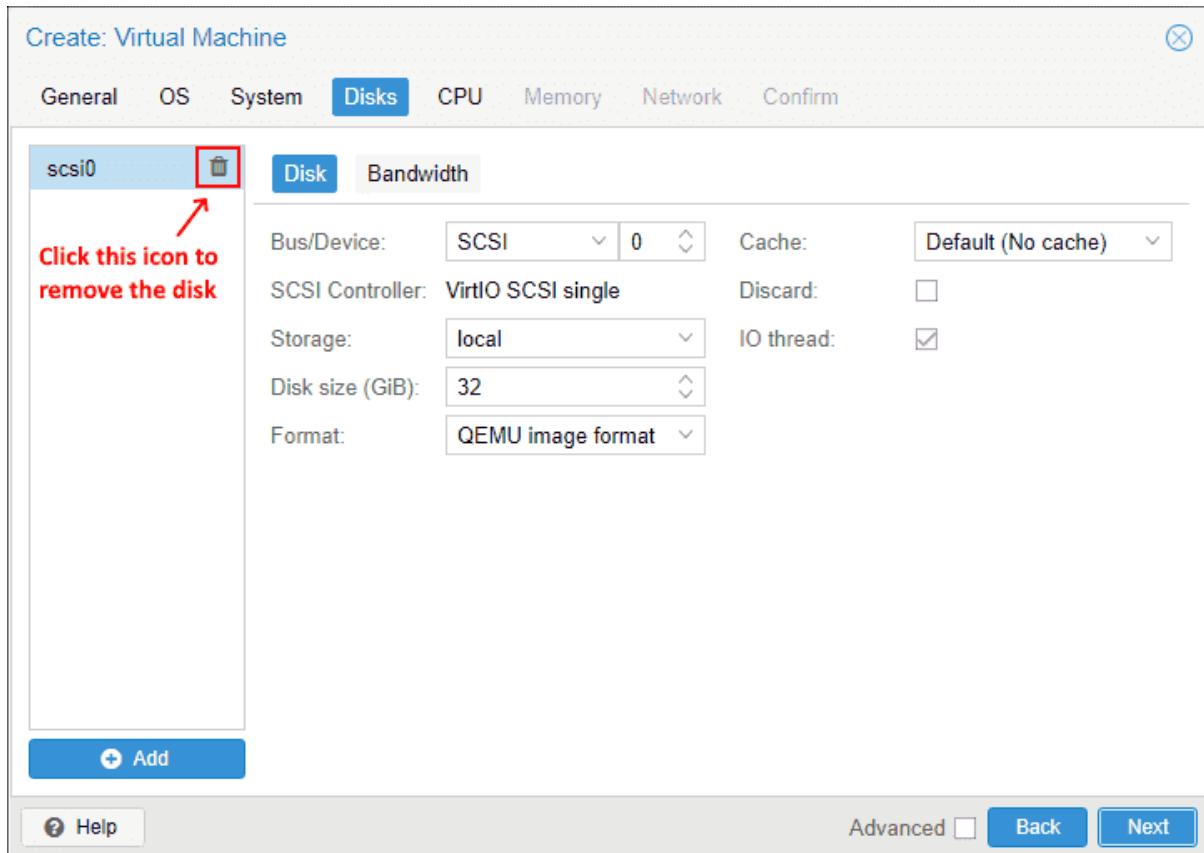


FIGURE A.5 – Supprimer le disque dur par défaut de la VM.

```
# Attacher le disque en tant que lecteur SCSI
qm set 100 --scsihw virtio-scsi-pci --scsi0 local-lvm:vm-100-disk-0

# Définir l'ordre de démarrage pour prioriser le lecteur Cloud-Init, puis le nouveau
qm set 100 --boot order="ide2;scsi0;net0" --bootdisk scsi0
```

### Étape 3 : Personnaliser l'image Cloud

- Dans l'interface utilisateur de Proxmox, sélectionnez la nouvelle VM et allez dans le panneau Cloud-Init.
- Configurez l'utilisateur, le mot de passe, les clés SSH et les paramètres réseau comme vous le souhaitez.
- Cliquez sur Regenerate Image pour appliquer les paramètres.

### Étape 4 : Préparer la VM pour la conversion en template (Optionnel)

Si vous avez besoin de logiciels ou de mises à jour supplémentaires dans votre template de base, vous pouvez effectuer ces étapes :

