الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
جامعة سعيدة د. مولاي الطاهر
كلية الرياضيات و الإعلام الآلي و الاتصالات السلكية و اللاسلكية
قسم: الإعلام الآلي

Mémoire de Master en informatique

Spécialité : Réseau informatique des systèmes répartis

T h è m e

Design and implementation of private cloud using OpenStack

- **Présenté par :**
  Berrezoug Sanaa
  Meghit Bouhana
- **Dirigé par :**
  Dr.Meddah Ishak

Année universitaire **2025-2026**

Remerciments

Avant tout propos, nous remercions Allah Le Tout-Puissant de nous avoir accordé la santé, la volonté, le courage et la persévérance nécessaires pour mener à bien ce travail. Nous implorons Sa bénédiction sur notre noble Prophète Mohammed Sallallahu Alayhi Wa Sallam, source de guidance et de lumière pour l'humanité toute entière.

Nos remerciements les plus sincères vont à notre encadrant, Docteur Meddah Ishak, pour la confiance qu'il nous a accordée en acceptant de diriger ce travail, pour la pertinence de ses orientations et pour ses précieux conseils qui ont guidé notre réflexion tout au long de l'élaboration de ce mémoire. Qu'il trouve ici l'expression de notre profonde gratitude et de notre plus grand respect.

Nos remerciements vont aussi à l'ensemble du corps enseignant de notre département qui, tout au long de notre cursus universitaire, a contribué à notre formation académique et professionnelle. Chaque cours, chaque encadrement et chaque échange a posé une pierre supplémentaire à l'édifice de nos connaissances et nous a aidés à forger les compétences qui ont rendu possible la réalisation de ce projet.

Nous tenons également à exprimer notre vive reconnaissance aux membres du jury qui ont accepté d'évaluer ce travail et de lui consacrer une partie de leur temps précieux.

Enfin, nous tenons à exprimer notre gratitude la plus profonde à toutes les personnes qui nous ont soutenus de près ou de loin durant la réalisation de ce travail — que ce soutien ait été technique, moral ou affectif. Leurs encouragements, leur patience et leur bienveillance ont représenté une source de motivation inestimable dans les moments de doute et de difficulté. Qu'ils sachent que leur contribution, quelle qu'en soit la forme, a été précieuse et sincèrement appréciée.

À tous, merci du fond du cœur.

Dedicase

je dédie ce travail à toutes les personnes qui occupent une place précieuse dans mon cœur. À

moi-même, Berrezoug Sanaa

La fille qui se bat, qui refuse le minimum et vise toujours les sommets. Celle qui n'a jamais abandonné, malgré tout.je suis fiere de toi .

À mon père, Hocine

Mon premier et plus fidèle soutien. Celui qui ne m'a jamais déçue, qui me rappelle sans cesse qui je suis et ce dont je suis capable. Que Dieu te protège et te garde comme une couronne brillant au-dessus de ma tête.

À ma mère, Leila

Celle dont les prières ont toujours été mon refuge, qui donne le meilleur d'elle-même pour pourvoir à tout ce dont j'ai besoin. Que Dieu te préserve, toi qui éclaires mon chemin comme une bougie dans la nuit.

À mes sœurs, Rofka, Timo, Hadjer et Leila

Mes amies, mes piliers, mes plus grandes supportrices. Je vous remercie du fond du cœur et suis profondément reconnaissante de vous avoir dans ma vie.

À ma binôme, Bouhana

Bien plus qu'une partenaire de projet, une véritable compagne de route qui m'a accompagnée tout au long de mon parcours académique, partageant chaque instant, dans ses joies comme dans ses épreuves.

À mon petit prince, Ghaith

Merci d'être la raison qui me fait sourire, même dans mes moments les plus sombres.

À mon ami Habib qui a toujours été là

Celui que le destin a placé sur ma route avant de nous séparer. Je te souhaite le succès et l'épanouissement où que tu sois.

Et enfin, à tous ceux qui m'ont soutenue de près ou de loin tout au long de ce chemin.

Berrezoug sanaa

Dédicace

Je dédie ce modeste travail :

À mes très chers parents

Pour leur amour inconditionnel, leurs sacrifices, leurs

Encouragements et leur soutien tout au long de mon parcours.

À mes frères et sœurs,

Pour leur affection et leur présence à mes côtés.

À ma binôme, Sanaa,

Pour sa collaboration sincère, son sérieux et son dévouement

Tout au long de la réalisation de ce travail. Je lui souhaite beaucoup de réussite dans sa vie

Professionnelle et personnelle.

À mes amies et collègues,

Pour leur aide, leurs encouragements et les agréables moments passés

Ensemble.

Enfin, à toutes les personnes qui ont participé de près ou de loin à la réalisation de ce mémoire.

Meghit Bouhana

Abstract

Abstract

In the current digital landscape, cloud computing has emerged as a transformative paradigm that reshapes the way organizations manage and deliver IT services. Its ability to provide on-demand resources, elastic scaling, and operational efficiency has driven widespread adoption across industries. Nevertheless, relying on public cloud providers introduces critical challenges around data privacy, sovereignty, and infrastructure governance, pushing many organizations to seek more controlled alternatives.

This thesis addresses these challenges by proposing the design and deployment of a private cloud infrastructure built upon OpenStack, a leading open-source cloud management platform. The resulting solution delivers a robust, adaptable, and self-hosted environment capable of orchestrating virtualized computing, storage, and network resources while maintaining full organizational control.

This study encompasses four interconnected phases: a theoretical grounding in cloud computing concepts and architectures, the design of a tailored private cloud topology, the hands-on deployment of OpenStack in a real environment, and a rigorous assessment of the platform's performance, reliability, scalability, and security posture.

Keywords: *Cloud Computing, Private Cloud, OpenStack, Virtualization, Infrastructure as a Service (IaaS), Security, High Availability.*

Résume

Résume :

À l'ère du numérique, l'informatique en nuage s'est imposée comme une composante incontournable des systèmes d'information modernes, offrant une gestion dynamique des ressources, une grande souplesse d'utilisation et une réduction sensible des coûts opérationnels. Toutefois, l'adoption des plateformes cloud publiques engendre des interrogations légitimes quant à la confidentialité des données, à la maîtrise des infrastructures et à l'indépendance technologique des organisations.

Dans cette perspective, le présent mémoire se propose de concevoir, de déployer et d'évaluer une infrastructure de cloud privé reposant sur la plateforme open source OpenStack. La solution développée ambitionne de répondre aux exigences de sécurité, d'évolutivité et de flexibilité, en assurant une administration centralisée et granulaire des ressources de traitement, de stockage et de communication réseau.

Ce travail s'articule autour de quatre axes principaux : une étude des fondements théoriques du Cloud Computing, la modélisation de l'architecture cible, la mise en œuvre technique de la plateforme OpenStack en environnement réel, et enfin une analyse approfondie des performances et du niveau de sécurité atteint par le système déployé.

Mots-clés : *Cloud Computing, Cloud Privé, OpenStack, Virtualisation, Infrastructure as a Service (IaaS), Sécurité.*

Liste des abréviations

ACL : Access Control List, Liste de contrôle d'accès

AES-256 : Advanced Encryption Standard 256 bits

AMQP : Advanced Message Queuing Protocol, Protocole de messagerie avancée

API : Application Programming Interface, Interface de programmation d'application

AWS: Amazon Web Services

CAPEX: Capital Expenditure

CI/CD: Continuous Integration / Continuous Deployment

CLI: Command Line Interface

CPU: Central Processing Unit

CSP: Cloud Service Provider

DHCP: Dynamic Host Configuration Protocol

DNS: Domain Name System

ENISA: European Union Agency for Cybersecurity

GB: Gigabyte

GPL: General Public License

HTTP: HyperText Transfer Protocol

HTTPS: HyperText Transfer Protocol Secure

HWA: Hardware-assisted Virtualization

IAM: Identity and Access Management

IaaS: Infrastructure as a Service

IDS: Intrusion Detection System

Intel VT-x: Intel Virtualization Technology for x86

IP: Internet Protocol

IT: Information Technology

JSON: JavaScript Object Notation

KVM: Kernel-based Virtual Machine

LDAP: Lightweight Directory Access Protocol

LTS: Long Term Support

LXC: Linux Containers

MAC: Media Access Control

MFA: Multi-Factor Authentication

NAT: Network Address Translation

NFS: Network File System

NIST: National Institute of Standards and Technology

NTP: Network Time Protocol

OPEX: Operational Expenditure

OVS: Open vSwitch

PaaS: Platform as a Service

QCOW2: QEMU Copy on Write v2

RBAC: Role-Based Access Control

REST : Representational State Transfer

RGPD : Règlement Général sur la Protection des Données

SaaS: Software as a Service

SDN: Software-Defined Networking

SIEM: Security Information and Event Management

SSH: Secure Shell

TLS: Transport Layer Security

URL: Uniform Resource Locator

UTC: Coordinated Universal Time

VIP: Virtual IP Address

VPC: Virtual Private Cloud

VNC: Virtual Network Computing

WSGI: Web Server Gateway Interface

Liste des figures

Liste des figures

Liste des tableaux

Table des matières

Table des matières

Table des matières

Introduction :

L'ère numérique contemporaine est marquée par une transformation profonde et irréversible des infrastructures informatiques. Les organisations, qu'elles soient publiques ou privées, font face à des exigences toujours plus élevées en matière de disponibilité des services, de capacité de traitement, de flexibilité des ressources et de sécurité des données. Cette pression constante vers plus d'agilité a conduit la communauté technologique mondiale à repenser fondamentalement le provisionnement, la gestion et la consommation des ressources IT.
C'est dans ce contexte que le Cloud Computing a émergé comme la révolution technologique la plus significative de ces deux dernières décennies. En proposant un modèle de consommation à la demande, via le réseau, avec une facturation basée sur l'usage réel, le Cloud a radicalement changé la relation entre les organisations et leur infrastructure. Il a permis de passer d'une logique d'investissement lourd (CAPEX) à une logique de dépenses opérationnelles flexibles (OPEX), tout en offrant une élasticité que les architectures traditionnelles ne pouvaient pas égaler.

La virtualisation des serveurs, des réseaux et du stockage constitue le fondement technique de cet édifice. Grâce aux hyperviseurs et aux technologies de conteneurisation, il est désormais possible de créer des environnements d'exécution isolés, reproductibles et portables. Cette abstraction du matériel physique a ouvert la voie à une gestion programmatique de l'infrastructure (Infrastructure as Code), permettant d'automatiser le déploiement et la configuration de l'ensemble des composants d'un système informatique.

Cependant, malgré ses avantages indéniables, le modèle de Cloud Public dominé par Amazon Web Services, Microsoft Azure et Google Cloud Platform soulève des préoccupations légitimes. La question de la souveraineté des données est devenue un enjeu stratégique majeur, particulièrement dans des secteurs sensibles comme la finance, la santé, la défense ou l'administration publique. Confier des données critiques à des prestataires tiers, soumis à des législations étrangères extraterritoriales et à des politiques de confidentialité parfois opaques, représente un risque géopolitique et juridique lourd. À cela s'ajoutent la dépendance technologique vis-à-vis d'un fournisseur (*vendor lock-in*), les coûts imprévisibles liés à une consommation non maîtrisée, ainsi que les strictes exigences de conformité réglementaire imposées par des cadres juridiques tels que le RGPD en Europe.

Face à ces contraintes, le modèle de Cloud Privé s'impose comme l'alternative stratégique de premier choix. En déployant une infrastructure au sein même de l'organisation, sur des serveurs

physiques entièrement contrôlés, il devient possible de conjuguer les bénéfices opérationnels du Cloud élasticité, automatisation, self-service avec les garanties de sécurité, de confidentialité et de conformité propres aux architectures dédiées. Le Cloud Privé offre ainsi un équilibre décisif entre modernité technologique et maîtrise organisationnelle.

Dans ce panorama, OpenStack se distingue comme la solution open source de référence mondiale pour la conception d'infrastructures de Cloud Privé et Hybride. Lancé en 2010 par la NASA et Rackspace, OpenStack est aujourd'hui porté par une communauté de développeurs et d'entreprises parmi les plus actives de l'écosystème open source. Son architecture modulaire, composée de services spécialisés et interopérables gérant respectivement le calcul, le réseau, le stockage, l'authentification et l'orchestration, lui confère une flexibilité exceptionnelle. Déployé dans des centaines de datacenters à travers le monde des opérateurs télécoms aux institutions bancaires, OpenStack témoigne d'une maturité technologique unique pour répondre à des besoins d'infrastructure à grande échelle.

Introduction Générale

Problématique :

La démocratisation du Cloud Computing a engendré une multiplicité de solutions, d'approches et de technologies. Cette profusion rend le choix et la mise en œuvre d'une infrastructure Cloud particulièrement complexes pour les organisations. Si les solutions de Cloud Public offrent une accessibilité immédiate, elles ne constituent pas une réponse universelle. Les organisations dont les activités impliquent le traitement de données sensibles, le respect de réglementations strictes ou la nécessité d'un contrôle granulaire sur les performances se trouvent dans l'obligation d'explorer des alternatives souveraines.

Le déploiement d'un Cloud Privé basé sur OpenStack représente une réponse techniquement viable à ces exigences, mais il soulève des défis majeurs :

- **La complexité de l'intégration modulaire :** OpenStack est un écosystème vaste composé de multiples services interconnectés. Sélectionner, configurer et orchestrer les composants de base (calcul, réseau, stockage, identité) requiert une compréhension approfondie de leurs interactions et de leurs dépendances mutuelles.
- **L'optimisation des ressources matérielles :** Garantir la performance brute sous charge, la sécurité de la pile technologique et la stabilité du système, tout en composant avec un environnement matériel défini (déploiement multi-nœuds sans redondance de haute disponibilité), constitue un arbitrage technique complexe.
- **La gouvernance opérationnelle :** La conception d'un Cloud Privé dépasse le cadre d'un exercice purement technique. Elle implique une réflexion approfondie sur l'architecture réseau, le dimensionnement des ressources et les processus opérationnels indispensables pour administrer et maintenir l'infrastructure.

C'est l'ensemble de ces défis que notre travail cherche à adresser, en proposant une démarche méthodologique complète allant de l'étude théorique des technologies retenues jusqu'à la validation expérimentale d'une infrastructure réelle.

Question de Recherche Centrale :

Comment concevoir et déployer une infrastructure de Cloud Privé fonctionnelle, sécurisée et performante basée sur les services fondamentaux d'OpenStack, capable de répondre aux besoins opérationnels d'une organisation tout en garantissant le contrôle total des ressources et la souveraineté des données ?

Questions de Recherche Secondaires :

- Fondements technologiques : Quels sont les concepts, les modèles et les technologies fondamentaux du Cloud Computing indispensables pour aborder le modèle de Cloud Privé, et en quoi se distingue-t-il des autres modes de déploiement ?
- Écosystème OpenStack : Quels sont les services fondamentaux d'OpenStack nécessaires à la construction d'un Cloud fonctionnel, quels sont leurs rôles respectifs, et comment s'articulent leurs interactions internes ?
- Conception architecturale : Comment concevoir une architecture de Cloud Privé basée sur OpenStack qui soit cohérente et évolutive, en tenant compte des contraintes matérielles disponibles, des exigences de performance et des politiques de sécurité ?
- Implémentation et intégration : Quelles sont les étapes et les défis techniques à surmonter lors du déploiement multi-nœuds des services retenus, incluant l'affectation des rôles (Controller, Compute) et la configuration des réseaux virtuels ?
- Validation et évaluation : Comment évaluer de manière rigoureuse les performances, la stabilité et la sécurité de l'infrastructure déployée, et dans quelle mesure les métriques obtenues confirment-elles la viabilité de la solution comme environnement de test ou de préproduction ?

Organisation de mémoire :

Pour répondre aux questions de manière progressive et cohérente, ce mémoire est organisé en quatre chapitres :

Le Chapitre 1 étudie les bases théoriques en abordant les fondements du Cloud Computing : ses caractéristiques essentielles, ses modèles de service (IaaS, PaaS, SaaS), ses modèles de déploiement et les technologies de virtualisation qui en constituent le socle, avec une attention particulière portée au modèle de Cloud Privé et à ses avantages stratégiques.

Le Chapitre 2 est consacré à l'étude approfondie de la plateforme OpenStack, de son architecture modulaire et de ses composants essentiels. Nous y analysons en détail le rôle et les mécanismes internes des services indispensables au cycle de vie des instances, avant d'établir une étude comparative avec les solutions concurrentes.

Le Chapitre 3 aborde la phase de conception de notre infrastructure de Cloud Privé. Nous y élaborons l'architecture cible, la topologie réseau, le dimensionnement des ressources, la répartition des rôles entre les nœuds (Controller, Compute) et les politiques de sécurité.

Enfin, le Chapitre 4 se concentre sur l'implémentation concrète de la solution ainsi que son évaluation. À travers des scénarios de tests de charge, d'allocation de ressources et d'audit de sécurité, les résultats sont analysés et interprétés pour démontrer la viabilité opérationnelle et les limites de l'infrastructure déployée.

Les Objectifs :

- Acquérir une maîtrise approfondie des concepts fondamentaux du Cloud Computing et des technologies de virtualisation qui le sous-tendent.
- Analyser l'architecture et les composants d'OpenStack afin d'en comprendre les interactions et les mécanismes internes.
- Concevoir une architecture de Cloud Privé adaptée aux contraintes fonctionnelles et non fonctionnelles d'une organisation.
- Déployer une infrastructure OpenStack fonctionnelle, sécurisée et correctement configurée dans un environnement réel.
- Valider les performances et la fiabilité de la solution à travers une série de tests et d'expérimentations.

Chapitre 01

Fondaments du cloud computing

1.Introduction :

L'informatique a connu une évolution majeure, passant des serveurs physiques locaux (On-Premise) à une infrastructure dématérialisée et flexible. Historiquement, les entreprises devaient investir lourdement dans du matériel coûteux et complexe à maintenir. L'émergence de la virtualisation a ensuite permis de mieux rentabiliser ces ressources, ouvrant la voie au Cloud Computing. Ce modèle permet désormais d'accéder à des ressources informatiques à la demande, via Internet, avec une agilité sans précédent.

Ce chapitre présente les bases théoriques essentielles du Cloud Computing, ses modèles de services et ses modes de déploiement. Nous analyserons également les mécanismes de sécurité et d'isolation nécessaires à la protection des données.

2.Evolution de l'informatique :

Le passage d'une informatique locale et rigide vers le modèle flexible du Cloud s'est appuyé sur trois piliers technologiques et économiques majeurs : le calcul distribué, l'abstraction matérielle et la consommation à l'usage.

2.1Grid Computing :

L'émergence du Grid Computing répond à la nécessité croissante de résoudre des problèmes scientifiques et industriels complexes nécessitant une puissance de calcul massive, dépassant les capacités d'une infrastructure isolée. Selon l'article séminal de Foster, Kesselman et Tuecke (2001) (I. Foster, 2001), cette technologie se définit comme une infrastructure logicielle permettant un partage de ressources coordonné, sécurisé et flexible au sein de « collections dynamiques

L'importance de ce paradigme réside dans sa capacité à transformer des ressources hétérogènes (processeurs, stockage, capteurs) en une entité logique unique, garantissant ainsi une interopérabilité sans précédent. Sur le plan structurel, l'article propose une architecture en couches standardisée (Layers), allant de la couche de « tissu » (*Fabric*) gérant les ressources locales, aux couches de connectivité et de ressources qui assurent les protocoles de communication et d'authentification. Cette approche modulaire, centrée sur des protocoles ouverts et des interfaces de programmation (API) extensibles, a jeté les bases critiques de l'abstraction et de la virtualisation, préfigurant ainsi l'évolution vers les modèles de services du Cloud Computing moderne.

2.2Virtualisation :

La virtualisation constitue le pilier technique majeur ayant rendu possible l'essor du Cloud Computing, en assurant l'abstraction des ressources matérielles physiques vers des entités logiques isolées et dynamiques. Formalisée par les travaux séminaux de Popek et Goldberg (Goldberg), elle repose sur l'interposition d'une couche logicielle l'hyperviseur ou *Virtual Machine Monitor* chargée de garantir trois propriétés fondamentales : l'équivalence du comportement, l'efficacité d'exécution et le contrôle exclusif des ressources. En dissociant le système d'exploitation de l'infrastructure matérielle sous-jacente, la virtualisation permet une consolidation optimale des serveurs et une isolation stricte des systèmes invités (VM), favorisant ainsi la continuité d'activité et l'optimisation énergétique (VMware, Consultez le 23/3/2026). Cette transformation d'un matériel statique en un pool de ressources mutualisées et extensibles constitue le socle technique de l'élasticité et du modèle à la demande, caractéristiques intrinsèques des environnements Cloud modernes.

2.3Utility Computing :

L'Utility Computing désigne un paradigme économique et opérationnel consistant à fournir des ressources informatiques puissance de calcul, stockage et services applicatifs selon un modèle comparable aux services publics traditionnels tels que l'électricité (Car, 200). Il repose sur la mutualisation des infrastructures, l'abstraction des ressources physiques et leur mise à disposition à la demande.

Ce modèle transforme l'informatique d'un actif détenu et géré en interne en un service externalisé, facturé selon un mécanisme de tarification à l'usage (Pay-as-you-go) (Rappa, 2004). Il permet ainsi le passage d'un modèle fondé sur les dépenses d'investissement (CAPEX) à un modèle basé sur les dépenses opérationnelles (OPEX).