## Annexe A. PROXMOX

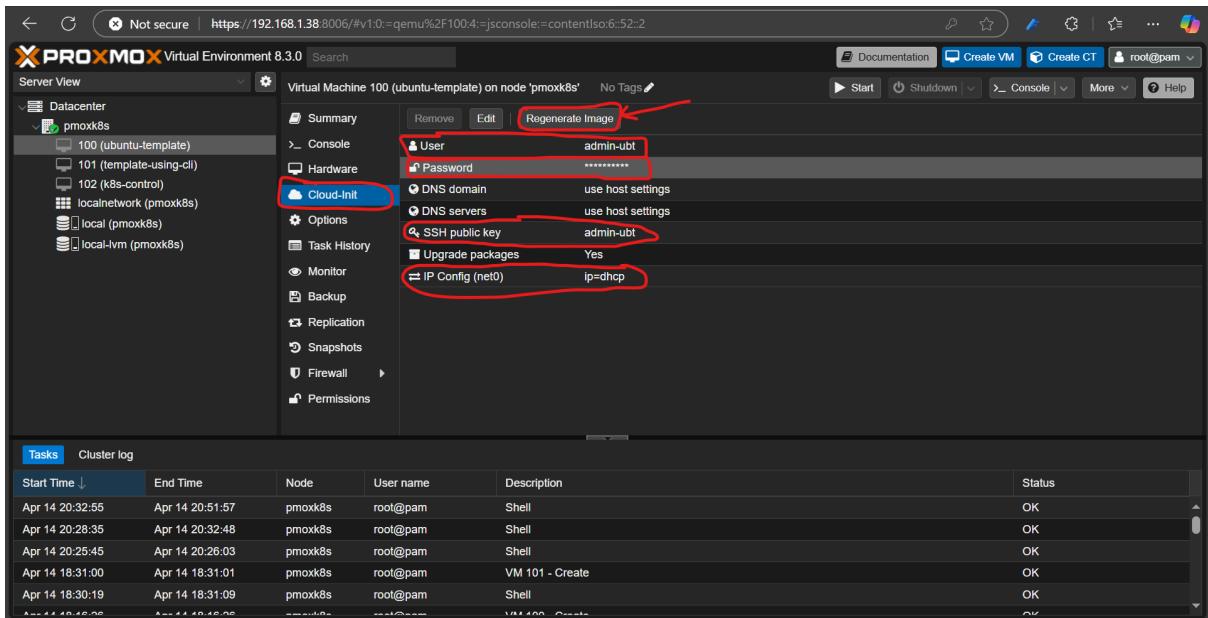


FIGURE A.6 – Personnaliser les paramètres de Cloud-Init.

- Démarrez la VM pour la première fois.
- Connectez-vous et installez toutes les mises à jour et logiciels nécessaires (par exemple, `apt update && apt upgrade -y`).
- Nettoyez les fichiers temporaires et effacez les journaux pour minimiser la taille du template.
- Éteignez proprement la VM.

### Étape 5 : Convertir la VM en template

Une fois la VM entièrement configurée et arrêtée, convertissez-la en template.

- Dans l’interface web de Proxmox, faites un clic droit sur la VM et sélectionnez **Convert to Template**.
- Alternativement, utilisez la CLI (remplacez 100 par l’ID de votre VM) :

```
qm template 100
```

### A.2.3 Crédit automatisée du template avec un script Bash

Comme alternative aux étapes manuelles, l’ensemble du processus peut être automatisé avec un seul script bash. C’est la méthode la plus efficace pour créer des templates de manière cohérente.

## Annexe A. PROXMOX

---

Tout d'abord, enregistrez le script suivant dans un fichier (par exemple, `create_template.sh`) sur votre hôte Proxmox.

**Note importante :** Vous devez supprimer tous les commentaires (lignes commençant par `#`) du script pour qu'il s'exécute correctement sans erreurs.

```
#!/bin/bash
# Ce script crée un template de VM pour Ubuntu Noble

# Variables
VM_ID=101
VM_NAME="ubuntu-template-using-bash"
CLOUD_IMAGE_URL_OFFLINE="/var/lib/vz/template/iso/noble-server-cloudimg-amd64.img"
CLOUD_IMAGE_PATH="/root/noble-server-cloudimg-amd64.qcow2"
CLOUD_IMAGE_PATH1="noble-server-cloudimg-amd64.qcow2"
DISK_SIZE="32G"
MEMORY=1024
CORES=1

# Étape 1 : Copier l'image
echo "Copie de l'image..."
cp "$CLOUD_IMAGE_URL_OFFLINE" "$CLOUD_IMAGE_PATH"

# Étape 2 : Redimensionner le disque
echo "Redimensionnement du disque..."
qemu-img resize "$CLOUD_IMAGE_PATH1" "$DISK_SIZE"

# Étape 3 : Créer la VM
echo "Création de la VM..."
qm create $VM_ID \
--name $VM_NAME \
--memory $MEMORY \
--cores $CORES \
--net0 virtio,bridge=vmbr0 \
--ide2 local-lvm:cloudinit \
--onboot 1 \
--agent 1

# Étape 4 : Importer le disque
echo "Importation du disque..."
```

## Annexe A. PROXMOX

---

```
qm importdisk $VM_ID "$CLOUD_IMAGE_PATH1" local-lvm

# Étape 5 : Attacher le disque et configurer le matériel
echo "Attachement du disque et configuration du matériel..."
qm set $VM_ID --scsihw virtio-scsi-pci --scsi0 local-lvm:vm-$VM_ID-disk-0
qm set $VM_ID --serial0 socket --vga serial0
qm set $VM_ID --boot order="ide2;scsi0;net0" --bootdisk scsi0

# Étape 6 : Configurer Cloud-Init
echo "Configuration de Cloud-Init..."
mkdir -p /root/.ssh
chmod 700 /root/.ssh
touch ~/.ssh/id_ed25519.pub
echo "ssh-ed25519 AAAAC3...[votre-cle-publique]... admin-ubt" > ~/.ssh/id_ed25519.pub

qm set $VM_ID \
    --ciuser admin-ubt \
    --cipassword admin \
    --sshkey ~/.ssh/id_ed25519.pub \
    --ipconfig0 ip=dhcp

# Étape 7 : Convertir la VM en template
echo "Conversion de la VM en template..."
qm template $VM_ID

echo "Création du template de VM terminée avec succès !"
```

Après avoir enregistré le script (et supprimé les commentaires), rendez-le exécutable et lancez-le :

```
# Rendre le script exécutable
chmod +x create_template.sh

# Exécuter le script
./create_template.sh
```

La création de templates de VM, que ce soit par le processus manuel étape par étape ou en utilisant un script automatisé, est une pratique fondamentale dans Proxmox VE. Elle simplifie considérablement le déploiement de nouvelles machines virtuelles, garantissant la cohérence et économisant un temps

d'administration précieux. En maîtrisant ces techniques, les administrateurs peuvent maintenir un environnement de virtualisation plus efficace, évolutif et gérable.

### A.3 Guide de configuration du cluster Ceph

Cette subsection fournit un guide étape par étape de la configuration du cluster Ceph effectuée via l'interface web de Proxmox.

#### A.3.1 Étape 1 : Initialisation du cluster et quorum des moniteurs

Le processus a commencé par l'installation des paquets Ceph nécessaires sur chaque noeud Proxmox via l'interface utilisateur. Le cluster a ensuite été initialisé en créant le premier Moniteur (MON) sur le noeud pmox01 et en définissant le réseau public (172.25.5.0/24). Pour établir un quorum résilient et hautement disponible, des Moniteurs supplémentaires ont ensuite été créés sur les noeuds pmox02 et pmox03, comme le montre la Figure A.7. Cela garantit que le cluster peut maintenir un état opérationnel même si un noeud moniteur devient indisponible.

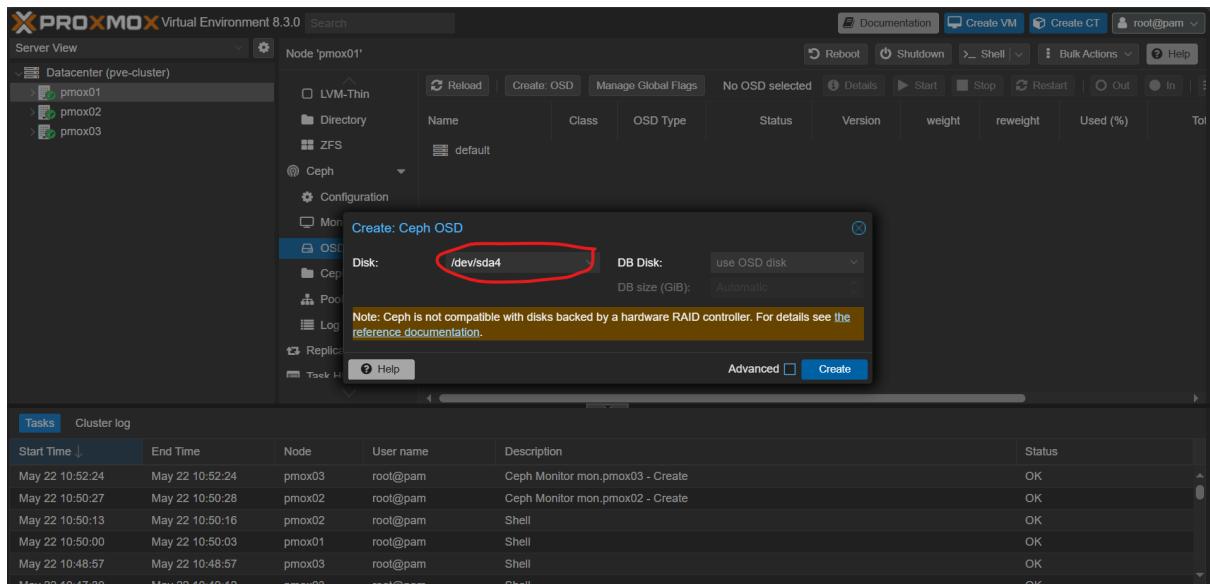


FIGURE A.7 – Crédit de moniteurs Ceph supplémentaires pour le quorum

### A.3.2 Étape 2 : Création des Daemons de Stockage d’Objets (OSDs)

Une fois le quorum des Moniteurs établi, l’étape suivante consistait à ajouter de la capacité de stockage au cluster. Sur chacun des trois hôtes Proxmox, la partition préparée `/dev/sda4` a été sélectionnée et désignée comme un Daemon de Stockage d’Objets (OSD). Cette action intègre le disque dans le cluster Ceph, rendant son stockage disponible pour la distribution et la réPLICATION à travers les nœuds (Figure A.8).

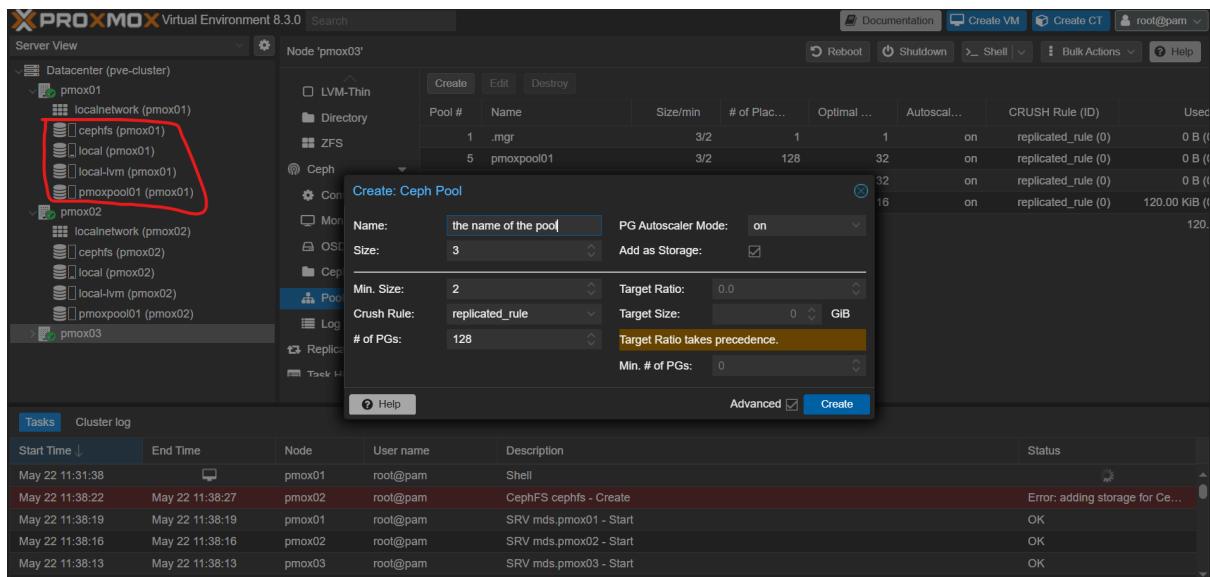


FIGURE A.8 – Création d’un Daemon de Stockage d’Objets (OSD) Ceph sur la partition préparée