Sur le plan technique, l'Utility Computing s'appuie sur la virtualisation, l'orchestration des ressources et des mécanismes de métrologie avancés assurant la mesure précise et en temps réel de la consommation. Il garantit l'élasticité, l'optimisation des coûts et une flexibilité organisationnelle accrue, tout en soulevant des enjeux liés à la gouvernance des données, à la dépendance aux fournisseurs et à la gestion des niveaux de service (R. Buyya, 2009)

3.Cloud Computing :

3.1Définition :

Le Cloud Computing s'inscrit dans la continuité évolutive des concepts de Grid Computing et d'Utility Computing, dont il systématise les fondements techniques et économiques. Il se définit comme un modèle permettant un accès réseau omniprésent, pratique et à la demande à un ensemble mutualisé de ressources informatiques configurables (réseaux, serveurs, stockage, applications et services), rapidement provisionnées et libérées avec un effort minimal de gestion. (Grance, 2011)

Ce paradigme repose sur cinq caractéristiques essentielles : le libre-service à la demande, l'accès réseau étendu, la mutualisation des ressources, l'élasticité rapide et le service mesuré. Il tend à assimiler l'informatique à une utilité comparable aux services publics traditionnels, dans laquelle la complexité infrastructurelle est masquée par des couches d'abstraction au profit d'une consommation flexible et orientée service. (R. Buyya, 2009)

En intégrant différents modèles de service (IaaS, PaaS, SaaS) ainsi que plusieurs modes de déploiement (public, privé, hybride), le Cloud Computing ne constitue pas uniquement une évolution technologique, mais une transformation structurelle des modalités de production, de gestion et de consommation des ressources numériques.

3.2Les Caractéristiques Essentielles du Cloud Computing :

Le modèle du Cloud Computing se distingue des architectures informatiques traditionnelles par un ensemble de caractéristiques structurelles qui en définissent la spécificité. Parmi les cinq propriétés fondamentales généralement admises, trois jouent un rôle central dans l'agilité opérationnelle et l'optimisation des ressources. (Grance, 2011)

- **Le libre-service à la demande (On-demand Self-service) :** désigne la capacité pour un utilisateur de provisionner de manière autonome des ressources informatiques puissance de calcul, stockage ou capacités réseau sans interaction humaine directe avec le fournisseur. Cette automatisation repose sur des interfaces programmables (API) ou des tableaux de bord de gestion. Dans une infrastructure telle que OpenStack, cette fonctionnalité est assurée par des services d'orchestration permettant la gestion complète du cycle de vie des instances virtuelles.
- **La mutualisation des ressources (Resource Pooling) :** correspond au regroupement des ressources physiques et virtuelles du fournisseur afin de servir simultanément plusieurs clients selon un modèle multi-locataire (multi-tenancy). Grâce aux mécanismes d'abstraction et de virtualisation, les ressources sont dynamiquement allouées et réallouées

en fonction de la demande, tout en garantissant une isolation logique stricte entre les environnements des différents utilisateurs.

- **L'élasticité rapide (Rapid Elasticity) :** se définit comme la capacité du système à adapter automatiquement les ressources disponibles aux variations de la charge de travail. Cette adaptation peut prendre la forme d'un redimensionnement vertical (scale-up/scale-down) ou horizontal (scale-out/scale-in). Elle permet une adéquation continue entre les besoins applicatifs et les ressources mobilisées, contribuant ainsi à l'optimisation du rapport performance/coût
- **L'accès réseau étendu (*Broad network access*) :** Cette propriété garantit que les services du Cloud sont accessibles via des mécanismes réseaux standards (protocoles HTTP, HTTPS, SSH) et sur une large gamme de terminaux (postes de travail, terminaux mobiles, serveurs distants). Dans une architecture OpenStack déployée sur Ubuntu 24.04, cela se traduit par l'exposition des services via des points d'accès (Endpoints) routables, permettant une administration à distance fluide et une interconnexion transparente des ressources.
- **Le service mesuré (*Measured service*) :** Le système Cloud contrôle et optimise automatiquement l'utilisation des ressources en s'appuyant sur des capacités de mesure (métrologie) adaptées au type de service (stockage, puissance de calcul, bande passante). Cette transparence, tant pour le fournisseur que pour l'utilisateur, est le moteur technique de l'**Utility Computing**. Elle permet une facturation précise basée sur la consommation réelle (*Pay-as-you-go*) et assure une gouvernance rigoureuse des ressources allouées.

3.3Les Modèles de Service du Cloud Computing :

L'offre de services au sein du Cloud Computing se structure traditionnellement autour de trois modèles principaux, complétés par des dimensions organisationnelles et infrastructurelles essentielles, comme le définit l'ENISA (Cybersecurity), 2026) :

- **Infrastructure as a Service (IaaS) :** À ce niveau, le fournisseur met à disposition des ressources matérielles virtualisées (calcul, stockage, réseau) accessibles en ligne. L'accès à ces ressources est orchestré par un hyperviseur. L'IaaS se décompose principalement en puissance de traitement et en stockage (bloc ou objet), offrant ainsi une flexibilité totale sur la gestion des systèmes.
- **Platform as a Service (PaaS) :** Ce modèle fournit un environnement d'exécution, notamment des serveurs d'applications, permettant aux clients de déployer des scripts (Python, PHP) ou du byte code (Java). Le fournisseur peut également inclure des outils de

développement, simplifiant ainsi le cycle de vie des applications sans gestion directe de l'infrastructure sous-jacente.

- **Software as a Service (SaaS) :** Le fournisseur délivre des applications complètes et prêtes à l'emploi (éditeurs de documents, CRM, messageries) via Internet, généralement accessibles par un navigateur web. Il est fréquent que les solutions SaaS soient elles-mêmes hébergées sur des infrastructures IaaS ou PaaS tierces, illustrant l'interdépendance des couches du Cloud.

Enfin, l'efficacité de ces services repose sur deux piliers critiques : les Infrastructures physiques (*Facilities*), regroupant les réseaux, le refroidissement et l'énergie, ainsi que l'Organisation, qui englobe les ressources humaines et les politiques de gestion garantissant la continuité et le support des services délivrés.

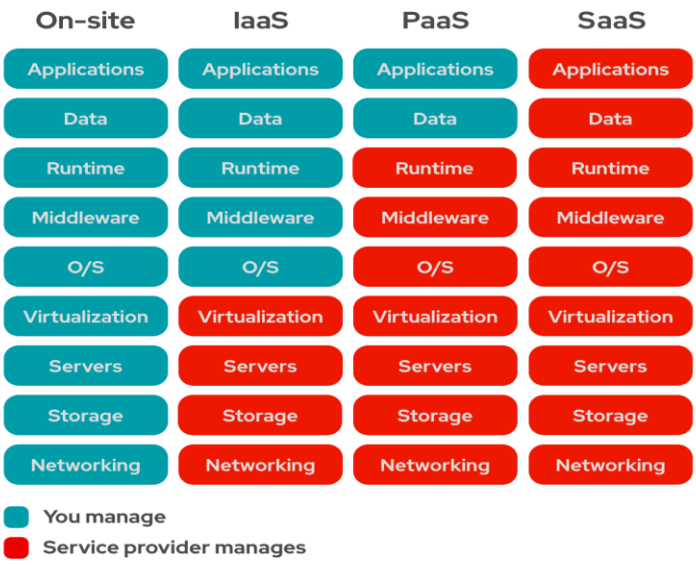

Figure 1 Comparaison des modèles de service et répartition des responsabilités (Rahmani, s.d.)

3.4Les Modèles De Déploiement De L'infrastructure Cloud :

Selon le cadre de référence du NIST (al, 2011,Consulté le 1er mars 2026), les modèles de déploiement de l'infrastructure cloud se distinguent par le degré d'exclusivité des ressources allouées au consommateur. On identifie quatre modèles fondamentaux :

Le Cloud Public : Dans ce modèle, l'infrastructure et les ressources de calcul sont rendues accessibles au grand public via un réseau ouvert. Propriété d'une organisation commerciale vendant des services de Cloud, il dessert une clientèle hétérogène et massive. Le Cloud public repose sur une mutualisation des ressources, permettant une élasticité élevée et un passage d'un

modèle d'investissement matériel (CapEx) vers des charges opérationnelles (OpEx) indexées sur l'usage réel.

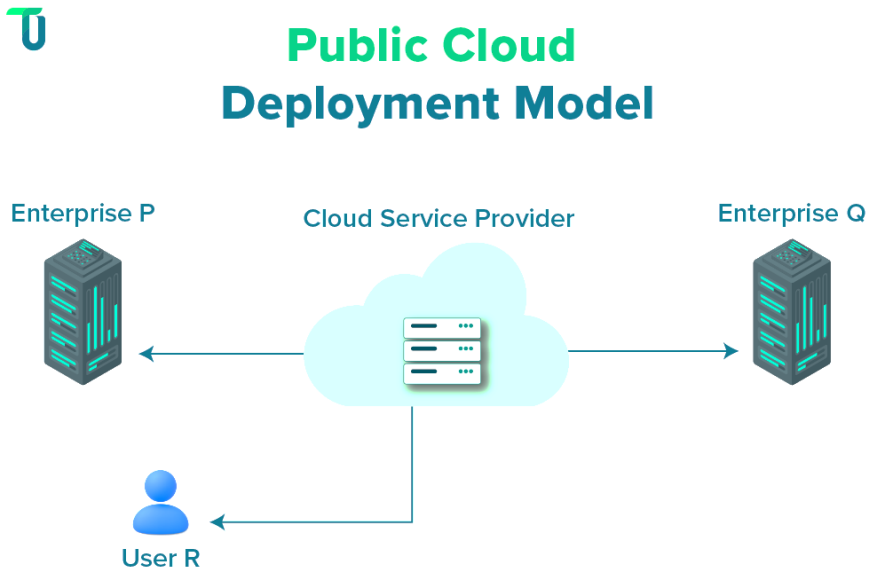

Figure 2 le flux de travail du cloud public (Services, Consultez le 29/1/2026)

Le Cloud Privé : Ce modèle garantit à l'organisation d'un consommateur unique l'accès et l'usage exclusifs de l'infrastructure et des ressources de calcul. Sa particularité réside dans sa flexibilité de gestion et de localisation : il peut être administré en interne par l'organisation elle-même (on-site) ou par un tiers, et peut être hébergé soit dans les locaux de l'entité, soit externalisé auprès d'une société d'hébergement (outsourced). C'est le modèle privilégié pour la souveraineté et le contrôle total des données.

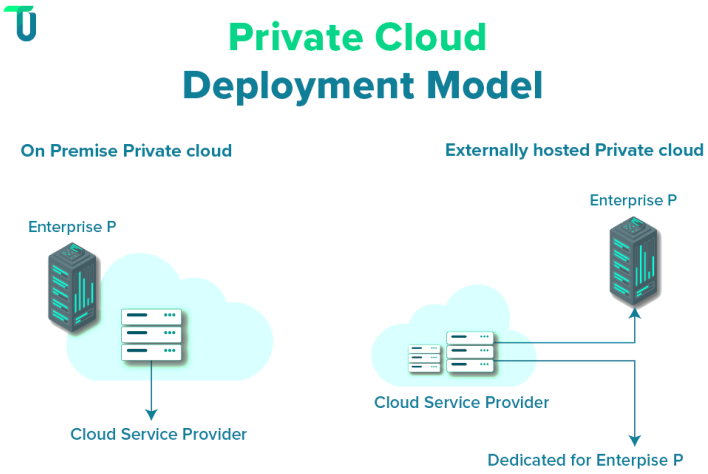

Figure 3 le cloud privé en action (Services, Consultez le 29/1/2026)

Le Cloud Hybride : Il représente une composition de deux ou plusieurs infrastructures de Cloud distinctes (privées, publiques ou communautaires). Bien que ces entités demeurent séparées et uniques, elles sont liées par des technologies propriétaires ou standardisées qui assurent la portabilité des données et des applications. Ce paradigme permet notamment de combiner la sécurité d'un Cloud privé avec la puissance de calcul flexible d'un Cloud public lors de pics d'activité.

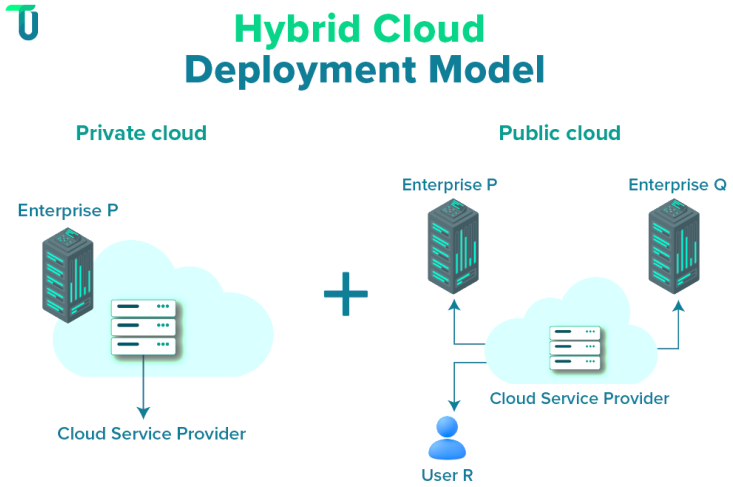

Figure 4 déploiement de cloud hybride (Services, Consultez le 29/1/2026)

Caractéristique	Cloud public	Cloud privé	Cloud hybride
Propriété	Fournisseur tiers	Organisation unique	Combinaison des deux
Accès	Partagé entre plusieurs utilisateurs	Réservé à une seule organisation	Mélange d'accès public et privé
Coût	Faible coût initial, paiement à l'usage	Coût plus élevé (infrastructure dédiée)	Coût équilibré
Sécurité	Standard, dépend du fournisseur	Élevée, contrôle total	Supérieure au public, inférieure au privé
Scalabilité	Très élevée	Limitée aux ressources internes	Bonne évolutivité sur les deux environnements
Contrôle	Faible	Contrôle total	Contrôle partiel
Idéal pour	Startups, tests, applications générales	Banques, santé, secteurs réglementés	Entreprises cherchant flexibilité et optimisation

Tableau 1 Comparaison synthétique des modèles de déploiement du Cloud Computing

4.Technologies de Virtualisations :

4.1Les Hyperviseurs :

L'architecture des systèmes virtualisés se définit par la position du moniteur de machines virtuelles (VMM) au sein de la pile logicielle (Miers, 2023), la distinction majeure repose sur l'interaction directe ou indirecte avec les ressources physiques de l'hôte.

a) **L'Hyperviseur de Type 1 (Bare Metal) :** s'exécute directement sur le matériel hôte (Bare Metal), sans couche logicielle intermédiaire. Cette configuration lui confère un contrôle total et exclusif sur les ressources critiques telles que le CPU, la mémoire vive et les interfaces d'entrées/sorties (E/S). En minimisant l'empilement des couches (overhead), le type 1 garantit des performances optimales et une latence réduite, indispensables pour les infrastructures de Cloud Computing et les centres de données. Sur le plan de la sécurité, cette architecture réduit considérablement la surface d'attaque : l'isolation est renforcée car les machines virtuelles (VM) communiquent

directement avec un micro-noyau sécurisé. Les solutions industrielles majeures comme VMware ESXi, Microsoft Hyper-V et Xen illustrent cette catégorie. (Tanenbaum, 2023)

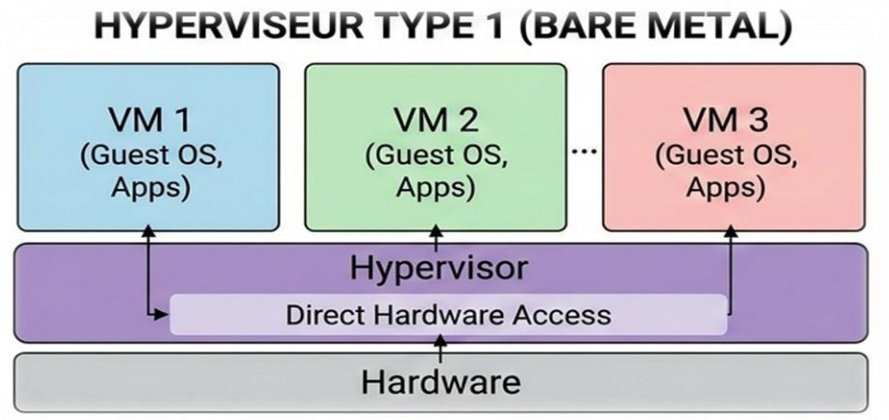

Figure 5 l'architecture d hyperviseur type 1 (Goffinet, s.d.)

b) **L'Hyperviseur de Type 2 (Hosted)** : À l'opposé, l'hyperviseur de type 2 est qualifié d'hyperviseur hébergé car il s'installe comme une application classique au-dessus d'un système d'exploitation hôte (Windows, Linux ou macOS). Bien que cette structure induise une surcharge de performance due à la gestion des pilotes par l'OS hôte, elle offre une flexibilité supérieure pour les environnements de test et de développement. Comme le soulignent Miers et Grambow (2023), l'efficacité de ces systèmes dépend du degré d'interception des appels systèmes, distinguant ainsi la virtualisation complète de la paravirtualisation et de la virtualisation assistée par le matériel (HWA). Des outils tels qu'Oracle VM VirtualBox ou VMware Workstation sont les représentants types de cette approche, privilégiant l'interopérabilité à la performance brute. (Miers, 2023)

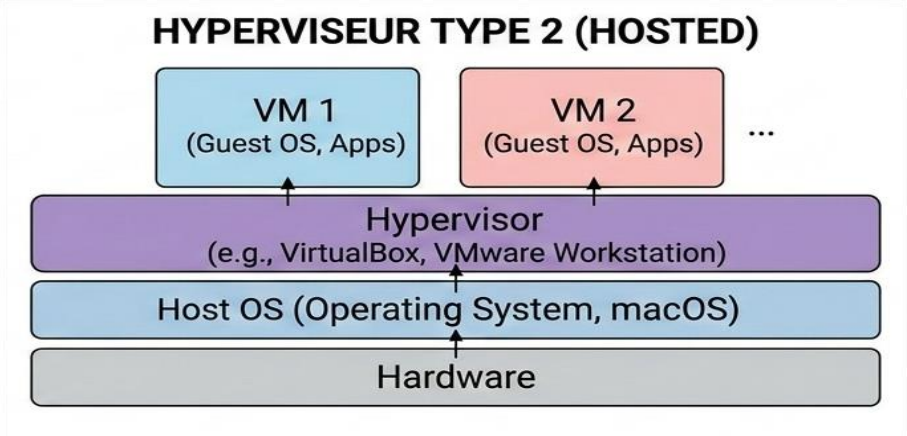

Figure 6 l'architecture d hyperviseur type 2 (Goffinet, s.d.)

4.2 Kernel-based Virtual Machine (KVM):

Le KVM (Kernel-based Virtual Machine) est un hyperviseur "Open Source" intégré nativement au noyau Linux. Contrairement aux solutions de Type 2, KVM transforme le noyau Linux en un hyperviseur de Type 1 (Dall, 2014), cette architecture permet au noyau de fonctionner comme un moniteur de machines virtuelles (VMM) en exploitant directement les extensions matérielles Intel VT-x ou AMD-V. Cette approche garantit des performances quasi-natives en délégant la gestion des ressources (mémoire, ordonnancement) aux mécanismes matures du noyau Linux, tout en assurant une isolation matérielle stricte (Miers, 2023)

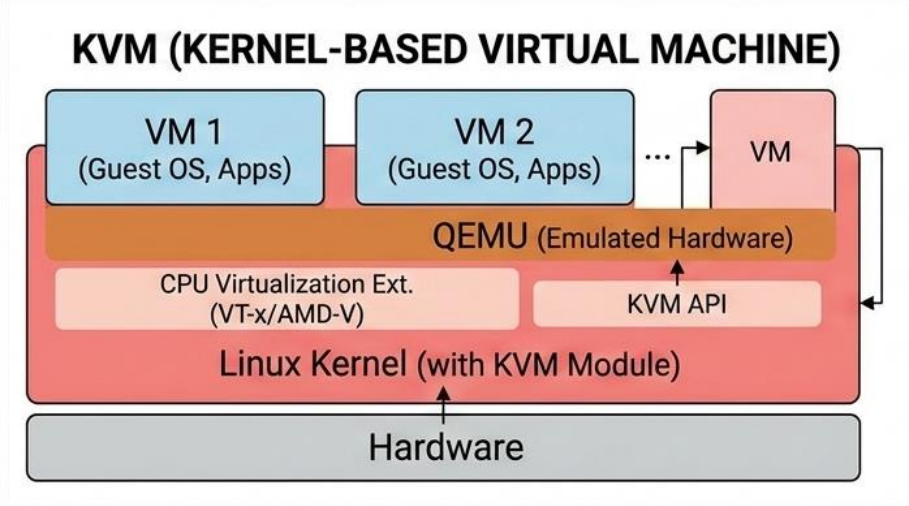

Figure 7 l'architecture Du kvm (Rahmani, s.d.)

4.3 La Conteneurisation (Virtualisation au niveau OS) :

La conteneurisation est une méthode de virtualisation légère qui permet d'exécuter des applications isolées en partageant le noyau (kernel) du système d'exploitation hôte. Contrairement aux machines virtuelles (VM) qui nécessitent un hyperviseur et un OS invité complet, les conteneurs encapsulent uniquement l'application et ses dépendances (bibliothèques, binaires).

Cette technologie repose sur deux piliers fondamentaux du noyau Linux : les Namespaces, qui assurent l'isolation des ressources (réseau, montages, processus), et les Control Groups (cgroups), qui limitent et surveillent la consommation des ressources matérielles (CPU, RAM, E/S). Cette architecture élimine la surcharge (overhead) liée à l'émulation matérielle, permettant des temps de démarrage en quelques millisecondes et une densité de déploiement nettement supérieure à celle des hyperviseurs de Type 1 ou 2. Toutefois, (Miers, 2023), le partage du noyau hôte induit une surface d'attaque plus large, rendant l'isolation des conteneurs intrinsèquement moins robuste que celle des machines virtuelles basées sur le matériel.

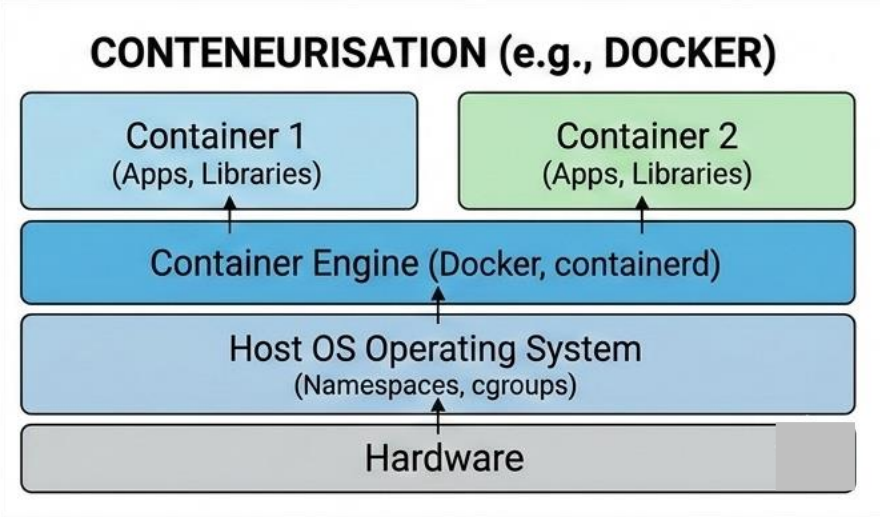

Figure 8 Architecture du conteneur

5. LES PILIERS DE LA SÉCURITÉ DANS LE CLOUD :

L'adoption du cloud computing impose un changement de paradigme en matière de sécurité, passant d'une protection périmétrique physique à une approche centrée sur l'identité et les données. La sécurité repose sur le principe du Modèle de Responsabilité Partagée, où le fournisseur (CSP) sécurise l'infrastructure physique et l'isolation, tandis que le client (CSC) assume la responsabilité de ses configurations d'accès, de ses données et de la gestion de ses identités.

5.1 Gestion des Identités et des Accès (IAM) :

L'IAM est le cadre de politiques et de technologies garantissant que seuls les utilisateurs et appareils autorisés accèdent aux ressources appropriées. Il constitue la défense principale contre le détournement de comptes (Account Hijacking).

- **Cycle de vie et Composants** : Un système complet gère l'identité de l'onboarding à la révocation, incluant un annuaire centralisé et la gestion des accès.
- **Contrôle d'Accès (RBAC) :** Il utilise le contrôle d'accès basé sur les rôles pour appliquer le principe du moindre privilège, limitant l'accès au strict nécessaire.
- **Authentification Avancée :** L'utilisation de l'authentification multi-facteurs (MFA) et d'identifiants dynamiques réduit considérablement les risques de compromission liés au phishing ou au vol d'identifiants.
- **Surveillance et Audit :** Des journaux d'activité détaillés permettent de détecter les anomalies comportementales et de répondre aux exigences de conformité telles que le RGPD.

5.2 Isolation des Ressources (Multi-tenance) :

L'isolation technique assure l'étanchéité entre les différents clients (tenants) qui partagent physiquement la même infrastructure. (Arunkumar, 2023)

- **Isolation au niveau Virtualisation :** L'hyperviseur sépare les machines virtuelles (VM) pour empêcher une instance d'accéder aux données d'une autre VM sur le même hôte.
- **Isolation Réseau :** L'utilisation de VPC (Virtual Private Cloud) et de la segmentation réseau permet d'isoler les applications critiques de l'Internet public, limitant ainsi la surface d'attaque.
- **Continuité de la Sécurité :** Lors de la migration de Machines virtuelles (VM) entre serveurs physiques, les politiques de filtrage doivent être maintenues pour éviter toute faille d'isolation durant le transfert.

5.3. Protection des Données :

La protection des données doit couvrir les états de repos (*at rest*), de transit (*in transit*) et d'utilisation (*in use*). (Arunkumar, 2023)

- **Chiffrement Avancé :** L'usage d'algorithmes comme AES-256 pour le stockage et de protocoles TLS/HTTPS pour les transferts garantit la confidentialité.
- **Souveraineté des données :** Le chiffrement côté client permet de garder le contrôle total des clés privées, empêchant même le fournisseur de cloud d'accéder aux données en clair.
- **Intégrité et Disponibilité :** Des mécanismes comme la signature numérique RSA et la géo-redondance (GRS) protègent les données contre les altérations malveillantes et les pannes majeures.

5.4. Comparaison entre cloud computing et grid :

Critère	Grid	Cloud
Facturation	Non	Oui
Virtualization	Faible	Forte
Elasticity	Limitée	Forte
Provisionnement	Manuel	Automatique

Tableau 2 comparaison entre cloud computing et grid

6.Avantages et limites du Cloud computing :

Avantages majeures:

- Réduction des coûts (passage de CapEx à OpEx)
- Élasticité et scalabilité à la demande
- Haute disponibilité (>99,9 %), mobilité, délégation de maintenance

Limites importantes:

- Dépendance à la connectivité Internet
- Risques sur la confidentialité et souveraineté des données (RGPD, etc.)
- Verrouillage fournisseur (vendor lock-in)
- Imprévisibilité des coûts et performances variables

7.Conclusion de chapitre :

Ce premier chapitre a posé les bases conceptuelles et technologiques nécessaires à la mise en œuvre d'une infrastructure cloud. Il a retracé l'évolution depuis le Grid et l'Utility Computing jusqu'aux architectures modernes, en soulignant le rôle central de la virtualisation (hyperviseurs Type 1 comme KVM, et conteneurisation).

L'analyse des modèles de service (IaaS, PaaS, SaaS) et de déploiement (public, privé, hybride) a permis de situer le projet. Les cinq caractéristiques essentielles du NIST (libre-service, accès réseau, mutualisation, élasticité, service mesuré) confirment la pertinence du cloud. Les enjeux de sécurité (identités, isolation, protection des données) sont également soulignés.

Face à ces limites, le **cloud privé** s'impose comme une alternative stratégique pour les organisations traitant des données sensibles, car il combine les avantages du cloud (flexibilité, automatisation) avec un contrôle total et la conformité réglementaire.

Ce chapitre a fourni les prérequis indispensables avant d'aborder, dans le chapitre suivant, l'étude approfondie d'**OpenStack** (historique, architecture modulaire, composants, comparaison avec les solutions concurrentes).

Chapitre 02

Etude de la plateforme open stack

1. Introduction :

En juillet 2010, Rackspace Hosting et la NASA ont lancé conjointement un nouveau projet *open source* dans le domaine du *cloud computing* sous le nom d'OpenStack . L'objectif du projet OpenStack est de permettre à toute organisation de créer et d'offrir des services de *cloud computing* en utilisant du matériel standard. La première version livrée par la communauté, dont le surnom est Austin, fut disponible dès octobre 2010. Il est prévu de livrer régulièrement des mises à jour logicielles à quelques mois d'intervalle. (Noisette, 2010)

OpenStack est une plateforme Open Source qui permet de créer et gérer des clouds privés ou publics à partir de pools de ressources virtuelles. Les outils (ou « projets ») qui constituent la plateforme OpenStack assurent les principaux services de cloud computing, à savoir, le calcul, la mise en réseau, le stockage, la gestion des identités et la gestion des images. La dizaine de projets restants, disponibles en option, peuvent également être groupés pour créer des clouds uniques.

2. Architecture modulaire :

OpenStack possède une architecture modulaire qui comprend de nombreux composants.

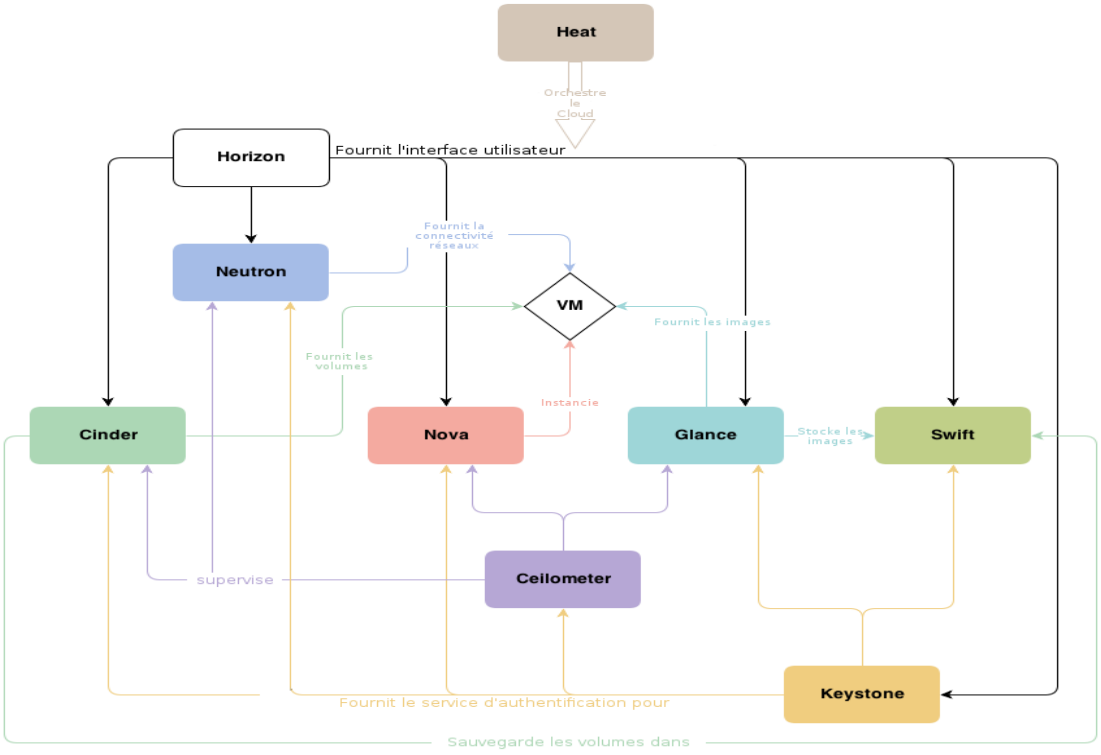

Figure 9 Architecture des composants Openstack (Docs, consultez 2/3/2026)

3. Cas d'usage

OpenStack est utilisé dans plusieurs contextes réels :

- **Cloud privé d'entreprise** : une organisation déploie son propre cloud interne pour gérer ses ressources IT de manière centralisée et sécurisée.
- **Hébergement de machines virtuelles** : lancement rapide de Machines virtuelles (VM) pour les développeurs ou les applications métier, sans avoir besoin de matériel physique dédié.
- **Environnements de développement et de test** : les équipes peuvent créer et supprimer des environnements à la demande, ce qui accélère le cycle de développement.
- **Stockage objet à grande échelle** : grâce au service Swift, OpenStack permet de stocker de grandes quantités de données de manière distribuée (similaire à Amazon S3).
- **Déploiement d'applications conteneurisées** : OpenStack peut héberger des clusters Kubernetes ou Docker pour les applications modernes.

4. LES COMPOSANTES PRINCIPAUX :

Le fonctionnement de la plateforme OpenStack repose sur un ensemble de composants spécialisés qui travaillent ensemble pour gérer les différentes ressources du cloud. Chaque composant joue un rôle spécifique dans l'administration de l'infrastructure, que ce soit pour le calcul, le stockage ou le réseau. Cette architecture modulaire permet d'assurer une meilleure flexibilité, une grande évolutivité ainsi qu'une gestion efficace des services cloud. Dans ce contexte, il est important de présenter les principaux composants qui constituent la plateforme OpenStack.

- ### Service de calcul (Nova) :

OpenStack Nova repose sur une architecture modulaire composée de six composants principaux : nova-api, file de messages (queue), nova-db, nova-conductor, nova-scheduler et nova-compute. Ces composants coopèrent afin d'assurer la gestion complète du cycle de vie des machines virtuelles. (Datt, A., Goel, A., & Gupta, S. C., 2015)

Le service nova-api constitue le point d'entrée pour les interactions avec les utilisateurs et prend en charge l'API OpenStack Compute. La file de messages représente le mécanisme de communication interne entre les différents composants de Nova, permettant l'échange

asynchrone de messages. Le composant nova-db correspond à la base de données relationnelle qui stocke l'état des instances ainsi que les informations relatives à l'infrastructure. (Datt, A., Goel, A., & Gupta, S. C., 2015)

Le module nova-conductor agit comme un intermédiaire entre nova-compute et la base de données. Il supprime l'accès direct de nova-compute à la base de données, renforçant ainsi la sécurité de l'architecture. Le composant nova-scheduler est responsable de la planification des machines virtuelles. Il reçoit les requêtes de création d'instances via la file de messages et utilise des algorithmes de planification afin de sélectionner l'hôte de calcul le plus approprié pour exécuter la machine virtuelle. (Datt, A., Goel, A., & Gupta, S. C., 2015)

Le service nova-compute est un démon chargé de gérer le cycle de vie des instances de machines virtuelles en s'appuyant sur les API des hyperviseurs pris en charge par la technologie de virtualisation utilisée. Il reçoit les requêtes à exécuter depuis la file de messages, effectue les opérations système correspondantes et met à jour les informations nécessaires dans la base de données par l'intermédiaire du service nova-conductor. (Datt, A., Goel, A., & Gupta, S. C., 2015)

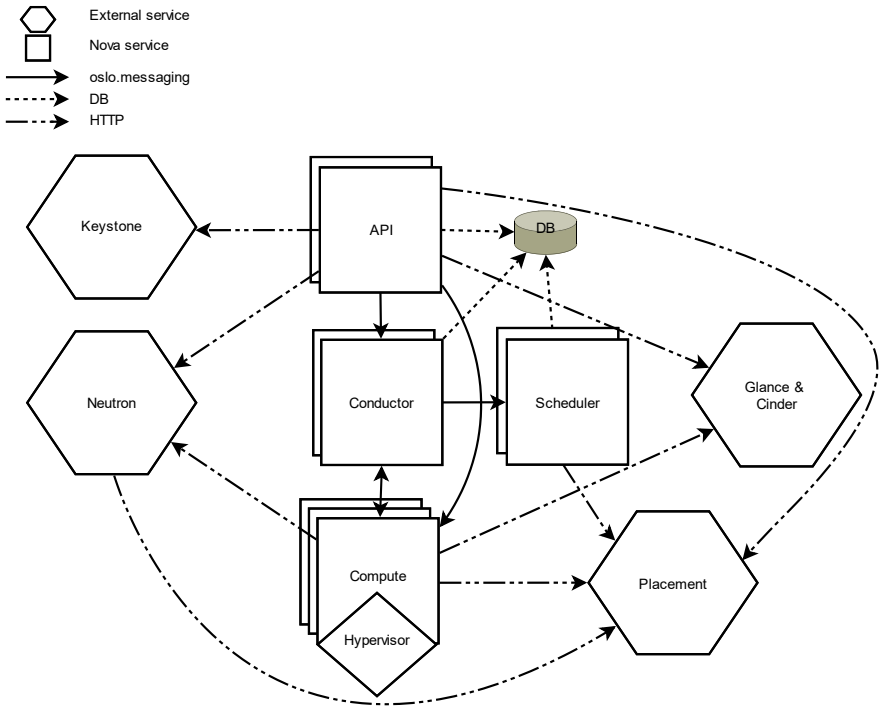

Figure 10 architecture_nova (Docs, consultez 2/3/2026)

- **Service réseau (Neutron) :**

OpenStack Neutron est le service de mise en réseau d'OpenStack, entièrement piloté par API. Il permet de définir, gérer et contrôler la configuration réseau au sein de l'infrastructure cloud. Grâce à Neutron, il est possible de créer, segmenter, fusionner ou isoler des réseaux à la demande, selon les besoins des utilisateurs. (Bakshi, 2015)

Neutron agit comme une couche d'abstraction : il n'implémente pas directement les fonctions réseau, mais fournit un ensemble de mécanismes, d'agents et de plugins permettant d'intégrer différentes technologies réseau. Son architecture repose sur un système de plugins offrant diverses implémentations réseau tout en conservant une API cohérente et de haut niveau. Des extensions d'API permettent également de gérer des fonctions avancées telles que la sécurité, la conformité, la supervision et le dépannage. (Bakshi, 2015)

Neutron reproduit le fonctionnement d'un réseau physique à travers des abstractions virtualisées telles que les sous-réseaux et les ports. Les sous-réseaux correspondent à des blocs d'adresses IP comprenant une passerelle, une liste de serveurs DNS et des routes spécifiques vers certains hôtes. Ces sous-réseaux constituent un pool d'adresses IP à partir duquel Neutron attribue des adresses aux machines virtuelles. (Bakshi, 2015)

Le port représente un point d'attachement virtuel associé à chaque machine virtuelle. Chaque port reçoit une adresse IP ainsi qu'une adresse MAC provenant du pool du sous-réseau. Si nécessaire, les utilisateurs peuvent demander des adresses IP spécifiques via des appels API. (Bakshi, 2015)

OpenStack Neutron repose sur une architecture composée de plusieurs types d'agents spécialisés. L'agent de plugin, installé sur les nœuds de calcul ou de réseau, assure la communication avec l'implémentation réseau choisie. L'agent DHCP attribue automatiquement des adresses IP aux instances via le protocole DHCP, tandis que l'agent L3 (Layer 3) implémente les fonctions de routage en utilisant la pile IP de Linux et les règles iptables, permettant notamment de relier différents réseaux de couche 2 via des routeurs. L'agent ML2 (Modular Layer 2) fournit un cadre modulaire pour les plugins de couche 2 et remplace progressivement les anciens plugins monolithiques. Pour compléter son fonctionnement, Neutron utilise plusieurs mécanismes complémentaires : les network namespaces assurent l'isolation des réseaux et empêchent le chevauchement des sous-réseaux entre différents locataires grâce aux espaces de noms Linux ; les **Floating IPs** sont des adresses IP publiques dynamiquement attribuables aux machines virtuelles, avec traduction d'adresses

(NAT) assurée par l'agent L3 ; et les Security Groups permettent de définir des règles de sécurité pour le trafic réseau, en précisant le type et la direction, chaque port étant associé à un groupe de sécurité dont les règles sont appliquées via iptables. (Bakshi, 2015)

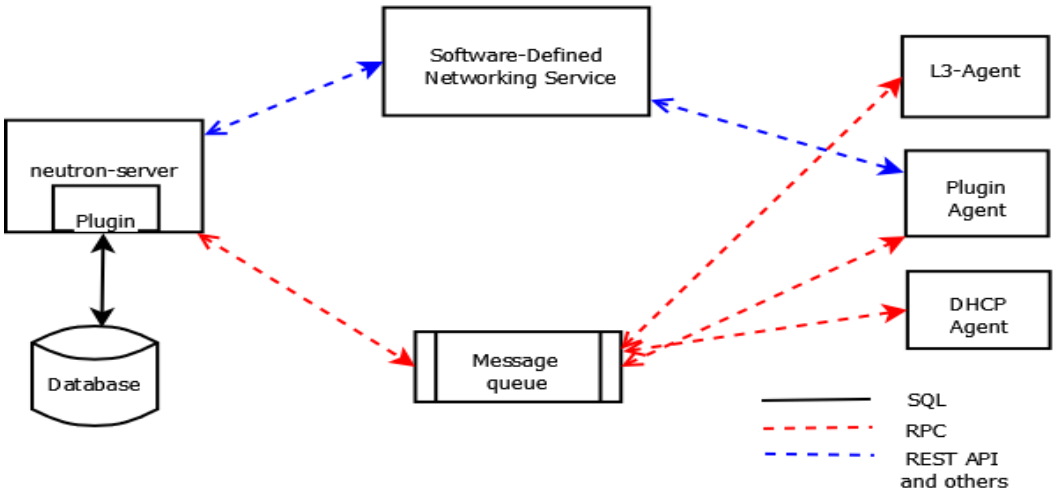

Figure 11 architecture_ neutron (Docs, consultez 2/3/2026)

- **Service stockage bloc (Cinder) :**

Le service Block Storage permet la gestion du stockage de blocs virtualisé devices sur plusieurs systèmes de stockage back-end, le démon cinder-scheduler est représentable pour Planification de volumes virtuels sur des hôtes de stockage physiques. (Yao, Papapanagiotou, & GriAffTh, Consultez le 5/4/2026)

L'algorithme de planification par défaut dans Cinder est appelé Capacité disponible et il est basé sur une procédure de filtrage/pondération, le planificateur fonctionne en trois étapes au début lorsqu'il est nouveau La demande de volume est disponible dans le planificateur en premier filtre les hôtes qui ont insuAffvient ressources comme la capacité ou trop de charge de travail pour répondre aux besoins de la demande puis la planificateur calcule un poids pour chacun des hôtes restants en fonction de la disponibilité capacité et les trie en fonction de ce poids dans un ordre croissant, le plus léger sera en haut, l'hôte qui a le moins de poids est choisi comme le meilleur candidat pour servir le nouvelle requête. (Yao, Papapanagiotou, & GriAffTh, Consultez le 5/4/2026)

- **Service stockage objet (Swift) :**

OpenStack Swift est un système de stockage d'objets distribué, hautement disponible et cohérent, pouvant fonctionner de manière autonome ou être intégré à la plateforme OpenStack. Il est conçu pour stocker de grandes quantités de données de façon fiable, sécurisée et économique grâce à un mécanisme de stockage redondant et évolutif. Contrairement aux systèmes de fichiers classiques utilisant une hiérarchie de répertoires et le stockage en blocs, Swift gère les données sous forme d'objets, chaque objet comprenant les données elles-mêmes, des métadonnées variables et un identifiant unique mondial. Les utilisateurs peuvent téléverser et télécharger des objets via l'API RESTful. Les objets sont organisés en conteneurs, similaires aux répertoires, mais ceux-ci ne peuvent pas être imbriqués. Chaque utilisateur dispose d'un compte Swift pouvant contenir plusieurs conteneurs. Pour gérer les comptes, conteneurs et objets, Swift utilise trois types de serveurs : le Account Server, qui gère les comptes et les conteneurs associés ; le Container Server, qui vérifie l'existence des objets dans les conteneurs ; et le Object Server, qui stocke et récupère les objets sur le serveur physique approprié. Lorsqu'un utilisateur fait une requête via le Proxy Server, celui-ci consulte le Account Server pour identifier les conteneurs, vérifie via le Container Server la présence de l'objet et utilise le Object Server pour récupérer l'objet. Le contrôle d'accès est assuré par les Listes de Contrôle d'Accès (ACL), à deux niveaux : au niveau du compte, permettant d'accorder un accès global à d'autres utilisateurs, et au niveau du conteneur, permettant de définir des permissions spécifiques telles que la lecture, l'écriture ou le listage des objets. (Biswas, Patwa, & Sandhu, 2015)