### A.3.3 Étape 3 : Provisionnement des pools de stockage et des systèmes de fichiers

Avec les OSDs sous-jacents fournissant une structure de stockage unifiée, deux types distincts de stockage logique ont été provisionnés :

- **Pool de stockage bloc répliqué (RBD)** : Un pool répliqué nommé `pmoxpool01` a été créé pour fournir un stockage bloc résilient pour les disques des machines virtuelles. Comme le montre la Figure A.9, ce pool a été configuré avec une `size` de réPLICATION de trois et une `min_size` de deux. Cette configuration répond directement à l’objectif de rechercher d’assurer la résilience des données, car le système peut tolérer la défaillance complète d’un nœud sans perte de données ni interruption de service pour les disques de VM stockés.
- **Système de fichiers partagé (CephFS)** : Pour fournir un stockage de fichiers partagé accessible sur tous les nœuds, un système de fichiers CephFS nommé `cephfs`

## Annexe A. PROXMOX

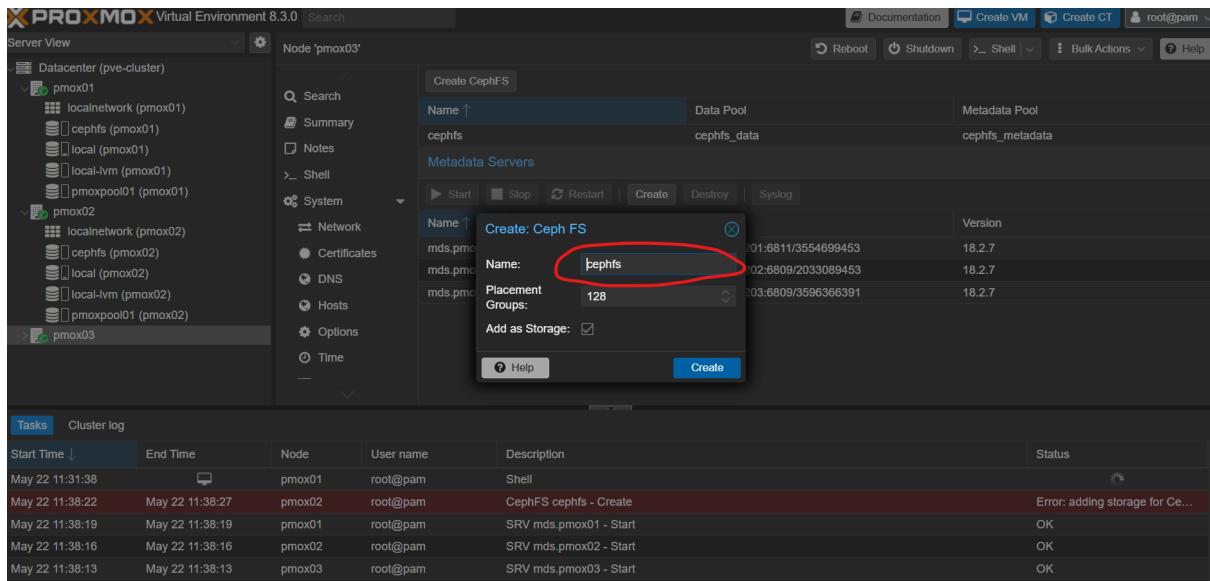


FIGURE A.9 – Création et configuration du pool répliqué 'pmoxpool01'

a été créé (Figure A.10). Ce processus a automatiquement configuré les pools de données et de métadonnées nécessaires et a déployé des serveurs de métadonnées (MDS) redondants à travers le cluster pour une haute disponibilité et une tolérance aux pannes.