- **Service images (Glance) :**

OpenStack Glance est le service central de gestion des images dans l'Infrastructure en tant que Service (IaaS) d'OpenStack. Il permet de gérer les images de disques et de serveurs ainsi que leurs métadonnées associées, que ce soit pour les utilisateurs ou pour d'autres composants comme Nova. Glance prend en charge le stockage des images dans différents dépôts, y compris le stockage en bloc via Cinder, assure des processus périodiques pour maintenir la mise en cache, la cohérence et la disponibilité des images, et permet leur réplication afin de garantir leur accessibilité à travers différents services et sites. Les principaux composants de Glance incluent : Glance API, qui reçoit les appels API pour la découverte, la récupération et le stockage des images et fournit l'interface principale pour les utilisateurs et les services OpenStack ; Glance Registry, un service interne qui stocke, traite et récupère les métadonnées des images, telles que

leur taille ou type ; la base de données, qui contient l'ensemble des métadonnées (souvent MySQL ou SQLite) ; le référentiel de stockage, qui conserve les fichiers d'images et peut utiliser différents types de dépôts comme Swift, Cinder, HTTP ou Amazon S3, certains pouvant être en lecture seule ; et enfin le service de définition de métadonnées, qui fournit une API commune permettant aux administrateurs, services et utilisateurs de définir des métadonnées personnalisées pour divers types de ressources telles que les images, artefacts, volumes, saveurs et agrégats, incluant la création de nouvelles clés de propriété, descriptions, contraintes et types associés. (Docs, consultez 2/3/2026)

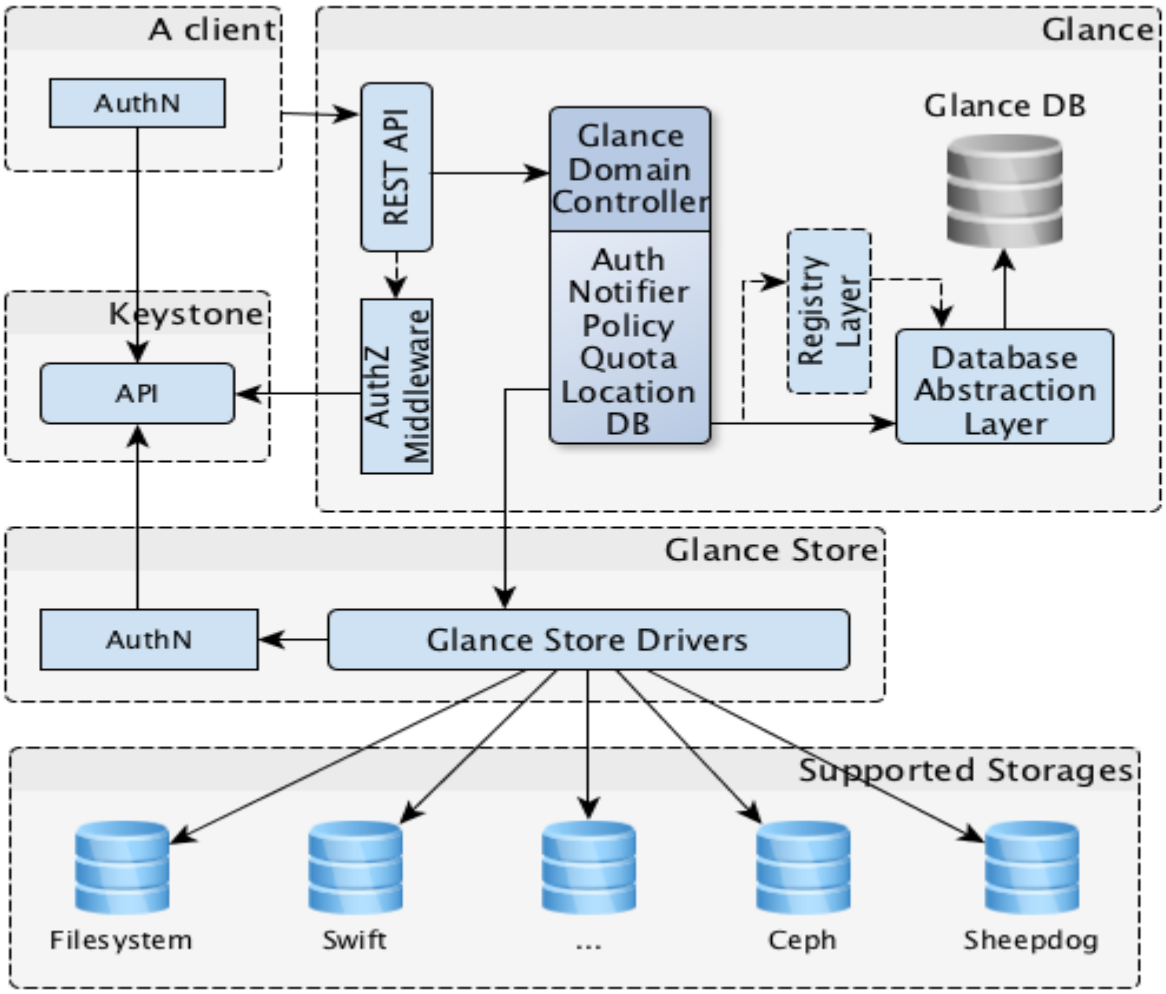

Figure 12architecture_ glance (Docs, consultez 2/3/2026)

- **Service identité (Keystone) :**

OpenStack Keystone (Docs, consultez 2/3/2026)constitue le service central d'authentification, d'autorisation et de gestion des identités au sein de la plateforme OpenStack. Il assure la sécurité et le contrôle d'accès pour l'ensemble des composants et des utilisateurs du cloud. Keystone

prend en charge plusieurs mécanismes d'authentification, notamment l'identification par nom d'utilisateur et mot de passe, les systèmes basés sur des jetons (tokens), l'authentification par paires de clés publique/privée, ainsi que l'intégration avec des services d'annuaire tels que Lightweight Directory Access Protocol (LDAP). Toutefois, son fonctionnement repose principalement sur un système de jetons. Lorsqu'un utilisateur souhaite accéder à un service OpenStack, il doit d'abord s'authentifier auprès de Keystone afin d'obtenir un jeton représentant son identité et les rôles qui lui sont attribués. Ce jeton, structuré au format JSON, contient des informations telles que l'identifiant de l'utilisateur, la méthode d'authentification utilisée, la date d'expiration, le catalogue des services disponibles ainsi que le projet ou domaine concerné. (Le Pompe, 2015) Pour effectuer des opérations courantes, comme la création d'une machine virtuelle ou le téléchargement d'une image, le jeton doit être étendu à un projet spécifique afin de vérifier les autorisations associées. Keystone maintient également un catalogue répertoriant tous les services déployés dans le cloud, chacun étant accessible via un ou plusieurs points de terminaison (endpoints). Lorsqu'un utilisateur présente son jeton à un service, celui-ci vérifie sa validité avant d'autoriser l'accès. En centralisant la gestion des identités, des rôles, des projets et des permissions, Keystone constitue ainsi un élément fondamental garantissant la sécurité et la cohérence du système OpenStack. (Bakshi, 2015)

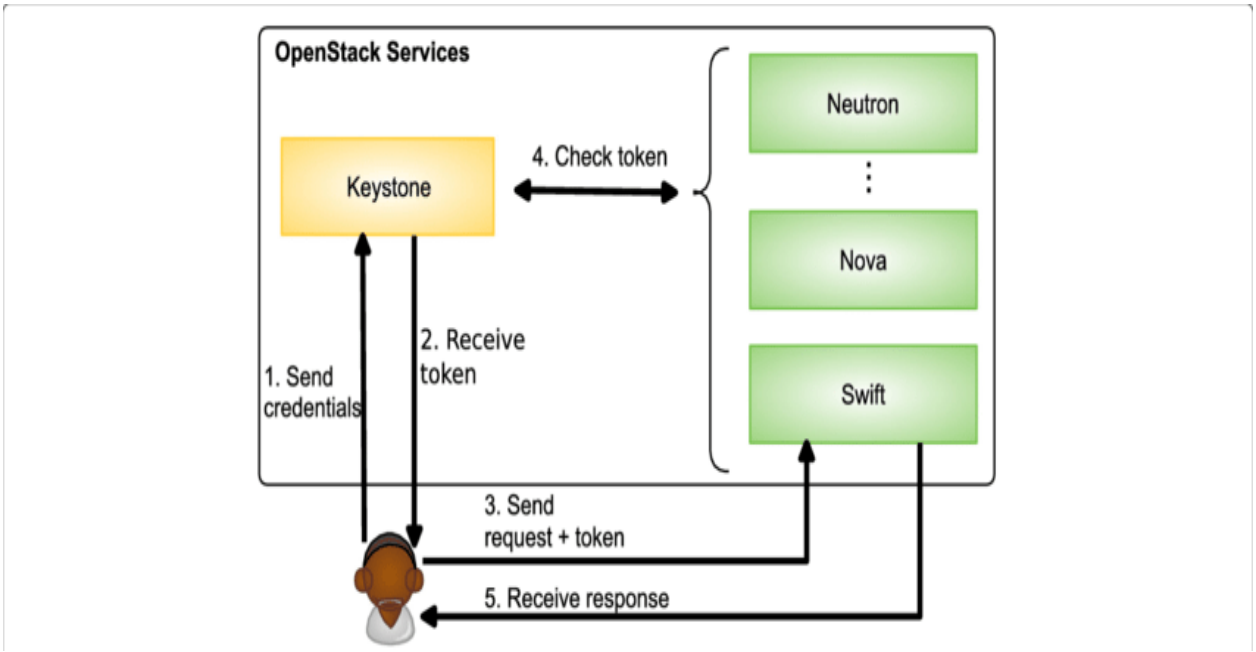

Figure 13architecture_ keystone (Docs, consultez 2/3/2026)

- **Interface web (Horizan) :**

La gestion d'un environnement OpenStack via l'interface en ligne de commande offre à l'administrateur un contrôle complet et précis sur l'infrastructure cloud. Toutefois, l'utilisation d'une interface graphique facilite considérablement l'administration des ressources et la gestion des instances. OpenStack Horizon, également appelé OpenStack Dashboard, fournit cette interface graphique conviviale. Il s'agit d'un service web fonctionnant généralement sur un serveur Apache HTTP Server et reposant sur l'interface Python Web Server Gateway Interface (WSGI) ainsi que sur le framework web Django, reconnu pour son développement rapide et structuré d'applications web. Grâce à Horizon, les utilisateurs et administrateurs peuvent gérer les projets, les utilisateurs, les réseaux, les volumes et les machines virtuelles à travers une interface intuitive, simplifiant ainsi l'exploitation quotidienne de l'environnement OpenStack. (Breu, Guggenbichler, & Wollmann, 2008)

3. Comparaison avec des solution :

Tableau 3 Comparaison des plateformes OpenStack, VMware, Azure Stack et Proxmox.

Critère	OpenStack	VMware	Microsoft Azure Stack	Proxmox
Type de solution	Plateforme Cloud IaaS Open Source	Plateforme de virtualisation propriétaire	Extension hybride de Microsoft Azure	Plateforme de virtualisation Open Source
Licence	Open Source (Apache 2.0)	Propriétaire (licence payante)	Propriétaire (Microsoft)	Open Source (AGPL v3)
Coût	Gratuit	Élevé	Élevé	Gratuit
Niveau de complexité	Élevée	Moyenne	Élevée	Faible à moyenne
Architecture	Modulaire (Nova, Neutron, Cinder, etc.)	Centralisée autour de vCenter	Intégrée à l'écosystème Azure	Intégrée (KVM + LXC)
Virtualisation	KVM, Xen, etc.	ESXi (hyperviseur VMware)	Hyper-V	KVM
Type de Cloud	Privé, Public, Hybride	Principalement privé	Hybride	Privé
Scalabilité	Très élevée (grandes infrastructures)	Élevée	Élevée	Moyenne
Public cible	Grandes entreprises, fournisseurs Cloud	Entreprises	Entreprises utilisant Azure	PME, laboratoires, universités

4.Conclusion de chapitre :

Le chapitre a présenté l'architecture et les principaux composants d'OpenStack, en mettant l'accent sur la gestion des ressources de calcul (Nova), du stockage (Cinder, Swift), du réseau (Neutron) et de l'identité (Keystone), ainsi que sur l'interface utilisateur Horizon. OpenStack se distingue par sa modularité, sa flexibilité et sa capacité à prendre en charge des infrastructures cloud à grande échelle, offrant ainsi une solution IaaS complète et évolutive. Ses services interdépendants et ses APIs ouvertes permettent une intégration facile avec d'autres systèmes et une personnalisation avancée selon les besoins des utilisateurs et des entreprises.

Cependant, cette puissance présente certaines limites : la complexité de déploiement et d'administration nécessite une expertise technique avancée, la performance et la disponibilité des services dépendent fortement de la qualité de l'infrastructure physique sous-jacente, et la maintenance des différents composants peut s'avérer délicate. Malgré ces contraintes, OpenStack reste une plateforme incontournable pour la conception et la gestion de clouds privés, publics ou hybrides. Ces points serviront de base pour le chapitre suivant, consacré à la mise en œuvre pratique d'un cloud OpenStack.

Chapitre 03

Conception d’un Cloud privé

1. Analyse des besoins :

1.1 Besoins matériels (serveurs, stockage, réseau) :

La conception d'une infrastructure de cloud privé repose avant tout sur une analyse rigoureuse des besoins matériels, afin de garantir la cohérence entre les exigences fonctionnelles du système et les ressources physiques disponibles. Dans le cadre de ce projet, l'infrastructure a été dimensionnée selon une architecture multi-noeuds minimale, composée de deux nœuds physiques distincts, chacun assumant un rôle fonctionnel précis au sein de la plateforme Open Stack.

a) Infrastructure de calcul

L'architecture retenue repose sur le principe de séparation des responsabilités entre le plan de contrôle (*control plane*) et le plan de données (*data plane*), conformément aux bonnes pratiques des déploiements OpenStack en environnement de production.

Le nœud contrôleur (Controller, 192.168.10.11) constitue le point central de l'infrastructure. Il héberge l'intégralité des services d'orchestration et de gestion : authentification et gestion des identités (Keystone), gestion du catalogue d'images (Glance), orchestration du calcul (Nova API), gestion du réseau virtuel (Neutron Server), ainsi que l'interface d'administration web (Horizon). Ce nœud joue le rôle de plan de contrôle et ne participe pas directement à l'exécution des charges de travail virtualisées.

Le nœud de calcul (Compute1, 192.168.10.12) constitue quant à lui le plan de données de l'infrastructure. Il est responsable de l'instanciation et de l'exécution des machines virtuelles grâce à l'hyperviseur KVM (Kernel-based Virtual Machine) et au service Nova Compute. Ce nœud communique en permanence avec le nœud contrôleur via des files de messages RabbitMQ pour recevoir et exécuter les requêtes d'approvisionnement des ressources.

b) Infrastructure réseau

La connectivité entre les nœuds est assurée par une liaison Ethernet dédiée, constituant un réseau local d'infrastructure (*underlay network*). Deux plans d'adressage logiques ont été définis :

- **Le réseau de gestion** (192.168.10.0/24) supporte les communications interservices OpenStack ainsi que l'administration SSH des nœuds. Il constitue le canal de signalisation de l'infrastructure.
- **Le réseau privé des instances** (10.10.10.0/24), géré par le service Neutron en conjonction avec Open vSwitch, forme le réseau *overlay* dédié aux machines virtuelles.

Il permet l'isolation du trafic applicatif du trafic d'administration, conformément aux principes de segmentation réseau en environnement cloud.

c) Infrastructure de stockage

Le stockage adopté dans ce projet repose sur une approche de stockage local (*local storage*), adaptée aux contraintes d'un environnement de démonstration à ressources limitées. Deux niveaux de stockage ont été identifiés :

- Le stockage des images système est pris en charge par le service Glance sur le nœud contrôleur, dans le répertoire /var/lib/glance/images/. Les images disponibles incluent Ubuntu Noble (600 MB) et CirrOS (20 MB), utilisées respectivement pour les instances applicatives et les tests de validation.
- Le stockage des disques d'instances est assuré par le service Nova sur le nœud de calcul, dans /var/lib/nova/instances/. Chaque instance dispose d'un disque virtuel au format QCOW2, permettant la gestion efficace de l'espace disque par allocation dynamique.

Il convient de noter que cette architecture de stockage, bien qu'adaptée au contexte de ce projet, présente des limitations en termes de haute disponibilité et de partage des données entre nœuds. Ces limitations sont discutées dans la section dédiée aux perspectives d'évolution.

1.2 Besoins logiciels

Le déploiement d'une infrastructure de cloud privé basée sur OpenStack nécessite un ensemble cohérent de composants logiciels, organisés en couches interdépendantes. L'analyse des besoins logiciels a été conduite selon une approche descendante, allant du système d'exploitation de base jusqu'aux services applicatifs hébergés dans le cloud.

a) Système d'exploitation

Les deux nœuds de l'infrastructure fonctionnent sous **Ubuntu 24.04 LTS** (Noble Numbat). Bien que la version Desktop ait été utilisée dans ce projet, celle-ci est pleinement compatible avec Kolla-Ansible et les services OpenStack, tout en offrant un environnement graphique facilitant l'administration et la démonstration de l'infrastructure. Ubuntu 24.04 LTS constitue la plateforme de référence pour le déploiement de Kolla-Ansible dans sa version 2024.2, garantissant ainsi la cohérence entre les dépendances système et les conteneurs Docker utilisés.

b) Plateforme de conteneurisation

L'ensemble des services OpenStack est déployé sous forme de **conteneurs Docker**, conformément à l'approche adoptée par Kolla-Ansible. Docker Engine assure l'isolation, la portabilité et la gestion du cycle de vie de chaque service. Cette architecture conteneurisée présente plusieurs avantages majeurs :

- **Isolation des services** : chaque composant OpenStack s'exécute dans un environnement cloisonné, limitant les risques d'interférences entre services.
- **Facilité de mise à jour** : le remplacement d'un service se réduit au remplacement de son image Docker, sans impact sur les autres composants.
- **Reproductibilité** : le déploiement est entièrement défini par des fichiers de configuration versionables, garantissant la cohérence entre environnements.

c) Outil de déploiement — Kolla-Ansible

Kolla-Ansible constitue le moteur de déploiement de l'infrastructure OpenStack. Il s'agit d'un projet officiel de l'écosystème OpenStack qui combine la puissance d'Ansible pour l'automatisation de la configuration et les images Docker Kolla pour la conteneurisation des services. Kolla-Ansible prend en charge l'installation, la configuration, la mise à jour et la maintenance de l'ensemble des composants OpenStack de manière déclarative et reproductible. Dans le cadre de ce projet, la version **2024.2 (Dalmatian)** de Kolla-Ansible a été utilisée, déployant les images officielles quay.io/openstack.kolla sur Ubuntu 24.04 LTS.

d) Services OpenStack déployés

Les services OpenStack suivants ont été déployés et configurés dans le cadre de ce projet :

Tableau 4 Services OpenStack déployés

Service	Composant OpenStack	Rôle fonctionnel
Identité	Keystone	Authentification et autorisation
Calcul	Nova	Gestion et instanciation des Machines virtuelles (VM)
Réseau	Neutron + Open vSwitch	Réseau virtuel et SDN
Images	Glance	Catalogue et stockage des images
Interface web	Horizon	Administration graphique
Messagerie	RabbitMQ	Communication inter-services
Base de données	MariaDB	Persistance des données des services
Cache	Memcached	Optimisation des performances
Supervision	Fluentd + Cron	Journalisation et tâches planifiées

e) Logiciels applicatifs

Deux applications web ont été développées en Python (bibliothèque standard http. Server) et déployées au sein de la machine virtuelle app-VM tournant sous Ubuntu Noble :

- **Application de gestion des soutenances** : application web légère utilisant un fichier **JSON** comme mécanisme de persistance, exposée sur le port 8090, hébergée dans /home/appuser/soutenance-app/.
- **Application de planning universitaire** : application web avec base de données SQLite, exposée sur le port 8092, hébergée dans /home/appuser/planning-app/.

La communication entre les clients externes et les applications hébergées dans le cloud est assurée par deux **proxies HTTP** développés en Python et déployés sur le nœud contrôleur (Controller), exploitant les *network namespaces* de Neutron via la commande ip netns exec pour acheminer les requêtes vers les instances virtuelles à l'adresse 10.10.10.35.

La persistance et la disponibilité de l'ensemble des services — applications et proxies — sont garanties par **systemd**, qui assure le démarrage automatique à chaque redémarrage des nœuds, éliminant ainsi toute intervention manuelle post-reboot.

1.3 Contraintes organisationnelles

La réalisation de ce projet s'est inscrite dans un cadre organisationnel spécifique, caractérisé par un ensemble de contraintes qu'il a été nécessaire d'identifier et de prendre en compte dès la phase de conception, afin de garantir la faisabilité et la cohérence de l'infrastructure déployée.

a) Contraintes matérielles et budgétaires

L'une des contraintes majeures de ce projet réside dans la limitation des ressources matérielles disponibles. En l'absence d'un environnement de serveurs dédiés, l'infrastructure a été déployée sur deux ordinateurs portables personnels, imposant ainsi des compromis en termes de capacité de calcul, de mémoire vive et d'espace de stockage. Cette contrainte a directement influencé les choix architecturaux, notamment l'adoption d'une topologie multi-noeuds minimale à deux nœuds, et l'utilisation d'un stockage local plutôt qu'une solution de stockage partagé (Ceph, NFS).