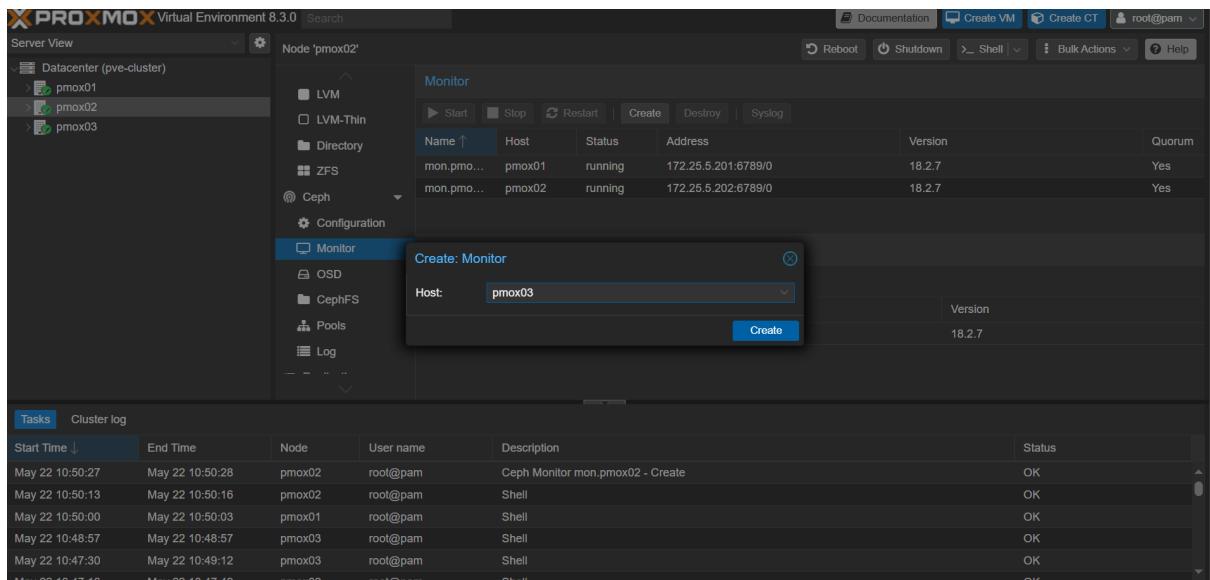


FIGURE A.10 – Création du système de fichiers CephFS

### A.3.4 Conclusion

Avec les OSDs contribuant au stockage, les MONs assurant le quorum, le pool répliqué fournissant des périphériques bloc résilients, et CephFS offrant un stockage de fichiers partagé, la couche de stockage hyper-convergée est maintenant complète. L'aboutissement de

## **Annexe A. PROXMOX**

---

ces étapes est une plate-forme de stockage unifiée, résiliente et entièrement opérationnelle, intégrée directement dans le cluster Proxmox, prête à supporter les charges de travail de calcul.

# Annexe B

## OPNsense

### B.1 Installation et configuration d'OPNsense

Le fondement de l'infrastructure réseau virtuelle de ce projet est l'appliance de pare-feu et de routage OPNsense. Déployée en tant que machine virtuelle dans l'environnement Proxmox, l'installation initiale a été effectuée à l'aide de l'installateur standard d'OPNsense via la console Proxmox. Une étape critique de la configuration initiale a consisté à obtenir l'accès à l'interface de gestion web. Par défaut, l'interface graphique d'OPNsense n'est accessible que sur son interface LAN, qui était connectée au pont réseau isolé `mainvnet`. Pour surmonter cela, une VM "de rebond" temporaire exécutant Ubuntu Desktop a été déployée sur le même `mainvnet`. Cette VM de rebond, dotée d'une adresse IP statique dans le sous-réseau LAN (192.168.0.2), a fourni l'environnement de navigateur nécessaire pour se connecter à l'interface graphique d'OPNsense à l'adresse <https://192.168.0.1> et terminer l'assistant de configuration initiale.

#### B.1.1 Périmètre de la configuration d'OPNsense

Pour les besoins de cette recherche, la configuration d'OPNsense s'est concentrée sur la fourniture de services de base de couches 3 et 4, essentiels au fonctionnement du cluster Kubernetes. Les fonctionnalités avancées telles que les protocoles de routage dynamique (par exemple, BGP), les services VPN et le trunking VLAN complexe ont été intentionnellement omises pour réduire la complexité et rester concentré sur les principaux objectifs de la recherche.

Il est important de noter, cependant, qu'OPNsense est une plate-forme très performante. Ces fonctionnalités avancées de réseau et de sécurité pourraient être mises en œuvre dans des travaux futurs pour améliorer l'évolutivité, la flexibilité et la posture de sécurité de l'environnement. Pour cette implémentation, la fonctionnalité de pare-feu a

également été désactivée après la configuration initiale pour garantir un flux de trafic sans restriction au sein de l'environnement de laboratoire sécurisé, simplifiant ainsi le déploiement et les tests des services conteneurisés.

### B.1.2 Configuration finale des services

La configuration réseau de base au sein d'OPNsense a ensuite été finalisée comme suit :

- Assignation des interfaces** : Les NICs virtuelles ont été assignées à des interfaces logiques dans OPNsense. Comme le montre la Figure B.1, `vtnet0` a été configuré comme interface LAN avec une IP statique de `192.168.0.1/16`, et `vtnet1` a été configuré comme interface WAN avec une IP statique de `172.25.5.204/24`.

Status	Interface	Device	VLAN	Link Type	IPv4	IPv6	Gateway	Routes	Commands
green	LAN (lan)	vtnet0		static	192.168.0.1/16			192.168.0.0/16	
green	WAN (wan)	vtnet1		static	172.25.5.204/24		172.25.5.1	default 172.25.5.0/24	
green	Loopback (lo0)	lo0		static	127.0.0.1/8	::1/128 fe80::1/64		127.0.0.1 172.25.5.204	
red	Unassigned Interface	enc0							
red	Unassigned Interface	pflago							

Showing 1 to 5 of 5 entries

FIGURE B.1 – Aperçu des interfaces OPNsense

- Passerelle et routage** : La route par défaut pour tout le trafic sortant a été dirigée via la passerelle du réseau physique. La `WAN_GW` a été explicitement définie avec l'adresse IP `172.25.5.1`, garantissant que le trafic du LAN virtuel peut être acheminé vers Internet (Figure B.2).
- Service DHCP** : Pour automatiser la gestion des adresses IP pour les clients sur le réseau virtuel, le serveur DHCP ISC a été activé sur l'interface LAN. Le serveur a été configuré pour gérer le sous-réseau `192.168.0.0/16`, en louant des adresses à partir d'un pool dynamique allant de `192.168.0.10` à `192.168.0.245` (Figure B.3). Cela fournit une configuration réseau automatique pour toutes les futures VMs et charges de travail de conteneurs.

La configuration réussie a abouti à un routeur virtuel entièrement opérationnel. Le tableau de bord d'OPNsense (Figure B.4) fournit un résumé de l'état opérationnel, confir-

## Annexe B. OPNsense

The screenshot shows the 'System: Gateways: Configuration' page. The left sidebar is titled 'System' and includes options like Access, Configuration, Firmware, Gateways, and Configuration. The main table has columns: Name, Interface, Protocol, Priority, Gateway, Monitor..., RTT, RTTd, Loss, Status, and Description. One entry is listed: 'WAN\_GW (active)' on 'WAN' interface with 'IPv4' protocol, priority '255 (upstream)', gateway '172.25.5.1', and a green status icon.

FIGURE B.2 – Configuration de la passerelle dans OPNsense

The screenshot shows the 'Services: ISC DHCPv4: [LAN]' configuration page. The left sidebar under 'Services' lists options like Captive Portal, DHCRelay, Dnsmasq DNS & DHCP, Intrusion Detection, ISC DHCPv4, and others. The main form has sections for Enable (checked), Deny unknown clients (unchecked), Ignore Client UIDs (unchecked), Subnet (192.168.0.0), Subnet mask (255.255.0.0), Available range (192.168.0.1 - 192.168.255.254), Range (from 192.168.0.10 to 192.168.0.245), Additional Pools, and WINS servers.

FIGURE B.3 – Configuration du serveur DHCP pour l'interface LAN

mant l'état des interfaces, la passerelle WAN active et les informations système, établissant ainsi un point de gestion et de surveillance centralisé pour l'ensemble du réseau virtuel.

## Annexe B. OPNsense

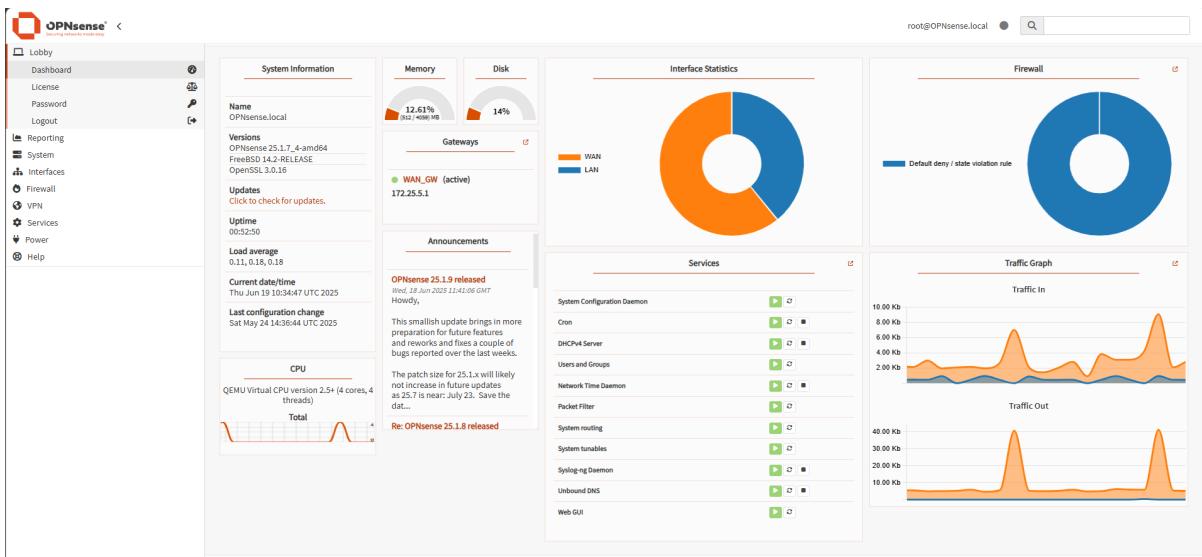


FIGURE B.4 – Le tableau de bord OPNsense montrant l'état opérationnel final

À l'issue de ces étapes, une base de réseau robuste et gérée de manière centralisée a été établie. L'appliance OPNsense fournit désormais des services essentiels, notamment le DHCP pour l'allocation automatique d'adresses IP et le routage par passerelle pour la connectivité externe. Cet environnement réseau configuré est un prérequis essentiel, fournissant la connectivité et la gestion IP nécessaires au déploiement ultérieur des noeuds du cluster Kubernetes. Le terrain est maintenant prêt pour la construction de la couche de calcul de l'infrastructure.

## Annexe C

# Diagrammes d'Architecture de l'Implémentation

Cette annexe fournit des diagrammes détaillés illustrant l'implémentation pratique de l'architecture décrite dans cette thèse. Ces aides visuelles ont pour but d'offrir une vue d'ensemble holistique et pratique des composants du système et de leurs interactions.

## C.1 Architecture Globale de l'Implémentation

La figure C.1 présente une vue d'ensemble complète de la plateforme implémentée, depuis la couche matérielle jusqu'à la couche d'IA agentique. Elle associe les technologies sélectionnées au Chapitre 4 à leurs rôles respectifs au sein de l'architecture multicouche, illustrant les flux de données et de contrôle qui rendent possibles les capacités autonomes du système.