Par ailleurs, le disque du nœud de calcul (Compute1) présentait un taux d'occupation critique de **96%** en cours de projet, ce qui a nécessité des opérations de nettoyage préalables (suppression des journaux système obsolètes, purge du cache Snap) avant de pouvoir réaliser avec succès un snapshot de l'instance app-VM. Ce snapshot, nommé app-VM-backup-20260503, a été créé au format QCOW2 et est disponible dans le catalogue Glance, constituant ainsi une sauvegarde complète de l'environnement applicatif déployé.

b) Contraintes temporelles

Le projet a été réalisé dans le cadre d'un mémoire de fin d'études de Master 2, impliquant une contrainte temporelle stricte. Cette limitation a conduit à prioriser les fonctionnalités essentielles de la plateforme cloud déploiement des services OpenStack, instanciation des machines virtuelles, déploiement des applications au détriment de fonctionnalités avancées telles que la haute disponibilité, l'équilibrage de charge (*load balancing*) et la supervision (*monitoring*), qui sont identifiées comme perspectives d'évolution futures.

c) Contraintes techniques

L'utilisation d'ordinateurs portables comme infrastructure physique a engendré plusieurs contraintes techniques spécifiques :

- **Connectivité réseau** : la liaison entre les deux nœuds repose sur un câble Ethernet direct, sans équipement réseau intermédiaire (switch, routeur), ce qui limite la scalabilité de l'infrastructure mais garantit une latence minimale entre les nœuds.
- **Absence de redondance** : l'architecture à nœud unique de calcul constitue un point de défaillance unique (*Single Point of Failure*), rendant l'ensemble des instances virtuelles indisponibles en cas de défaillance du nœud Compute1.
- **Accès aux instances** : en l'absence d'un réseau externe routable, l'accès aux machines virtuelles depuis l'extérieur du cloud est assuré par des proxies HTTP développés spécifiquement pour ce projet, exploitant les *network namespaces* de Neutron comme mécanisme de contournement.
- **Persistance des données** : les applications déployées dans les instances virtuelles étaient initialement stockées dans le répertoire temporaire /tmp, exposant les données à une perte lors de chaque redémarrage. Ce problème a été résolu par la migration des applications vers /home/appuser/ et la mise en place de services **systemd** assurant le démarrage automatique des applications à chaque redémarrage de l'instance.

d) Contraintes de sécurité

Dans le cadre de ce projet, les contraintes de sécurité ont été volontairement simplifiées pour se concentrer sur les aspects fonctionnels de la plateforme. Ainsi, les communications entre services s'effectuent en HTTP plutôt qu'en HTTPS, et les politiques de contrôle d'accès (*security groups*) ont été configurées de manière permissive. Ces aspects constituent des axes d'amélioration prioritaires dans une perspective de mise en production réelle.

2 Architecture proposée

2.1 Architecture physique

L'architecture physique de l'infrastructure cloud privée repose sur deux nœuds physiques interconnectés, chacun assumant un rôle fonctionnel bien défini au sein de la plateforme OpenStack. Cette architecture, bien que minimale, respecte les principes fondamentaux de séparation des responsabilités préconisés par les déploiements OpenStack en environnement multi-noeuds.

a) Nœud contrôleur (Controller)

Le nœud contrôleur constitue le cerveau de l'infrastructure cloud. Il héberge l'intégralité des services de gestion et d'orchestration d'OpenStack, ainsi que les composants de support nécessaires à leur fonctionnement :

- **Système d'exploitation** : Ubuntu 24.04 LTS Desktop
- **Host Name** : Controller
- **Adresse IP** : 192.168.10.11
- **Interface réseau** : enp1s0
- **Services hébergés**: Keystone, Glance, Nova API, Neutron Server, Horizon, RabbitMQ, MariaDB, Memcached, HAProxy, Fluentd, Cron
- **Services applicatifs** : proxy HTTP soutenance (port 8090), proxy HTTP planning (port 8093), gérés par systemd

b) Nœud de calcul (Compute)

Le nœud de calcul constitue le plan de données de l'infrastructure. Il est exclusivement dédié à l'exécution des machines virtuelles et à la gestion des ressources de virtualisation :

- **Système d'exploitation** : Ubuntu 24.04 LTS Desktop
- **Host Name** : Compute1
- **Adresse IP** : 192.168.10.12
- **Interface réseau** : enp0s31f6
- **Services hébergés**: Nova Compute, Nova LibVirt, Neutron OpenVSwitch Agent, Open vSwitch
- **Instances hébergées** : app-VM (IP : 10.10.10.35), vol-test-01 (IP : 10.10.10.114)

c) Interconnexion physique

Les deux nœuds sont interconnectés via un câble Ethernet direct (liaison point-à-point), formant un réseau local dédié à l'infrastructure cloud. Cette topologie, bien que simple, garantit une

latence minimale et une bande passante maximale entre les nœuds, conditions essentielles au bon fonctionnement des communications inter-services OpenStack.

2.2 Architecture logique

L'architecture logique décrit l'organisation des composants logiciels et leurs interactions au sein de la plateforme OpenStack. Elle s'articule autour de trois plans fonctionnels distincts :

a) Plan de contrôle (Control Plane)

Le plan de contrôle regroupe l'ensemble des services responsables de la gestion, de l'orchestration et de la supervision de l'infrastructure cloud. Il est intégralement hébergé sur le nœud contrôleur et comprend :

- **Keystone** : point d'entrée unique pour l'authentification et l'autorisation de tous les services et utilisateurs de la plateforme. Il délivre des tokens d'accès utilisés par l'ensemble des composants OpenStack.
- **Nova API** : interface de gestion du cycle de vie des instances virtuelles (création, démarrage, arrêt, suppression). Elle reçoit les requêtes des utilisateurs et les transmet au nœud de calcul via RabbitMQ.
- **Neutron Server** : gestionnaire du réseau virtuel, responsable de la création et de la gestion des réseaux, sous-réseaux et ports virtuels. Il s'appuie sur Open vSwitch pour l'implémentation du plan de données réseau.
- **Glance** : service de gestion du catalogue d'images, permettant le stockage et la récupération des images système utilisées lors de l'instanciation des machines virtuelles.
- **Horizon** : interface web d'administration, offrant une vue unifiée de l'ensemble des ressources cloud et permettant leur gestion via un navigateur web.
- **RabbitMQ** : bus de messages assurant la communication asynchrone entre les différents services OpenStack.
- **MariaDB** : base de données relationnelle centralisant la persistance des états et configurations de l'ensemble des services OpenStack.

b) Plan de données (Data Plane)

Le plan de données est hébergé sur le nœud de calcul et regroupe les composants responsables de l'exécution effective des charges de travail virtualisées :

- **Nova Compute** : agent de calcul responsable de l'instanciation et de la gestion des machines virtuelles sur l'hyperviseur KVM via l'interface libvirt.
- **Neutron OVS Agent** : agent réseau implémentant les politiques réseaux définis par Neutron Server, en configurant les flux Open vSwitch sur le nœud de calcul.

- **KVM/libvirt** : hyperviseur de type 1 assurant la virtualisation matérielle des instances. Chaque instance dispose d'un disque virtuel au format QCOW2 stocké dans /var/lib/nova/instances/.

c) Plan applicatif

Le plan applicatif représente la couche la plus haute de l'architecture, hébergée au sein des instances virtuelles :

- **App-VM** (10.10.10.35) : instance virtuelle Ubuntu Noble hébergeant les deux applications web développées dans le cadre de ce projet. Les applications sont gérées par systemd et persistent dans /home/appuser/.
- **Proxies HTTP** : déployés sur le nœud contrôleur, ils assurent la passerelle entre le réseau physique et le réseau virtuel des instances, en exploitant les *network namespaces* Neutron (qdhcp-735fa9f3-2132-43b8-9003-f5dcb64e7525).

2.3 Plan d'adressage IP

Le plan d'adressage IP adopté dans ce projet définit deux espaces d'adressage distincts, corresp ondant aux deux réseaux logiques de l'infrastructure :

Réseau	Plage d'adresses	Masque	Rôle
Réseau de gestion	192.168.10.0/24	255.255.255.0	Communication inter-nœuds et administration
Réseau privé des instances	10.10.10.0/24	255.255.255.0	Communication inter-Machines virtuelles (VM)

Tableau 5 Plan d'adressage IP

Hôte	Adresse IP	Réseau	Rôle
Controller	192.168.10.11	Gestion	Nœud contrôleur
Compute1	192.168.10.12	Gestion	Nœud de calcul
App-VM	10.10.10.35	Privé	Instance applicative
Vol-test-01	10.10.10.114	Privé	Instance de test

Tableau 6 tableau des hôtes

2.4 Schéma réseau

L'architecture réseau de l'infrastructure cloud privée peut être représentée selon deux niveaux d'abstraction complémentaires :

a) Réseau physique (Underlay)

La couche physique repose sur une liaison Ethernet point-à-point entre les deux nœuds, sur le réseau 192.168.10.0/24. Cette liaison constitue le support physique de toutes les communications de l'infrastructure, qu'il s'agisse des échanges inter-services OpenStack ou du trafic de gestion des instances virtuelles.

b) Réseau virtuel (Overlay)

La couche virtuelle est gérée par Neutron en conjonction avec Open vSwitch. Elle implémente un réseau privé (private-net, 10.10.10.0/24) isolé du réseau physique, au sein duquel les instances virtuelles communiquent entre elles. L'accès aux applications hébergées dans ce réseau depuis le réseau physique est rendu possible grâce aux proxies HTTP déployés sur le nœud contrôleur, qui exploitent les *network namespaces* de Neutron comme passerelle entre les deux plans réseau.

3. Choix technologiques

3.1 Hyperviseur KVM

La virtualisation constitue le fondement technique de toute infrastructure cloud. Le choix de l'hyperviseur est donc une décision architecturale majeure, dont dépendent directement les performances, la compatibilité et la maintenabilité de la plateforme.

Dans le cadre de ce projet, l'hyperviseur retenu est KVM (Kernel-based Virtual Machine), solution de virtualisation intégrée nativement au noyau Linux depuis la version 2.6.20. KVM transforme le noyau Linux en un hyperviseur de type 1 (*bare-metal*), offrant des performances proches du natif grâce à l'exploitation des extensions de virtualisation matérielle Intel VT-x et AMD-V.

Justification du choix :

- **Intégration native avec OpenStack** : KVM constitue l'hyperviseur de référence pour les déploiements OpenStack, bénéficiant d'un support natif et d'une intégration optimale avec Nova via l'interface libvirt.
- **Performances** : en tant qu'hyperviseur de type 1, KVM offre des performances supérieures aux solutions de virtualisation de type 2 (VirtualBox, VMware Workstation), grâce à l'accès direct aux ressources matérielles.
- **Maturité et stabilité** : KVM est une technologie éprouvée, utilisée en production par les principaux fournisseurs de cloud public (Google Cloud, AWS via KVM-based Nitro).
- **Licence open source** : KVM est distribué sous licence GPL, garantissant la liberté d'utilisation et la transparence du code source.

- **Support de libvirt** : l'interface libvirt fournit une couche d'abstraction standardisée au-dessus de KVM, utilisée par Nova Compute pour gérer le cycle de vie des instances virtuelles de manière uniforme et portable.

La vérification du support KVM sur les nœuds de calcul a été effectuée en amont du déploiement, via la commande kvm-ok, confirmant la disponibilité des extensions de virtualisation matérielle nécessaires.

3.2 Base de données MariaDB

La persistance des états et des configurations de l'ensemble des services OpenStack repose sur un système de gestion de bases de données relationnelles. Le choix s'est porté sur **MariaDB**, fork communautaire de MySQL, déployé sous forme de conteneur Docker par Kolla-Ansible.

Justification du choix :

- **Compatibilité native avec OpenStack** : MariaDB est la base de données relationnelle officiellement supportée et recommandée par le projet OpenStack pour les déploiements Kolla-Ansible.
- **Performances et fiabilité** : MariaDB offre des performances supérieures à MySQL sur les charges de travail transactionnelles, grâce à des optimisations du moteur de stockage InnoDB et à l'introduction de nouvelles fonctionnalités comme le *thread pool*.
- **Support du clustering** : MariaDB Galera Cluster permet, dans des déploiements plus avancés, la mise en place d'une base de données hautement disponible en configuration multi-maître, constituant ainsi une perspective d'évolution naturelle pour ce projet.
- **Licence open source** : MariaDB est distribué sous licence GPL, dans la continuité de l'héritage open source de MySQL.

Chaque service OpenStack dispose de sa propre base de données au sein de l'instance MariaDB, garantissant l'isolation des données entre composants. Les bases de données créées lors du déploiement incluent notamment Keystone, nova, neutron, glance et horizon.

3.3 Service de messagerie RabbitMQ

La communication asynchrone entre les services OpenStack constitue un aspect fondamental de l'architecture distribuée de la plateforme. Ce besoin est adressé par un système de messagerie (*message broker*), pour lequel le choix s'est porté sur **RabbitMQ**.

Justification du choix :

- **Standard de facto pour OpenStack** : RabbitMQ est le broker de messages officiellement supporté et utilisé par défaut dans les déploiements OpenStack,

bénéficiant d'une intégration profonde avec la bibliothèque de communication inter-services **oslo Messaging**.

- **Protocole AMQP** : RabbitMQ implémente le protocole AMQP (Advanced Message Queuing Protocol), standard ouvert garantissant l'interopérabilité et la fiabilité des échanges de messages entre services.
- **Fiabilité des messages** : RabbitMQ garantit la livraison des messages grâce à des mécanismes d'accusé de réception (*acknowledgment*) et de persistance des files d'attente, assurant qu'aucune requête n'est perdue en cas de défaillance temporaire d'un service.
- **Découplage des services** : le recours à un bus de messages permet un découplage fort entre les producteurs et les consommateurs de messages, facilitant la scalabilité horizontale des composants OpenStack.

Dans l'architecture déployée, RabbitMQ joue un rôle central dans la communication entre Nova API (nœud contrôleur) et Nova Compute (nœud de calcul), permettant la transmission des requêtes d'instanciation et de gestion des machines virtuelles de manière asynchrone et fiable.

4 Modèle de sécurité

La sécurité constitue un aspect fondamental de toute infrastructure cloud, qu'elle soit publique ou privée. Dans le cadre de ce projet, un modèle de sécurité adapté aux contraintes d'un environnement de démonstration a été défini, tout en respectant les principes de base de la sécurisation des plateformes OpenStack. Ce modèle s'articule autour de trois axes principaux : la gestion des rôles et des identités, les politiques d'accès, et la segmentation réseau.

4.1 Gestion des rôles

OpenStack implémente un système de contrôle d'accès basé sur les rôles (RBAC Role-Based Access Control), géré par le service Keystone. Ce mécanisme permet de définir de manière granulaire les droits et privilèges accordés à chaque utilisateur ou service au sein de la plateforme.

a) Structure des rôles OpenStack

Dans le cadre de ce projet, la structure de rôles suivante a été adoptée :

Rôle	Périmètre	Privilèges
Admin	Plateforme globale	Accès total à l'ensemble des ressources et services OpenStack
Member	Projet (*tenant*)	Gestion des ressources au sein du projet assigné
Reader	Projet (*tenant*)	Consultation des ressources en lecture seule
Heat_stack_owner	Projet (*tenant*)	Gestion des stacks d'orchestration Heat
Manager	Projet (*tenant*)	Administration étendue au niveau du projet

Tableau 7 Structure des rôles OpenStack

b) Gestion des identités

Keystone assure la gestion centralisée des identités au sein de la plateforme. Il délivre des tokens d'accès à durée de vie limitée, utilisés par l'ensemble des services OpenStack pour authentifier les requêtes entrantes. Les credentials d'administration de la plateforme sont stockés dans le fichier /etc/kolla/admin-openrc.sh sur le nœud contrôleur, contenant les variables d'environnement nécessaires à l'authentification CLI :

```
OS_AUTH_URL=http://192.168.10.11:5000
OS_PROJECT_NAME=admin
OS_USERNAME=admin
OS_USER_DOMAIN_NAME=Default
OS_PROJECT_DOMAIN_NAME=Default
OS_IDENTITY_API_VERSION=3
```

c) Gestion des tokens Fernet

OpenStack Keystone utilise le mécanisme de tokens Fernet, solution légère et performante ne nécessitant pas de persistance en base de données. Les clés de chiffrement Fernet sont gérées par le service keystone-fernet, déployé sous forme de conteneur Docker sur le nœud contrôleur, et font l'objet d'une rotation périodique automatique pour limiter les risques de compromission.

4.2 Politiques d'accès

Les politiques d'accès définissent les règles d'autorisation appliquées à chaque opération effectuée sur la plateforme OpenStack. Elles sont implémentées à deux niveaux complémentaires.

a) Politiques OpenStack (policy. yaml)

Chaque service OpenStack dispose de son propre fichier de politique (policy. Yaml), définissant les règles d'autorisation pour chaque API. Dans le cadre de ce projet, les politiques par défaut fournies par Kolla-Ansible ont été conservées, accordant les privilèges suivants :

- Les opérations d'administration (création de flavors, upload d'images, gestion des réseaux) sont réservées au rôle admin.
- Les opérations courantes (création d'instances, gestion des volumes, consultation des ressources) sont accessibles aux rôles member et manager.
- La consultation des ressources est accessible au rôle reader en lecture seule.

b) Security Groups

Les *security groups* constituent le mécanisme de pare-feu virtuel d'OpenStack, implémenté par Neutron au niveau des ports virtuels de chaque instance. Ils définissent les règles de filtrage du trafic entrant (*ingress*) et sortant (*egress*) pour chaque machine virtuelle.

Dans le cadre de ce projet, un security group permissif a été configuré pour l'instance app-VM, autorisant l'ensemble du trafic entrant et sortant. Cette configuration, adaptée à un environnement de démonstration, devra être renforcée dans une perspective de mise en production, en appliquant le principe du moindre privilège (*least privilege*) : seuls les ports applicatifs nécessaires (8090, 8092) devraient être autorisés en entrée.

4.3 Pare-feu et segmentation réseau

La segmentation réseau constitue un pilier fondamental de la sécurité des infrastructures cloud, permettant d'isoler les différents plans de communication et de limiter la propagation des menaces en cas de compromission d'un composant.

a) Segmentation réseau

L'architecture réseau de ce projet implémente une segmentation en deux plans distincts :

- **Réseau de gestion** (192.168.10.0/24) : réseau dédié aux communications inter-services OpenStack et à l'administration SSH des nœuds. Ce réseau est physiquement isolé du réseau des instances, limitant ainsi les risques d'interférence entre le trafic de gestion et le trafic applicatif.
- **Réseau privé des instances** (10.10.10.0/24) : réseau virtuel géré par Neutron, hébergeant les machines virtuelles. Ce réseau est logiquement isolé du réseau physique grâce aux mécanismes de virtualisation réseau d'Open vSwitch, garantissant que le trafic inter-VM ne transite pas sur le réseau de gestion.

b) Isolation par network namespaces

Neutron exploite les **network namespaces** Linux pour assurer l'isolation réseau entre les différents projets (*tenants*) hébergés sur la plateforme. Dans ce projet, le namespace qdhcp-735fa9f3-2132-43b8-9003-f5dcb64e7525 est utilisé par le service DHCP pour fournir des adresses IP aux instances du réseau privé, tout en maintenant une isolation stricte avec le réseau de gestion.

c) Accès sécurisé aux nœuds

L'accès aux nœuds de l'infrastructure est sécurisé par **SSH** (Secure Shell), protocole de communication chiffré garantissant la confidentialité et l'intégrité des sessions d'administration. La communication entre le nœud contrôleur et le nœud de calcul s'effectue exclusivement via SSH, notamment pour les opérations de déploiement Kolla-Ansible et la gestion des conteneurs Docker.

d) Limites et perspectives

Il convient de noter que le modèle de sécurité implémenté dans ce projet présente plusieurs limitations inhérentes au contexte de démonstration :

- Les communications entre services s'effectuent en **HTTP** plutôt qu'en **HTTPS**, exposant potentiellement les données en transit à des risques d'interception.
- Les mots de passe des instances virtuelles (app123456) ne respectent pas les bonnes pratiques de complexité des credentials en environnement de production.
- L'absence de mécanisme de supervision de la sécurité (*SIEM*, *IDS*) limite la capacité de détection des incidents.

Ces limitations constituent des axes d'amélioration prioritaires identifiés dans les perspectives d'évolution du projet, notamment par l'activation du chiffrement TLS pour les communications inter-services et le renforcement des politiques de *security groups*.

Ce chapitre a présenté la conception de l'infrastructure de cloud privé déployée dans le cadre de ce projet. L'analyse des besoins a permis d'identifier les contraintes matérielles, logicielles et organisationnelles ayant guidé les choix architecturaux. L'architecture proposée, basée sur une topologie multi-noeuds à deux nœuds, respecte les principes fondamentaux de séparation des responsabilités entre plan de contrôle et plan de données, conformément aux recommandations du projet OpenStack.

Les choix technologiques retenus — KVM comme hyperviseur, MariaDB comme système de gestion de bases de données, et RabbitMQ comme broker de messages — sont tous des solutions open source matures, bénéficiant d'un support natif dans l'écosystème OpenStack et d'une large adoption en environnement de production.

Enfin, le modèle de sécurité mis en place, bien qu'adapté aux contraintes d'un environnement de démonstration, pose les bases d'une architecture sécurisée pouvant être renforcée dans une perspective de mise en production réelle.

Le chapitre suivant détaillera la phase d'implémentation de cette architecture, en présentant les étapes de déploiement, de configuration et de validation de l'infrastructure cloud privée.

Chapitre 04

Implémentation

1 Préparation de l'environnement

1.1 Installation des serveurs

La première étape de l'implémentation consiste en la préparation des serveurs physiques constituant l'infrastructure cloud. Les deux nœuds ont été configurés sous Ubuntu 24.04.4 LTS (Noble Numbat), noyau Linux 6.17.0, architecture x86-64. Le tableau suivant présente les caractéristiques matérielles détaillées de chaque nœud, telles que relevées lors de la phase d'initialisation :

Tableau 8 les caractéristiques matérielles de chaque nœud

Caractéristique	Nœud contrôleur	Nœud de calcul
Hostname	Controller	Compute1
Modèle	HP ProBook 450 G5	HP EliteBook 840 G3
Processeur	Intel Core i5-8250U @ 1.60GHz	Intel Core i5-6300U @ 2.40GHz
Cœurs / Threads	8 CPUs / 2 threads par cœur	4 CPUs / 2 threads par cœur
Mémoire RAM	15 Gi	15 Gi
Swap	8.0 Gi	8.0 Gi
Stockage	92 Go (36% utilisé)	38 Go (96% utilisé)
Système d'exploitation	Ubuntu 24.04.4 LTS	Ubuntu 24.04.4 LTS
Noyau Linux	6.17.0-14-generic	6.17.0-23-generic
Architecture	x86-64	x86-64

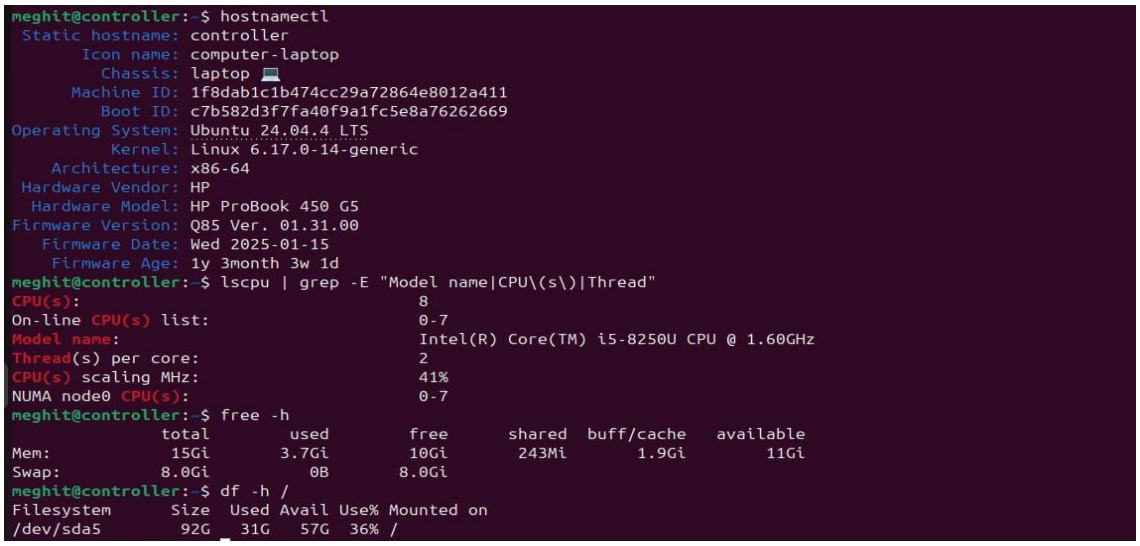

Figure 14Caractéristiques matérielles du nœud contrôleur (hostnamectl, lscpu, free -h, df -h

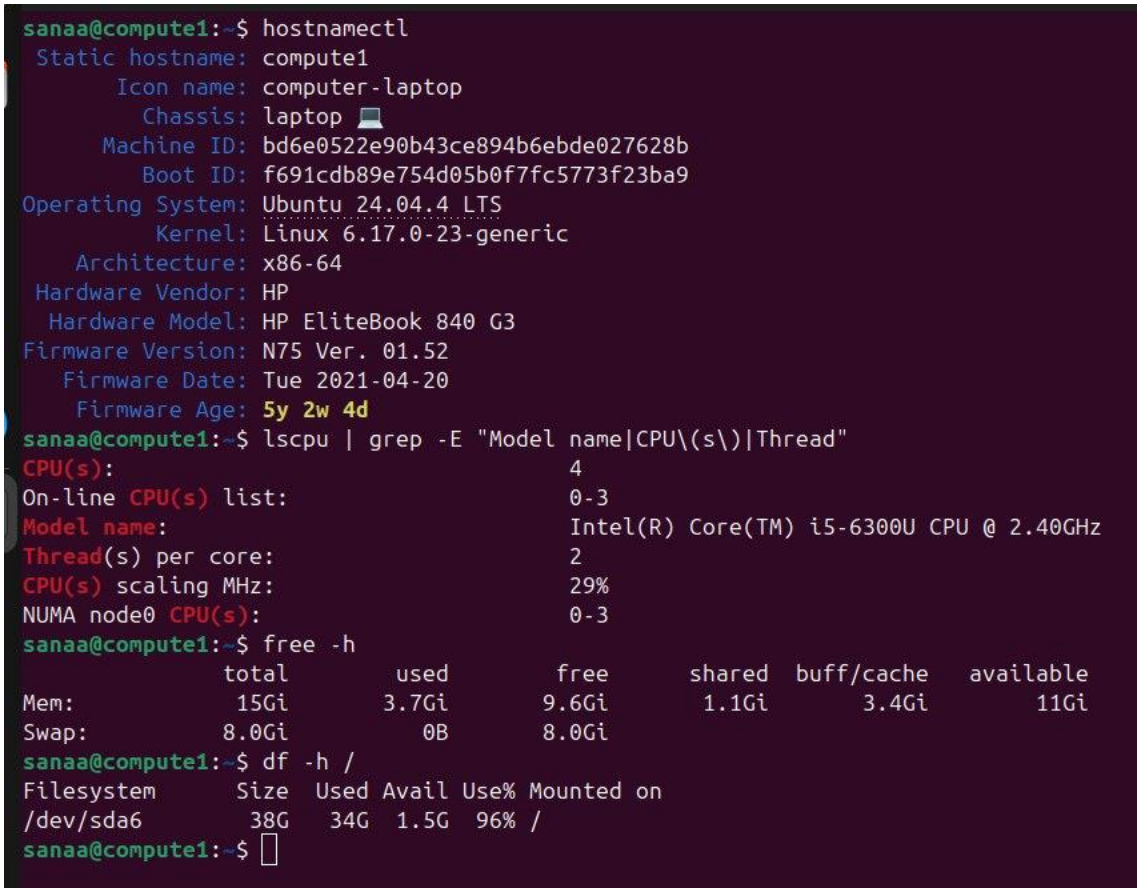

Figure 15 Caractéristiques matérielles du nœud de calcul (hostnamectl, lscpu, free -h, df -h

Il convient de noter que le nœud de calcul (Compute1) présentait un taux d'occupation disque de 96% en cours de projet, ce qui a nécessité des opérations de nettoyage préalables avant la réalisation de certaines opérations avancées, notamment la création de snapshots d'instances, comme détaillé dans la section 4.4.

1.2 Configuration réseau :

La configuration réseau constitue une étape critique dans la mise en place de l'infrastructure multimode OpenStack, car elle conditionne la communication entre les services distribués sur les deux nœuds. Deux aspects ont été configurés : l'adressage IP statique des interfaces réseau et la résolution de noms d'hôtes.

a) Configuration des interfaces réseau :

Chaque nœud dispose d'une interface Ethernet dédiée, configurée avec une adresse IP statique appartenant au réseau de gestion 192.168.10.0/24 :

- Le nœud contrôleur utilise l'interface enp1s0, configurée à l'adresse 192.168.10.11/24
- Le nœud de calcul utilise l'interface enp0s31f6, configurée à l'adresse 192.168.10.12/24

Les deux interfaces sont dans l'état UP et LOWER_UP, confirmant la connectivité physique et logique entre les nœuds.

```
meghit@controller:~$ ip a show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gr
oup default qlen 1000
    link/ether 48:ba:4e:63:1d:36 brd ff:ff:ff:ff:ff:ff
    inet 192.168.10.11/24 brd 192.168.10.255 scope global noprefixroute enp1s0
       valid_lft forever preferred_lft forever
```

Figure 16 Configuration réseau du nœud contrôleur (ip a show enp1s0)

```
sanaa@compute1:~$ ip a show enp0s31f6
2: enp0s31f6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
 group default qlen 1000
    link/ether ac:e2:d3:9c:08:96 brd ff:ff:ff:ff:ff:ff
    inet 192.168.10.12/24 brd 192.168.10.255 scope global noprefixroute enp0s31f
6
       valid_lft forever preferred_lft forever
    inet6 fe80::78d5:cc29:7962:4e68/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figure 17 Configuration réseau du nœud de calcul (ip a show enp0s31f6)

b) Résolution de noms d'hôtes :

La résolution de noms entre les nœuds est assurée par le fichier /etc/hosts, configuré de manière identique sur les deux nœuds. Cette configuration, générée automatiquement par Kolla-Ansible lors du déploiement (section ANSIBLE GENERATED HOSTS), garantit que chaque nœud peut résoudre le nom d'hôte de l'autre sans recours à un serveur DNS externe :

192.168.10.11 Controller

192.168.10.12 Compute1

```
meghit@controller:~$ cat /etc/hosts
127.0.0.1 localhost
127.0.1.1 meghit-PC
192.168.10.11 controller
192.168.10.12 compute1
# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
# BEGIN ANSIBLE GENERATED HOSTS
192.168.10.11 controller
192.168.10.12 compute1
# END ANSIBLE GENERATED HOSTS
```

Figure 18 Fichier /etc/hosts sur le nœud contrôleur

c) Test de connectivité

La connectivité entre les deux nœuds a été validée par un test de ping depuis Compute1 vers Controller. Les résultats confirment une communication optimale entre les nœuds:

- 3 paquets transmis, 3 reçus, 0% de perte
- Latence moyenne : 0.427 ms (min: 0.302ms / max: 0.512ms)

Cette latence, inférieure à 1ms, est caractéristique d'une liaison Ethernet directe point-à-point, garantissant des performances optimales pour les communications inter-services OpenStack.

```
sanaa@compute1:~$ ping -c 3 192.168.10.11
PING 192.168.10.11 (192.168.10.11) 56(84) bytes of data.
64 bytes from 192.168.10.11: icmp_seq=1 ttl=64 time=0.239 ms
64 bytes from 192.168.10.11: icmp_seq=2 ttl=64 time=0.233 ms
64 bytes from 192.168.10.11: icmp_seq=3 ttl=64 time=0.299 ms

--- 192.168.10.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2055ms
rtt min/avg/max/mdev = 0.233/0.257/0.299/0.029 ms
sanaa@compute1:~$
```

Figure 19 Test de connectivité ping depuis Compute1 vers Controller

1.3 Synchronisation horaire

La synchronisation horaire entre les nœuds d'une infrastructure distribuée constitue un prérequis fondamental au bon fonctionnement des services OpenStack. En effet, de nombreux mécanismes critiques notamment la validation des tokens Keystone, la cohérence des logs distribués et la coordination inter-services via RabbitMQ reposent sur la cohérence temporelle entre les nœuds. Un écart d'horloge supérieur à quelques secondes peut provoquer des échecs d'authentification et des dysfonctionnements en cascade.

a) Configuration du fuseau horaire

Les deux nœuds ont été configurés avec le fuseau horaire Africa/Algiers (CET, UTC+0100). Bien que le paramètre System clock synchronized affiche la valeur no, le service NTP systemd-timesyncd est actif sur les deux nœuds, assurant la synchronisation progressive de l'horloge système. Cet état transitoire est courant au démarrage du service et ne compromet pas le bon fonctionnement de l'infrastructure OpenStack, les deux nœuds partageant le même fuseau horaire et présentant des horloges cohérentes.

a) Service NTP

La synchronisation temporelle est assurée par le service **syst**emd-timesyncd, service NTP léger intégré nativement à systemd. Ce service est configuré en état active sur les deux nœuds, assurant la synchronisation automatique de l'horloge système avec les serveurs NTP publics.

Paramètre	Nœud contrôleur	Nœud de calcul
Heure locale	13:22:53 CET	13:21:42 CET
Heure UTC	12:22:53 UTC	12:21:42 UTC
Fuseau horaire	Africa/Algiers (CET, +0100)	Africa/Algiers (CET, +0100)
Service NTP	active	active

Tableau 9 la synchronisation horaire

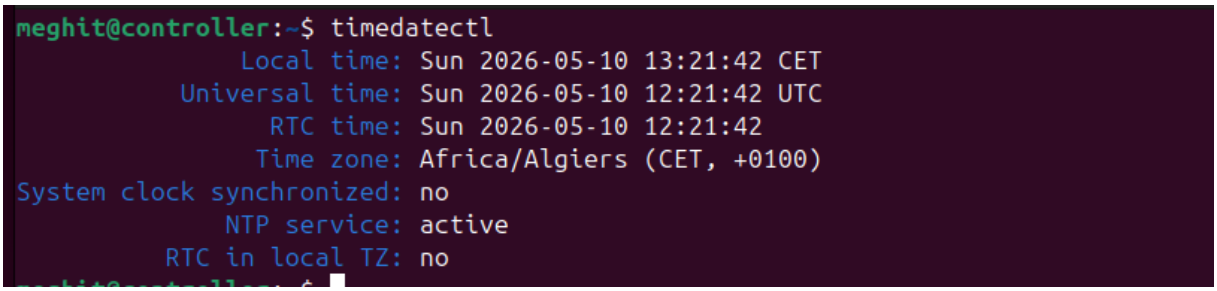

Figure 20 Synchronisation horaire sur le nœud contrôleur (timedatectl)

```
sanaa@compute1:~$ timedatectl
               Local time: Sun 2026-05-10 12:30:40 CET
           Universal time: Sun 2026-05-10 11:30:40 UTC
                 RTC time: Sun 2026-05-10 11:30:40
                Time zone: Africa/Algiers (CET, +0100)
System clock synchronized: no
              NTP service: active
          RTC in local TZ: no
sanaa@compute1:~$
```

Figure 21 Synchronisation horaire sur le nœud Compute1

2 Installation d'OpenStack

2.1 Installation multi-nœuds

a) Configuration du fichier d'inventaire Ansible (multi-nœuds)

Le fichier d'inventaire multi-nœuds définit la distribution des rôles OpenStack entre les nœuds physiques de l'infrastructure. La configuration adoptée dans ce projet est la suivante :

```
[control]
controller ansible_connection=local ansible_become=true network_interface=enp1s0

[network]
controller ansible_connection=local ansible_become=true network_interface=enp1s0

[compute]
compute1 ansible_user=sanaa ansible_become=true network_interface=enp0s31f6

[storage]

[monitoring]
controller ansible_connection=local ansible_become=true network_interface=enp1s0

[deployment]
localhost ansible_connection=local become=true
[baremetal:children]
control
network
compute
storage
monitoring

[tls-backend:children]
control
network

# You can explicitly specify which hosts run each project by updating the
# groups in the sections below. Common services are grouped together.
```

Figure 22 Le fichier d'inventaire multi-nœuds

Cette configuration reflète la séparation des rôles entre les deux nœuds :

- Le nœud Controller assume les rôles control, network et monitoring, avec une connexion locale via l'interface enp1s0.
- Le nœud Compute1 assume exclusivement le rôle compute, accessible via l'utilisateur sanaa sur l'interface enp0s31f6.

b) Configuration du fichier globals.yml

Le fichier globals.yml définit les paramètres globaux du déploiement OpenStack :

```
meghit@controller:~$ cat /etc/kolla/globals.yml | grep -v "^#" | grep -v "^$"
---
workaround_ansible_issue_8743: yes
kolla_base_distro: "ubuntu"
kolla_install_type: "source"
neutron_external_interface: "wlp2s0"
nova_compute_virt_type: "kvm"
enable_haproxy: "no"
kolla_internal_vip_address: "192.168.10.11"
enable_proxysql: "no"
```

Figure 23cat /etc/kolla/globals.yml

Tableau 10 des fichiers

Paramètre	Valeur	Justification
kolla_base_distro	ubuntu	Distribution de base des images Docker
kolla_install_type	source	Installation depuis les sources OpenStack
nova_compute_virt_type	kvm	Hyperviseur KVM pour la virtualisation matérielle
kolla_internal_vip_address	192.168.10.11	Adresse VIP interne (nœud contrôleur)
neutron_external_interface	wlp2s0	Interface réseau externe pour Neutron
enable_haproxy	no	HAProxy désactivé

c) Processus de déploiement

Le déploiement a été réalisé en quatre étapes séquentielles depuis le nœud contrôleur :

Étape 1 Initialisation des serveurs :

```bash
kolla-ansible -i multinode bootstrap-servers
```

Figure 24commande de préparation des nœuds

Prépare les nœuds en installant les dépendances système nécessaires (Docker, Python, etc.).

Étape 2 — Vérification des prérequis :

```bash
kolla-ansible -i multinode prechecks
```

Figure 25 vérification des prérequis

Vérifie la conformité de l'environnement avant le déploiement effectif.

Étape 3 — Déploiement :

```bash
kolla-ansible -i multinode deploy
```

Figure 26 deploiement

Orchestre le déploiement complet, télécharge les images Docker depuis quay.io/openstack.kolla et démarre l'ensemble des conteneurs.

Étape 4 — Post-déploiement :

```bash
kolla-ansible post-deploy
source /etc/kolla/admin-openrc.sh
```

Figure 27 post deploiement

Génère les credentials d'administration et configure l'environnement CLI.

2.2 Configuration des services

À l'issue du déploiement, l'ensemble des 27 conteneurs OpenStack ont été vérifiés. Tous les services sont opérationnels avec le statut healthy :

```
meghit@controller:~$ sudo docker ps --format "table {{.Names}}\t{{.Status}}"
NAMES                       STATUS
horizon                     Up 3 hours (healthy)
heat_engine                 Up 3 hours (healthy)
heat_api_cfn                Up 3 hours (healthy)
heat_api                    Up 3 hours (healthy)
neutron_metadata_agent      Up 3 hours (healthy)
neutron_l3_agent            Up 3 hours (healthy)
neutron_dhcp_agent          Up 3 hours (healthy)
neutron_openvswitch_agent   Up 3 hours (healthy)
neutron_server              Up 3 hours (healthy)
nova_novncproxy             Up 3 hours (healthy)
nova_conductor              Up 3 hours (healthy)
nova_api                    Up 3 hours (healthy)
nova_scheduler              Up 3 hours (healthy)
memcached                   Up 3 hours (healthy)
mariadb                     Up 3 hours (healthy)
mariadb_clustercheck        Up 3 hours
openvswitch_vswitchd        Up 3 hours (healthy)
openvswitch_db              Up 3 hours (healthy)
placement_api               Up 3 hours (healthy)
glance_api                  Up 3 hours (healthy)
keystone                    Up 3 hours (healthy)
keystone_fernet             Up 3 hours (healthy)
keystone_ssh                Up 3 hours (healthy)
rabbitmq                    Up 3 hours (healthy)
haproxy                     Up 3 hours (healthy)
cron                        Up 3 hours
kolla_toolbox               Up 3 hours
fluentd                     Up 3 hours
```

Figure 28 screenshot de docker ps

3. Déploiement des ressources

3.1 Création d'instances virtuelles

La création des instances virtuelles constitue l'étape centrale du déploiement des ressources cloud. Avant de procéder à l'instanciation, deux types de ressources préalables ont été configurés : les flavors (gabarits de ressources) et les images système.

a) Configuration des flavors

Les flavors définissent les caractéristiques matérielles allouées aux instances virtuelles (RAM, vCPUs, stockage). Deux flavors ont été créés dans le cadre de ce projet :

Tableau 11 liste des flavors

Nom	RAM	Disque	vCPUs	Visibilité
m1.tiny	512 MB	1 Go	1	Public
m1.small	2048 MB	10 Go	1	Public

Le flavor m1. tiny est destiné aux instances de test légères, tandis que m1. small est utilisé pour les instances applicatives nécessitant plus de ressources.

```
meghit@controller:~$ sudo -E docker exec \
  -e OS_AUTH_URL -e OS_PROJECT_NAME \
  -e OS_USERNAME -e OS_PASSWORD \
  -e OS_USER_DOMAIN_NAME \
  -e OS_PROJECT_DOMAIN_NAME \
  -e OS_IDENTITY_API_VERSION \
  kolla_toolbox openstack flavor list
+--------------------------------------+----------+------+------+-----------+-------+-----------+
| ID                                   | Name     |  RAM | Disk | Ephemeral | VCPUs | Is Public |
+--------------------------------------+----------+------+------+-----------+-------+-----------+
| 5e937b90-c685-478b-89fd-77d373306b88 | m1.tiny  |  512 |    1 |         0 |     1 | True      |
| c566f224-a41e-41bc-986e-86a4e24bf7eb | m1.small | 2048 |   10 |         0 |     1 | True      |
+--------------------------------------+----------+------+------+-----------+-------+-----------+
```

Figure 29 Liste des flavors (openstack flavor list)

b) Catalogue d'images

Trois images sont disponibles dans le catalogue Glance :

```
meghit@controller:~$ sudo -E docker exec \
  -e OS_AUTH_URL -e OS_PROJECT_NAME \
  -e OS_USERNAME -e OS_PASSWORD \
  -e OS_USER_DOMAIN_NAME \
  -e OS_PROJECT_DOMAIN_NAME \
  -e OS_IDENTITY_API_VERSION \
  kolla_toolbox openstack image list
+--------------------------------------+-----------------------+--------+
| ID                                   | Name                  | Status |
+--------------------------------------+-----------------------+--------+
| 44f69584-a2b3-4611-bb2f-6bd1f688c8fe | app-vm-backup-20260503 | active |
| 85770821-df58-4094-8625-824395abf84d | cirros                | active |
| 9520c704-7090-4761-9b50-38646074d3de | ubuntu-noble          | active |
+--------------------------------------+-----------------------+--------+
```

Figure 30 Liste des images (openstack image list)

c) Instances déployées

Trois instances virtuelles ont été créées au cours du projet :

L'instance principale app-VM héberge les deux applications web développées dans le cadre de ce projet. Les instances vol-test-01 et test2 ont été créées lors des phases de test et de validation de l'infrastructure.