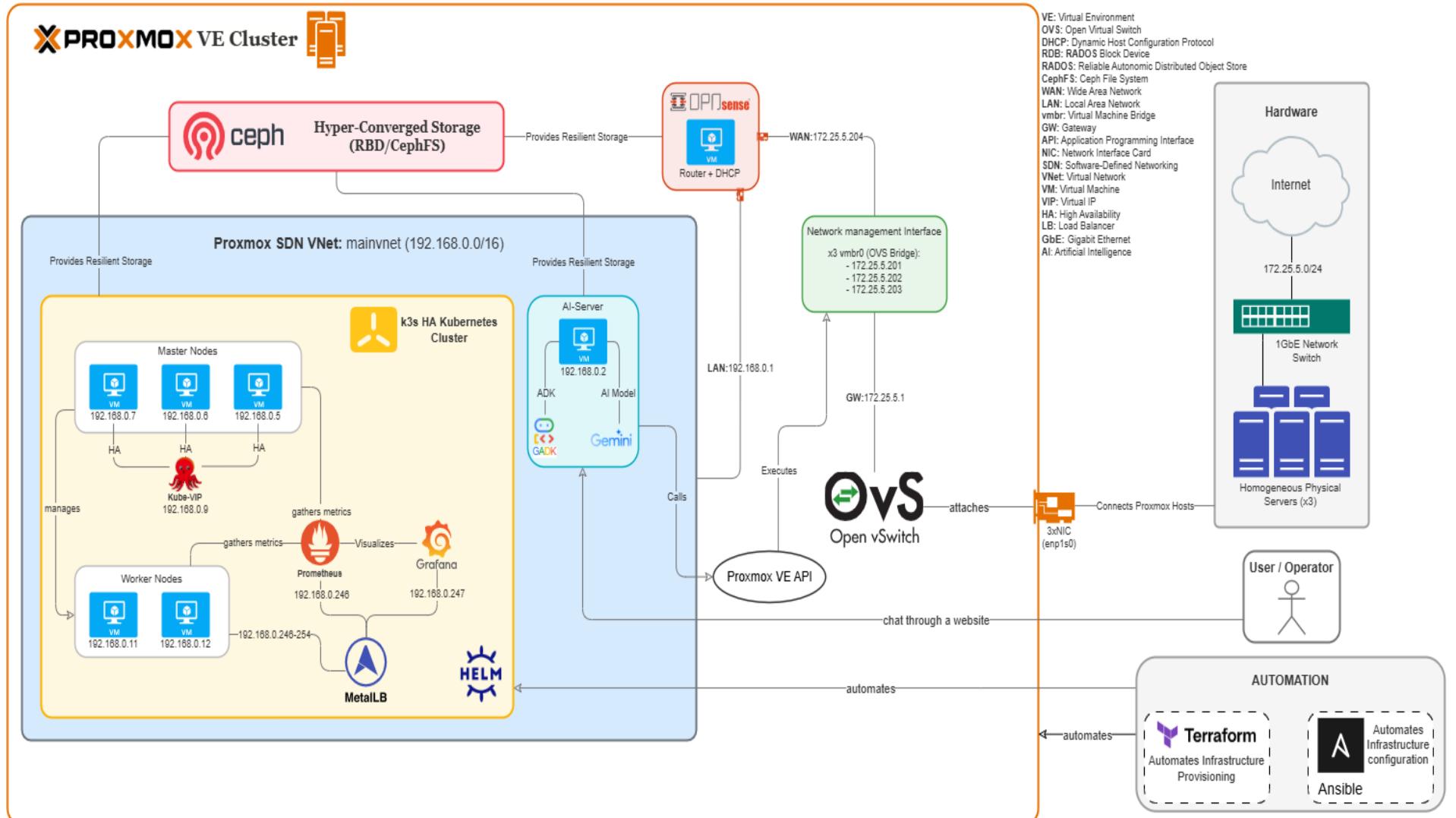


FIGURE C.1 – Un diagramme détaillé du système final implémenté

**Description de l'Architecture Globale de l'Implémentation** Le diagramme illustre l'infrastructure hyperconvergée et étroitement intégrée construite pour cette recherche :

- **Couche Physique et de Virtualisation** : À la base, trois **Serveurs Physiques Homogènes** fournissent les ressources de calcul, de stockage et de réseau. Un **Cluster Proxmox VE** est déployé sur ce matériel. La nature hyperconvergée est réalisée grâce à **Ceph**, qui fournit un stockage résilient en mode bloc (RBD) et fichier (CephFS) directement depuis les disques des serveurs, éliminant le besoin d'une baie de stockage distincte.
- **Couche Réseau** : Le réseau virtuel est géré par deux composants clés. **Open vSwitch (OVS)** sert de fabric de commutation défini par logiciel au sein de Proxmox. Une appliance virtuelle **OPNsense** agit comme routeur principal, pare-feu et serveur DHCP, connectant le **VNet SDN de Proxmox** interne et isolé (mainvnet, 192.168.0.0/16) au réseau externe.
- **Couche d'Orchestration de Conteneurs** : Les VM qui hébergent le **Cluster k3s HA Kubernetes** s'exécutent au sein du mainvnet. Cela inclut trois noeuds maîtres ('master') pour la haute disponibilité et plusieurs noeuds de travail ('worker'). Les services applicatifs sont exposés via **MetalLB**, qui fournit la fonctionnalité de 'LoadBalancer', tandis que l'empaquetage et le déploiement des applications sont gérés par **Helm**.
- **Supervision et Automatisation** : L'observabilité est assurée par **Prometheus** pour la collecte de métriques et **Grafana** pour la visualisation, qui recueillent les métriques du cluster Kubernetes. L'ensemble de l'infrastructure est automatisé en utilisant une approche d'Infrastructure as Code (IaC), avec **Terraform** pour le provisionnement des machines virtuelles et **Ansible** pour la gestion de leur configuration et le déploiement de k3s.
- **Couche d'IA Agentique** : La pierre angulaire de l'architecture est le **Serveur d'IA** ('AI-Server'), une VM dédiée hébergeant le Système Multi-Agents. Ce système est construit avec le **Kit de Développement d'Agents (ADK)** de Google et est propulsé par le modèle d'IA **Gemini**. Il interagit directement avec l'**API de Proxmox VE** pour effectuer des tâches de provisionnement intelligentes.
- **Interaction Utilisateur** : Le point d'entrée principal pour l'utilisateur est une interface de chat web, qui lui permet d'émettre des commandes en langage naturel au système agentique.

## C.2 Flux de Travail de la Couche d'IA Agentique

La figure C.2 détaille le flux de travail interne du Système Multi-Agents (SMA) qui a été implémenté pour gérer le provisionnement intelligent sur Proxmox. Elle illustre la

## Annexe C. Diagrammes d'Architecture de l'Implémentation

séquence des opérations et la collaboration entre les agents spécialisés, depuis la réception de la requête initiale de l'utilisateur jusqu'à la livraison du manifeste d'infrastructure final et affiné.