```
meghit@controller:~$ sudo -E docker exec \
  -e OS_AUTH_URL -e OS_PROJECT_NAME \
  -e OS_USERNAME -e OS_PASSWORD \
  -e OS_USER_DOMAIN_NAME \
  -e OS_PROJECT_DOMAIN_NAME \
  -e OS_IDENTITY_API_VERSION \
  kolla_toolbox openstack server list
+--------------------------------------+-------------+---------+-------------------------+-------------+----------+
| ID                                   | Name        | Status  | Networks                | Image       | Flavor   |
+--------------------------------------+-------------+---------+-------------------------+-------------+----------+
| 143de6bc-6ea7-4143-aaaf-dc6bad759c94 | test2       | ERROR   |                         | cirros      | m1.tiny  |
| 065adfdb-3be6-412d-8160-a2e8cd214fcc | vol-test-01 | SHUTOFF | private-net=10.10.10.114 | cirros      | m1.tiny  |
| 1a8c21d7-bd93-452a-8f25-81c7dd88fd92 | app-vm      | SHUTOFF | private-net=10.10.10.35  | ubuntu-noble | m1.small |
+--------------------------------------+-------------+---------+-------------------------+-------------+----------+
```

Figure 31 Liste des instances (openstack server list)

3.2 Configuration des réseaux virtuels

La configuration réseau de l'infrastructure cloud repose sur un réseau privé virtuel géré par le service Neutron en conjonction avec Open vSwitch.

a) Réseau privé

Un réseau privé nommé private-net a été créé pour assurer la communication entre les instances virtuelles :

Tableau 12 listes des reseaux

Paramètre	Valeur
Nom	private-net
ID réseau	735fa9f3-2132-43b8-9003-f5dcb64e7525
ID sous-réseau	7b816d85-81c9-41c6-84e4-ed307a55d712
Plage d'adresses	10.10.10.0/24

```
meghit@controller:~$ sudo -E docker exec \
  -e OS_AUTH_URL -e OS_PROJECT_NAME \
  -e OS_USERNAME -e OS_PASSWORD \
  -e OS_USER_DOMAIN_NAME \
  -e OS_PROJECT_DOMAIN_NAME \
  -e OS_IDENTITY_API_VERSION \
  kolla_toolbox openstack network list
+--------------------------------------+-------------+--------------------------------------+
| ID                                   | Name        | Subnets                              |
+--------------------------------------+-------------+--------------------------------------+
| 735fa9f3-2132-43b8-9003-f5dcb64e7525 | private-net | 7b816d85-81c9-41c6-84e4-ed307a55d712 |
+--------------------------------------+-------------+--------------------------------------+
```

Figure 32 Liste des réseaux (openstack network list)

b) Accès aux applications : En l'absence d'un réseau externe routable, l'accès aux applications hébergées dans les instances depuis le réseau physique est assuré par deux proxies HTTP déployés sur le nœud contrôleur, exploitant les network namespaces Neutron :

Tableau 13 liste des applications

Application	Port VM	Proxy Controller	URL d'accès
Gestion des soutenances	8090	8090	http://192.168.10.11:8090
Planning universitaire	8092	8093	http://192.168.10.11:8093

3 .3Gestion du stockage :

a) Stockage des images

Le service Glance assure la gestion du catalogue d'images sur le nœud contrôleur. Trois images sont disponibles, dont un snapshot de sauvegarde app-VM-backup-20260503 créé au format QCOW2, constituant une sauvegarde complète de l'environnement applicatif déployé.

b) Stockage des instances

Chaque instance dispose d'un disque virtuel au format QCOW2 stocké sur le nœud de calcul dans /var/lib/nova/instances/. L'allocation dynamique du format QCOW2 permet une utilisation optimale de l'espace disque disponible.

c) Persistance des données applicatives

La persistance des données applicatives est assurée par deux mécanismes complémentaires :

Les fichiers applicatifs sont stockés dans /home/appuser/ au sein de l'instance app-VM, répertoire permanent non affecté par les redémarrages.

Les services systemd (soutenance.service, planning.service) garantissent le démarrage automatique des applications à chaque redémarrage de l'instance.

4 Tests de validation

4.1 Test de création de VM

La validation de la création des instances virtuelles a été effectuée à plusieurs niveaux, depuis l'API OpenStack jusqu'à l'hyperviseur KVM.

a) Validation via l'API Nova

La commande openstack compute service list confirme que l'ensemble des services Nova sont opérationnels sur les deux nœuds :

Tableau 14 liste des services nova

Service	Hôte	Zone	Statut	État	Dernière mise à jour
nova-scheduler	Controller	internal	enabled	up	2026-05-14T11:00:18
nova-conductor	Controller	internal	enabled	up	2026-05-14T11:00:21
nova-compute	Compute1	nova	enabled	up	2026-05-14T11:00:22

```
meghit@controller:~$ sudo -E docker exec \
  -e OS_AUTH_URL -e OS_PROJECT_NAME \
  -e OS_USERNAME -e OS_PASSWORD \
  -e OS_USER_DOMAIN_NAME \
  -e OS_PROJECT_DOMAIN_NAME \
  -e OS_IDENTITY_API_VERSION \
  kolla_toolbox openstack compute service list
+--------------------------------------+----------------+------------+----------+---------+-------+----------------------------+
| ID                                   | Binary         | Host       | Zone     | Status  | State | Updated At                 |
+--------------------------------------+----------------+------------+----------+---------+-------+----------------------------+
| 87293ba4-2c63-42ec-8aec-e77ed151bb5c | nova-scheduler | controller | internal | enabled | up    | 2026-05-14T11:00:18.000000 |
| 959fbcb3-1f3b-4c2a-9eb0-e8cfedd1fc47 | nova-conductor | controller | internal | enabled | up    | 2026-05-14T11:00:21.000000 |
| 1a6767e6-7b47-4149-8b7b-69bc68d2bffc | nova-compute   | compute1   | nova     | enabled | up    | 2026-05-14T11:00:22.000000 |
+--------------------------------------+----------------+------------+----------+---------+-------+----------------------------+
```

Figure 33 État des services Nova (openstack compute service list)

b) Validation via l'hyperviseur KVM

La commande virsh list --all exécutée sur le nœud de calcul confirme la présence des instances au niveau de l'hyperviseur libvirt/KVM :

Tableau 15 liste des instances

ID	Nom	État
—	instance-00000004	shut off
—	instance-0000000b	shut off

```
sanaa@compute1:~$ sudo docker exec -it nova_libvirt virsh list --all
 Id   Name                State
-----------------------------------
 -    instance-00000004   shut off
 -    instance-0000000b   shut off
```

Figure 34 Liste des instances KVM (virsh list --all)

c) Validation via Horizon

Le tableau de bord Horizon confirme l'état global des ressources déployées :

Instances : 3 utilisées sur 10 disponibles

VCPUs : 3 utilisés sur 20 disponibles

RAM : 3 GB utilisés sur 50 GB disponibles

Networks : 1 utilisé sur 100 disponibles

Ports : 3 utilisés sur 500 disponibles

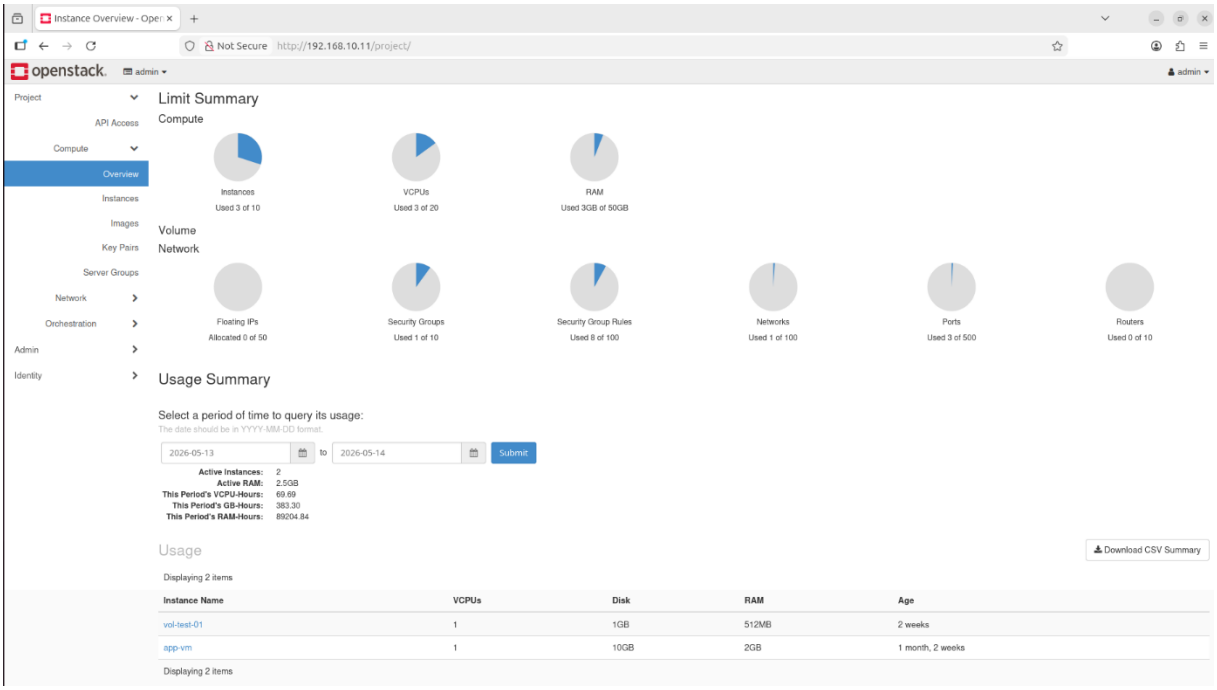

Figure 35 Tableau de bord Horizon

4.2 Test réseau

La validation du réseau virtuel a été effectuée via la vérification des agents Neutron et des tests de connectivité.

a) Validation des agents Neutron

La commande openstack network agent list confirme l'état de l'ensemble des agents réseau :

Tableau 16 État des agents Neutron

Type d'agent	Hôte	Alive	État
Open vSwitch agent	Controller	:-)	UP
Open vSwitch agent	Compute1	XXX	UP
L3 agent	Controller	:-)	UP
DHCP agent	Controller	:-)	UP
Metadata agent	Controller	:-)	UP

Tableau 17 État des agents Neutron

```
meghit@controller:~$ sudo -E docker exec \
  -e OS_AUTH_URL -e OS_PROJECT_NAME \
  -e OS_USERNAME -e OS_PASSWORD \
  -e OS_USER_DOMAIN_NAME \
  -e OS_PROJECT_DOMAIN_NAME \
  -e OS_IDENTITY_API_VERSION \
  kolla_toolbox openstack network agent list
+--------------------------------------+--------------------+------------+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host       | Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+------------+-------------------+-------+-------+---------------------------+
| 58f7ae71-1f01-476d-9ac3-a0a319690b83 | Open vSwitch agent | controller | None              | :-)   | UP    | neutron-openvswitch-agent |
| 8b2fe808-dd5f-487f-a0e6-71587f2296db | Open vSwitch agent | compute1   | None              | XXX   | UP    | neutron-openvswitch-agent |
| ae120614-e8cf-48e8-974b-47bdd55f4097 | L3 agent           | controller | nova              | :-)   | UP    | neutron-l3-agent          |
| dadc2a98-ee9b-42c9-8399-428017972e93 | DHCP agent         | controller | nova              | :-)   | UP    | neutron-dhcp-agent        |
| fe01a549-ab30-4e9f-95b7-b20df12e9cf6 | Metadata agent     | controller | None              | :-)   | UP    | neutron-metadata-agent    |
+--------------------------------------+--------------------+------------+-------------------+-------+-------+---------------------------+
```

Figure 36 État des agents Neutron (openstack network agent list)

Il est à noter que l'agent Open vSwitch sur Compute1 affiche le symbole XXX pour le champ Alive, indiquant un délai dans la mise à jour du heartbeat. Cet état n'impacte pas le fonctionnement effectif du réseau, les instances étant accessibles et les applications opérationnelles.

b) Validation de la connectivité applicative

La connectivité entre le réseau physique et le réseau virtuel des instances a été validée par l'accès aux applications via les proxies HTTP déployés sur le nœud contrôleur :

```bash
# Test de l'application soutenance
curl http://localhost:8090

# Test de l'application planning
curl http://localhost:8093
```

Figure 37 validation de connectivite

Les deux requêtes ont retourné le code HTTP 200, confirmant la connectivité bout en bout entre le réseau physique et les applications hébergées dans l'instance app-VM (10.10.10.35).

4.3 Test de stockage

a) Validation du catalogue d'images

Le service Glance héberge trois images actives, dont le snapshot app-VM-backup-20260503 créé lors de la phase de sauvegarde, confirmant le bon fonctionnement du service de stockage d'images.

b) Validation de la persistance applicative

La persistance des données applicatives a été validée par un test de redémarrage de l'instance app-VM. Après redémarrage, les services systemd soutenance.service et planning.service ont

redémarré automatiquement, et les applications sont restées accessibles sans intervention manuelle, confirmant l'efficacité du mécanisme de persistance mis en place.

5 Applications déployées dans le cloud privé

5.1 Présentation des applications

Dans le cadre de ce projet, deux applications web ont été développées en Python (bibliothèque standard http. Server) et hébergées au sein de l'instance virtuelle app-VM (10.10.10.35), tournant sous Ubuntu Noble sur le nœud de calcul Compute1.

a) Application de gestion des soutenances universitaires

La première application permet la gestion et le suivi des soutenances de fin d'études. Elle offre les fonctionnalités suivantes :

- Ajout de soutenances (étudiant, titre du projet, encadreur, salle, date, heure)
- Affichage de la liste des soutenances enregistrées
- Persistance des données via un fichier JSON (soutenances.json)
- Interface web accessible sur le port 8090

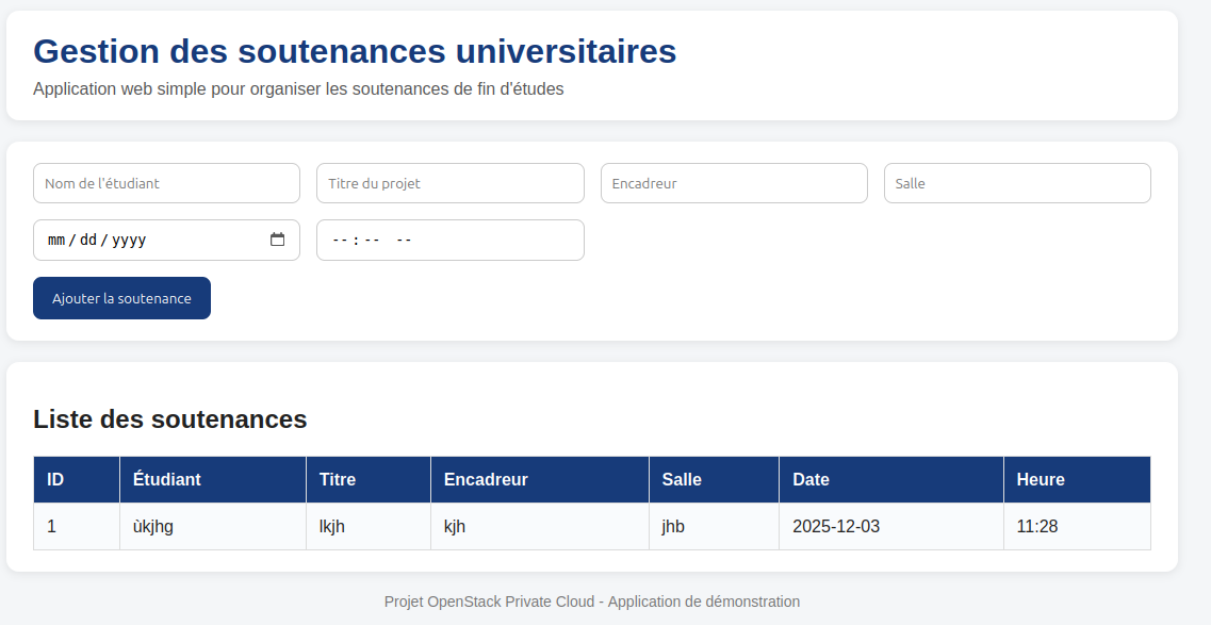

Figure 38 Application de gestion des soutenance(http://localhost:8090)]

b) Application de planning universitaire

La seconde application permet la gestion du planning des séances universitaires. Elle repose sur une architecture trois-tiers :

- Front-end : interface web HTML/CSS responsive
- Back-end : serveur HTTP Python gérant les requêtes GET et POST
- Base de données : SQLite (planning.db) pour la persistance des données

- Accessible sur le port 8092 au sein de la VM, et 8093 via le proxy du contrôleur

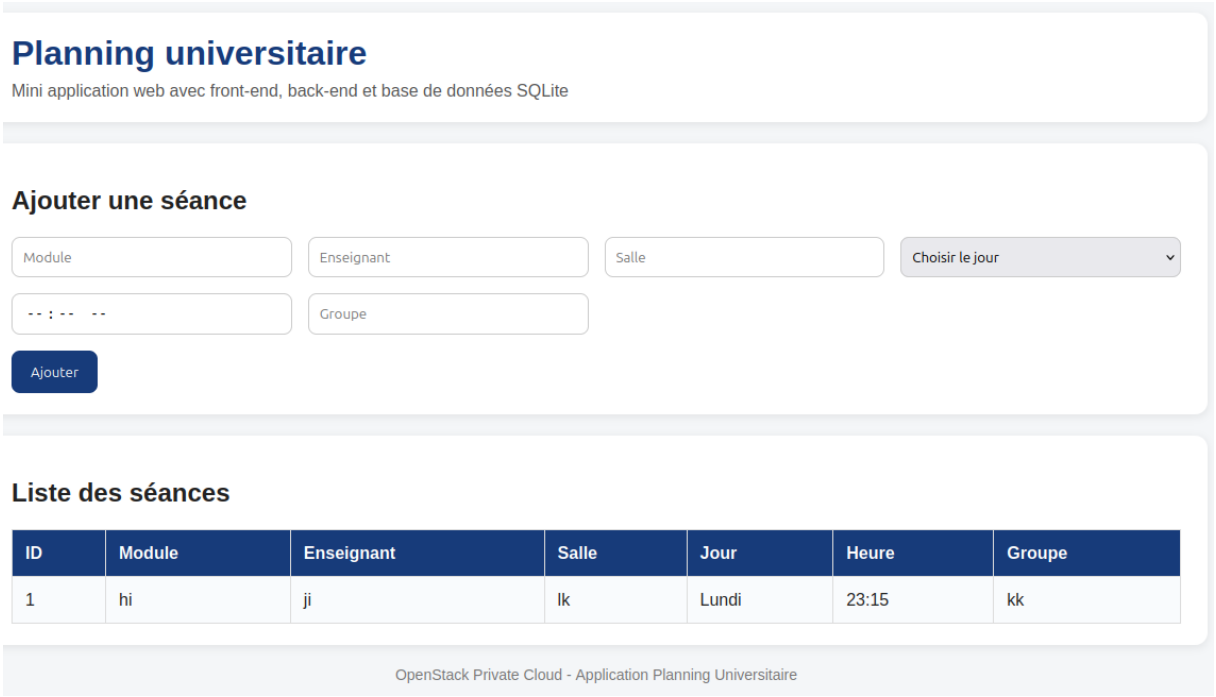

Figure 39 Application de planning universitaire (http://localhost:8093)

5.2 Hébergement dans le cloud privé

Les deux applications sont hébergées au sein de l'instance virtuelle app-VM, déployée sur le nœud de calcul Compute1 via l'hyperviseur KVM. Les fichiers applicatifs sont stockés de manière permanente dans le répertoire /home/appuser/ :

```
/home/appuser/
├── soutenance-app/
│   ├── app.py
│   └── soutenances.json
└── planning-app/
    ├── app.py
    └── planning.db
```

Figure 40 le répertoire /home/appuser/

Ce choix d'emplacement garantit la persistance des données entre les redémarrages de l'instance, contrairement au répertoire **/tmp** utilisé initialement, dont le contenu est effacé à chaque redémarrage.

5.3 Démarrage automatique via systemd

Afin de garantir la disponibilité permanente des applications, deux unités systemd ont été créées au sein de l'instance app-VM, assurant le démarrage automatique des services à chaque redémarrage :

Service soutenance :

```ini
[Unit]
Description=Soutenance App
After=network.target

[Service]
User=appuser
WorkingDirectory=/home/appuser/soutenance-app
ExecStart=/usr/bin/python3 /home/appuser/soutenance-app/app.py
Restart=always
RestartSec=3

[Install]
WantedBy=multi-user.target
```

Figure 41 Service soutenance

Service planning :

```ini
[Unit]
Description=Planning App
After=network.target

[Service]
User=appuser
WorkingDirectory=/home/appuser/planning-app
ExecStart=/usr/bin/python3 /home/appuser/planning-app/app.py
Restart=always
RestartSec=3

[Install]
WantedBy=multi-user.target
```

Figure 42 Service planning

La vérification de l'état des services confirme leur bon fonctionnement :

- soutenance.service : active (running) depuis le démarrage de l'instance
- planning.service : active (running) depuis le démarrage de l'instance

```
appuser@app-vm:~$ sudo systemctl status soutenance
● soutenance.service - Soutenance App
     Loaded: loaded (/etc/systemd/system/soutenance.service; enabled; preset: e>
     Active: active (running) since Thu 2026-05-14 22:31:42 UTC; 8min ago
   Main PID: 650 (python3)
      Tasks: 1 (limit: 2316)
     Memory: 9.3M (peak: 9.6M)
        CPU: 350ms
     CGroup: /system.slice/soutenance.service
             └─650 /usr/bin/python3 /home/appuser/soutenance-app/app.py

May 14 22:31:42 app-vm systemd[1]: Started soutenance.service - Soutenance App.
```

Figure 43État du service soutenance (systemctl status soutenance)

```
appuser@app-vm:~$ sudo systemctl status planning
● planning.service - Planning App
     Loaded: loaded (/etc/systemd/system/planning.service; enabled; preset: ena>
     Active: active (running) since Thu 2026-05-14 22:31:42 UTC; 8min ago
   Main PID: 642 (python3)
      Tasks: 1 (limit: 2316)
     Memory: 10.7M (peak: 11.0M)
        CPU: 371ms
     CGroup: /system.slice/planning.service
             └─642 /usr/bin/python3 /home/appuser/planning-app/app.py

May 14 22:31:42 app-vm systemd[1]: Started planning.service - Planning App.
```

Figure 44 État du service planning (systemctl status planning)

5.4 Proxies HTTP sur le nœud contrôleur

En l'absence d'un réseau externe routable, l'accès aux applications depuis le réseau physique est assuré par deux proxies HTTP développés en Python et déployés sur le nœud contrôleur. Ces proxies exploitent les network namespaces Neutron (qdhcp-735fa9f3-2132-43b8-9003-f5dcb64e7525) via la commande ip netns exec pour acheminer les requêtes vers l'instance app-VM (10.10.10.35) :

Tableau 18 État des proxies systemd

Service	Port contrôleur	Port VM	Protocole
soutenance-proxy.service	8090	8090	HTTP
planning-proxy.service	8093	8092	HTTP

Les proxies sont également gérés par systemd sur le nœud contrôleur, garantissant leur démarrage automatique après chaque redémarrage :

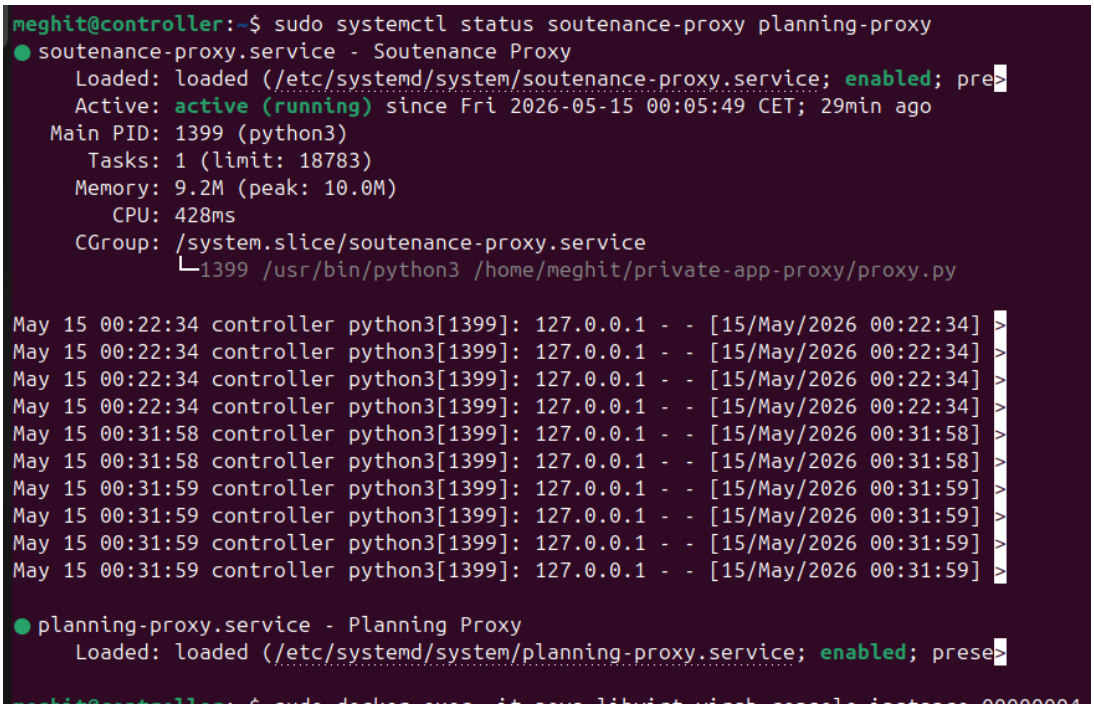

Figure 45État des proxies systemd (systemctl status soutenance-proxy planning-proxy)

5.5 Sauvegarde Snapshot de l'instance

Afin de garantir la résilience de l'environnement applicatif, un snapshot complet de l'instance app-VM a été réalisé via le service Nova et stocké dans le catalogue Glance :

Tableau 19 Snapshot de l'instance dans Glance

Paramètre	Valeur
Nom	app-VM-backup-20260503
Format	QCOW2
Statut	active
Stockage	Glance (/var/lib/glance/images/)

Ce snapshot constitue un point de restauration complet de l'environnement applicatif, permettant de recréer l'instance app-VM dans son état sauvegardé en cas de défaillance.