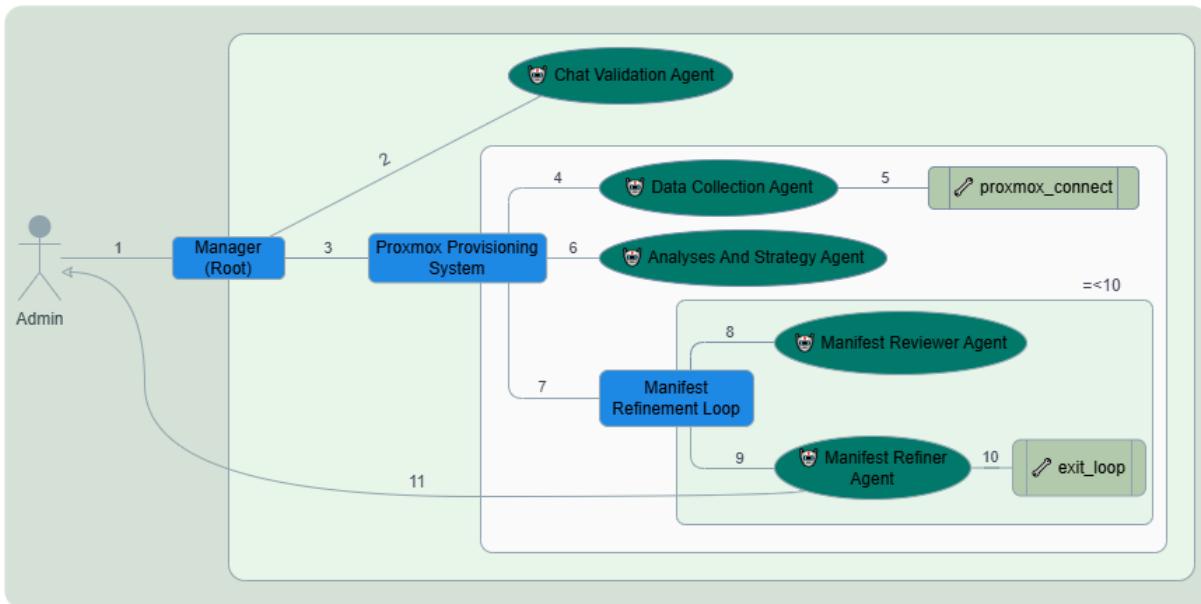


FIGURE C.2 – Le flux de travail étape par étape du Système Multi-Agents pour le provisioning sur Proxmox.

**Description du Flux de Travail de la Couche d'IA Agentique** Le flux de travail traduit une requête utilisateur de haut niveau en un artefact concret et déployable à travers une séquence d'actions coordonnées par les agents :

- Initiation : L'Administrateur** (utilisateur) soumet une requête en langage naturel (par ex., "crée une vm webserver") à l'agent **Manager (Racine)**, qui sert d'orchestrateur principal.
- Validation et Clarification** : Le Manager délègue l'interaction initiale à l'agent de **Validation par Chat** ('Chat Validation Agent'). Cet agent engage un dialogue avec l'utilisateur pour clarifier les ambiguïtés et recueillir les détails nécessaires, comme démontré au Chapitre 4.
- Invocation du Système** : Une fois l'intention clarifiée, le Manager initie formellement le **Système de Provisionnement Proxmox**, un sous-système spécialisé dédié à cette tâche.
- Collecte de Données** : L'agent de **Collecte de Données** ('Data Collection Agent') est activé. Il utilise un outil (`proxmox_connect`) pour s'interfacer avec l'API de Proxmox, collectant des données en temps réel sur l'état du cluster, telles que les ressources disponibles, les VM existantes et les pools de stockage.
- Formulation de la Stratégie** : L'agent d'**Analyse et de Stratégie** ('Analyses And Strategy Agent') reçoit l'objectif validé de l'utilisateur et les données en direct

de l'environnement. Il synthétise ces informations pour créer un plan de déploiement de haut niveau, décidant de paramètres tels que le placement des noeuds, l'allocation des ressources et la configuration réseau.

6. **Boucle d'Affinage Itératif** : Le cœur du processus de génération est la **Boucle d'Affinage du Manifeste** ('Manifest Refinement Loop'). Cette boucle incarne un schéma de "Raisonnement et Action" (ReAct) :

- L'agent **Réviseur de Manifeste** ('Manifest Reviewer Agent') examine l'ébauche initiale du fichier de configuration Terraform, identifiant toute information manquante, erreur logique ou anti-patron.
- L'agent **Affineur de Manifeste** ('Manifest Refiner Agent') prend en compte les retours du réviseur et travaille à corriger le manifeste. Cela peut impliquer de consulter la documentation, d'appliquer les meilleures pratiques ou d'utiliser son propre raisonnement pour générer le code de configuration manquant.
- Cette boucle peut s'exécuter plusieurs fois (jusqu'à une limite prédéfinie de 10, comme indiqué par `=<10`) jusqu'à ce que le manifeste soit jugé complet et correct.

7. **Finalisation et Livraison** : Une fois la boucle d'affinage terminée (`exit_loop`), le manifeste Terraform final et validé est retourné à l'**Administrateur**. Cette étape de "l'humain dans la boucle" ('human-in-the-loop') garantit la sécurité, permettant à l'opérateur de vérifier le code avant de l'appliquer à l'infrastructure.

Ce flux de travail structuré et collaboratif permet au système de combler de manière fiable et intelligente le fossé entre l'intention humaine de haut niveau et l'exécution de l'infrastructure de bas niveau.