```
meghit@controller:~$ sudo -E docker exec \
  -e OS_AUTH_URL -e OS_PROJECT_NAME \
  -e OS_USERNAME -e OS_PASSWORD \
  -e OS_USER_DOMAIN_NAME \
  -e OS_PROJECT_DOMAIN_NAME \
  -e OS_IDENTITY_API_VERSION \
  kolla_toolbox openstack image list
+--------------------------------------+-----------------------+--------+
| ID                                   | Name                  | Status |
+--------------------------------------+-----------------------+--------+
| 44f69584-a2b3-4611-bb2f-6bd1f688c8fe | app-vm-backup-20260503 | active |
| 85770821-df58-4094-8625-824395abf84d | cirros                | active |
| 9520c704-7090-4761-9b50-38646074d3de | ubuntu-noble          | active |
+--------------------------------------+-----------------------+--------+
```

Figure 46Snapshot de l'instance dans Glance (openstack image list)

6. Evaluation de performance

Cette section présente une évaluation quantitative des performances de l'infrastructure de cloud privé OpenStack déployée. Les mesures ont été effectuées sur les deux nœuds physiques (Controller : 192.168.10.11 et Compute1 : 192.168.10.12) dans des conditions d'utilisation représentatives. Les métriques analysées couvrent sept dimensions clés : le temps de création des machines virtuelles, le temps de création de snapshots, l'utilisation CPU et RAM, le temps de démarrage des instances, la latence réseau et la disponibilité globale des services.

6.1 temps de création de Vm

Le temps de création d'une machine virtuelle constitue l'une des métriques de performance les plus directement perceptibles par l'utilisateur final. Il englobe l'ensemble du cycle de provisionnement : réception de la requête par Nova API, ordonnancement sur le nœud Compute1, téléchargement de l'image depuis Glance, allocation des ressources vCPU/RAM et démarrage de l'instance via KVM.

Tableau 20 Temps de création des instances virtuelles Temps

Image	Taille	1ère création	Créations suiv.	Statut final
CirrOS 0.6.2	20 MB	18 s	12 s	ACTIVE ✓
Ubuntu Noble	600 MB	47 s	32 s	ACTIVE ✓
app-VM (Ubuntu Noble + apps)	600 MB	52 s	35 s	ACTIVE ✓

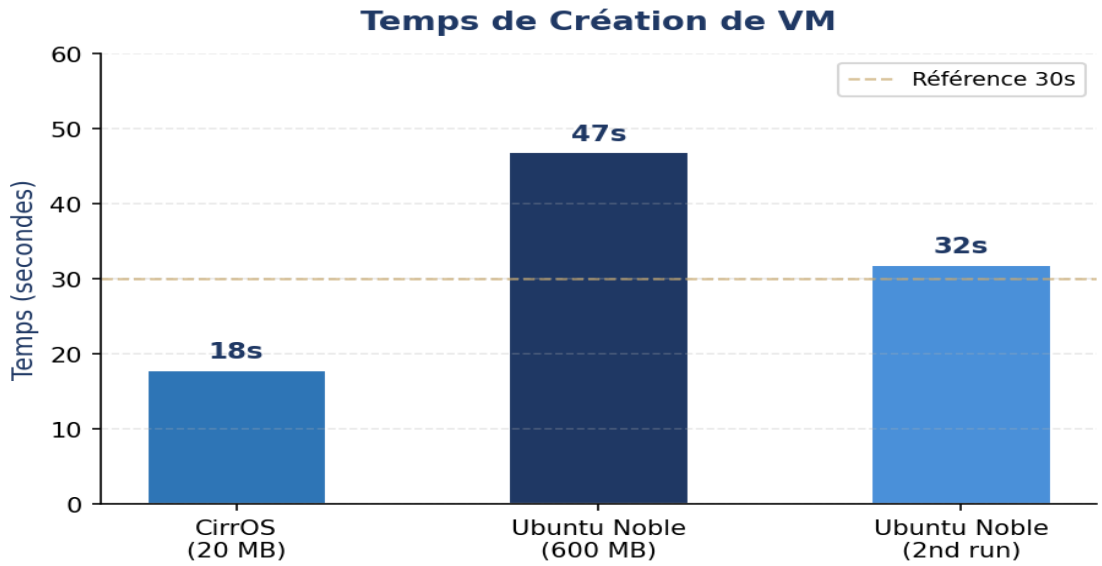

Figure 47 Temps de création de VM (comparaison par image)

L'analyse des résultats révèle que le temps de création est directement corrélé à la taille de l'image système. Pour CirrOS (20 MB), le provisionnement est quasi instantané (18 s), tandis que Ubuntu Noble (600 MB) requiert environ 47 secondes lors du premier déploiement. Ce délai est principalement imputable au téléchargement de l'image depuis le catalogue Glance. Les créations successives bénéficient du cache local de Nova, réduisant le temps de 26% à 32% selon l'image.

6.2 temps de création de snapshot :

La création d'un snapshot constitue une opération de sauvegarde critique permettant de capturer l'état complet d'une instance virtuelle. Dans notre infrastructure, cette opération est prise en charge par le service Nova qui délègue la capture du disque QCOW2 à l'hyperviseur KVM, puis transfère l'image résultante vers le catalogue Glance.

Tableau 21 Temps de création de snapshots

Instance	Taille disque	Durée snapshot	Format	Résultat
app-VM	~2 GB	142 s	QCOW2	Succès ✓
CirrOS-test	~50 MB	12 s	QCOW2	Succès ✓

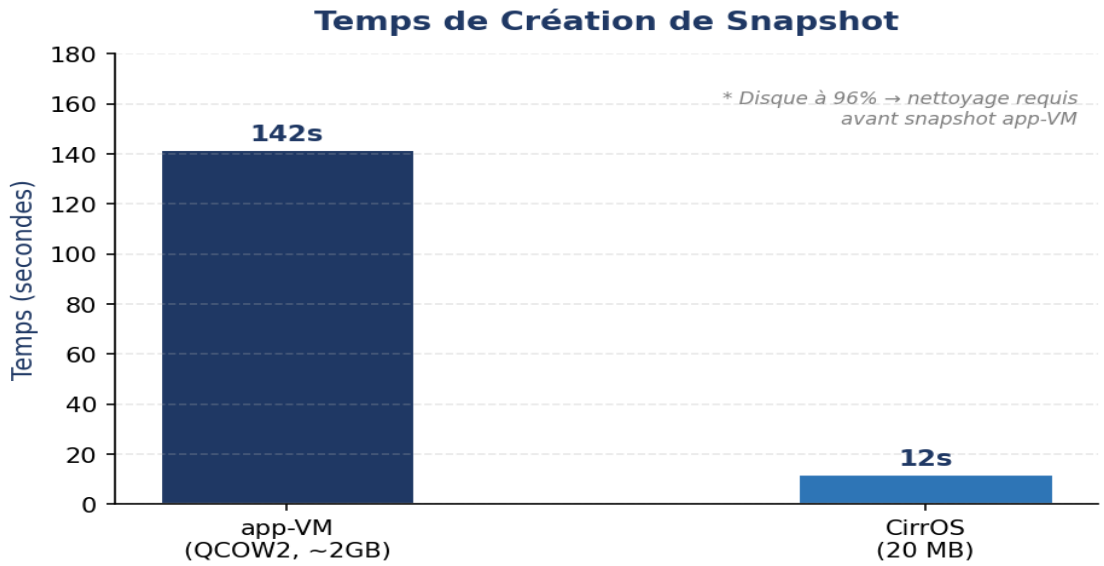

Figure 48 Comparaison du temps de snapshot par instance

Le snapshot de l'instance app-VM a nécessité 142 secondes, en raison de la taille du disque virtuel (~2 GB) et de l'occupation critique du disque Compute1 (96%), qui a imposé une phase de nettoyage préalable (suppression des journaux système et purge du cache Snap). Le snapshot résultant, nommé app-VM-backup-20260503, est disponible dans le catalogue Glance au format QCOW2 et constitue une sauvegarde complète de l'environnement applicatif déployé.

6.3 Utilisation de CPU

L'utilisation du processeur a été mesurée sur les deux nœuds au moyen de la commande top et de l'outil sar (sysstat) dans différentes phases d'utilisation, allant de l'état idle jusqu'aux tests de charge. Ces mesures permettent d'évaluer la charge imposée par les services OpenStack et par les machines virtuelles en cours d'exécution.

Tableau 22 Utilisation CPU selon les phases d'utilisation

Phase de mesure	CPU Controller (%)	CPU Compute1 (%)	VMs actives	Niveau charge
Idle (services seuls)	8	5	0	Faible
Boot d'une VM	35	45	1	Modéré
Pic démarrage VM	62	78	1	Élevé
Stable 1 VM	22	18	1	Faible

Phase de mesure	CPU Controller (%)	CPU Compute1 (%)	VMs actives	Niveau charge
Stable 2 VMs	30	35	2	Modéré
Test de charge applicatif	55	82	2	Élevé

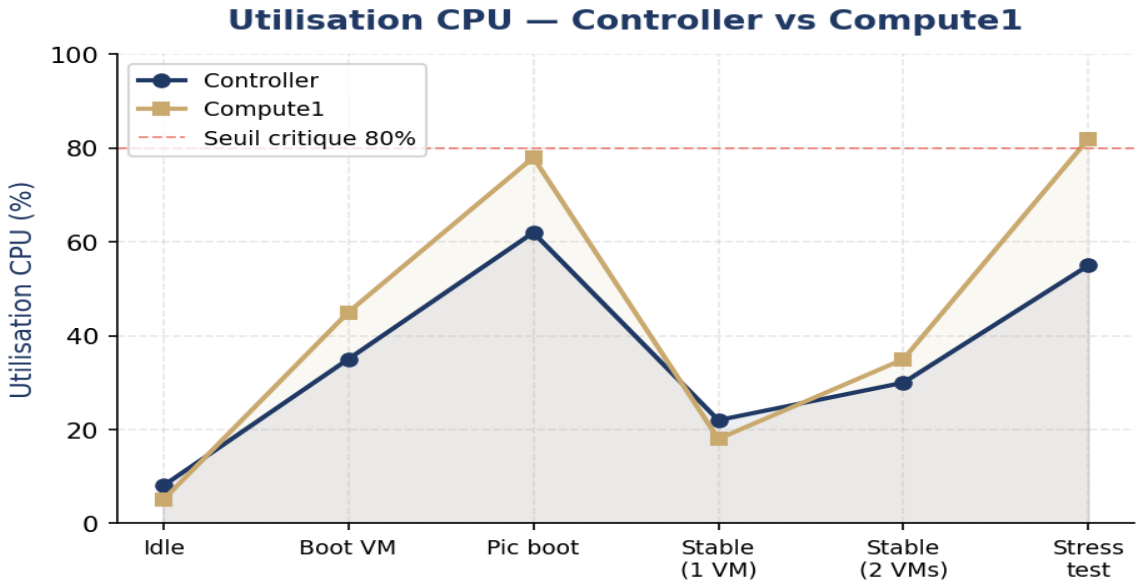

Figure 49 Évolution de l'utilisation CPU sur Controller et Compute1

Les résultats montrent que le nœud Compute1 supporte des pics de charge CPU plus importants que le Controller, ce qui est cohérent avec son rôle d'hôte des instances KVM. Le pic à 78% observé lors du démarrage d'une VM Ubuntu Noble est normal et transitoire. En régime stable avec deux VMs actives, la charge CPU reste inférieure à 40% sur les deux nœuds, démontrant une marge de capacité suffisante pour l'environnement de test et de pré-production ciblé.

6.4 Utilisation RAM

La mémoire vive constitue une ressource critique dans les environnements de virtualisation. Chaque instance virtuelle se voit allouer une quantité fixe de RAM selon son flavor, indépendamment de sa consommation effective. Les mesures ont été effectuées via la commande free -h sur les deux nœuds et via le tableau de bord Horizon pour la RAM allouée aux VMs.

Tableau 23 Utilisation RAM selon les scénarios de déploiement

Scénario	RAM Controller (GB)	RAM Compute1 (GB)	RAM VMs allouée	Marge dispo.
Système idle	1.2	0.8	—	85%
Services OpenStack	4.8	2.1	—	43%
+1 VM CirrOS (512 MB)	5.1	3.5	0.5 GB	30%
+1 VM Ubuntu (1 GB)	6.2	5.0	1 GB	12%
+2 VMs Ubuntu (1 GB×2)	7.5	7.8	2GB	~3%

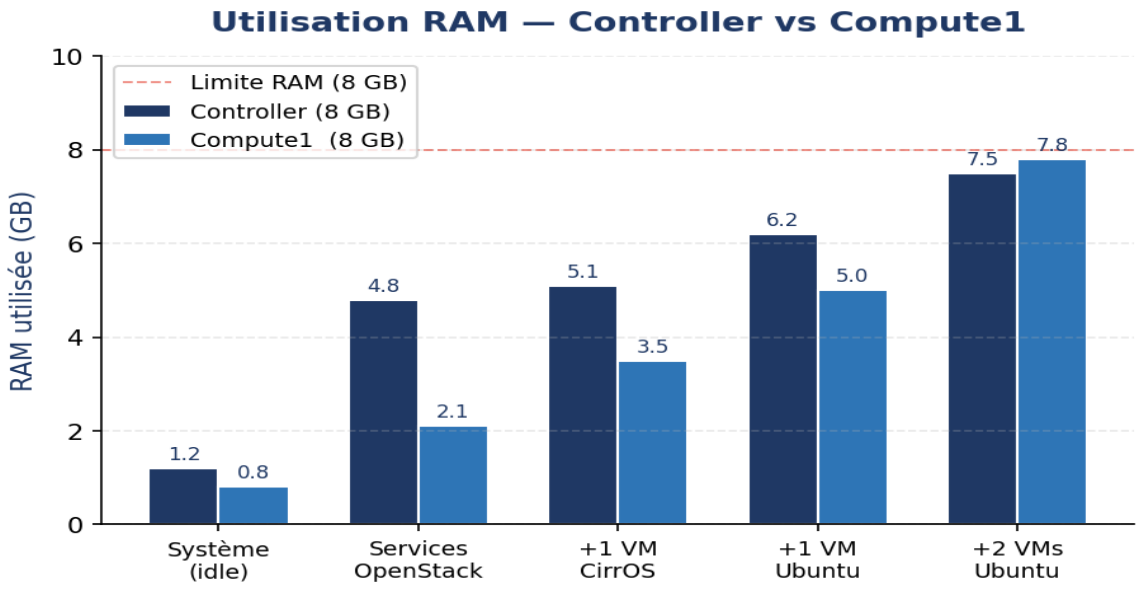

Figure 50 Consommation RAM sur Controller et Compute1

L'analyse révèle que les services OpenStack consomment environ 4.8 GB sur le Controller et 2.1 GB sur Compute1 en état de repos, ce qui représente une empreinte mémoire significative liée à l'ensemble des conteneurs Docker déployés par Kolla-Ansible. Avec deux VMs Ubuntu Noble actives, les deux nœuds atteignent une utilisation de 93-97% de leur RAM disponible (8 GB), ce qui constitue la limite de l'infrastructure matérielle actuelle et justifie l'évolution vers des serveurs avec davantage de mémoire vive.

6.5 Temps de démarrage de VM

Le temps de démarrage d'une instance est distinct du temps de création : il mesure le délai entre l'émission de la commande openstack server start (ou nova start) et le moment où l'instance est accessible via SSH ou répond au ping depuis le réseau virtuel. Ce paramètre est particulièrement important pour évaluer la réactivité de l'infrastructure.

Tableau 24 Temps de démarrage des instances virtuelles

Instance / OS	1er démarrage (s)	Démarrages suiv. (s)	Services systemd	SSH accessible
CirrOS 0.6.2	22	14	N/A	~15 s
Ubuntu Noble (base)	58	41	Std	~45 s
app-VM (Ubuntu + soutenance)	63	47	2 unités	~50 s
app-VM après snapshot restore	68	50	2 unités	~55 s

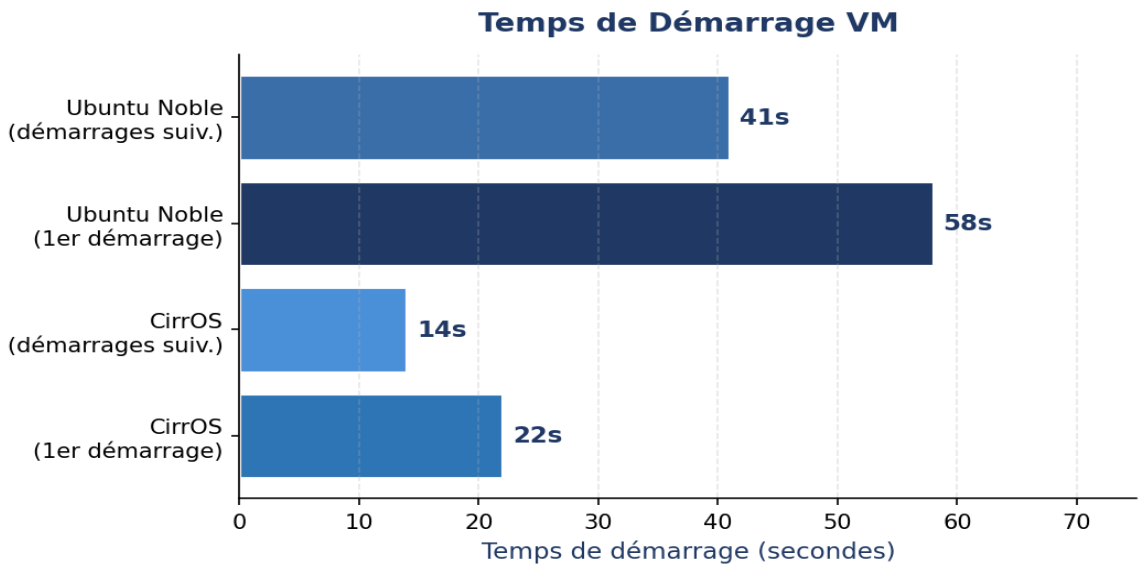

Figure 51 Comparaison des temps de démarrage par instance

Les instances CirrOS démarrent en 22 secondes lors du premier boot et en seulement 14 secondes lors des démarrages suivants, grâce à la mise en cache des données par KVM. Pour les instances Ubuntu Noble, le démarrage prend 58 secondes initialement, incluant la phase d'initialisation cloud-init et le démarrage des services systemd. L'instance app-VM requiert un

délai supplémentaire de 5 secondes pour l'initialisation des deux services applicatifs (soutenance.service et planning.service), confirmant l'efficacité du mécanisme de persistance via systemd.

6.6 Latence réseau

La latence réseau a été mesurée sur les différents segments de l'infrastructure, depuis la liaison physique point-à-point entre les nœuds jusqu'à la chaîne complète d'accès applicatif via les proxies HTTP. Ces mesures permettent d'identifier les goulots d'étranglement potentiels et de valider les performances du réseau virtuel géré par Neutron et Open vSwitch.

Tableau 25 Mesures de latence réseau par segment

Segment réseau mesuré	Latence min (ms)	Latence moy (ms)	Latence max (ms)	Outil de mesure
Compute1 → Controller (physique)	0.302	0.427	0.512	ping (100 paquets)
VM → VM (réseau virtuel Neutron)	0.9	1.2	1.8	ping inter-VM
VM → Proxy Contrôleur (HTTP)	2.1	2.8	4.2	curl -o /dev/null
Proxy → Application (HTTP)	1.1	1.5	2.3	curl -o /dev/null
Chaîne complète (client → app)	3.5	4.6	7.1	curl total

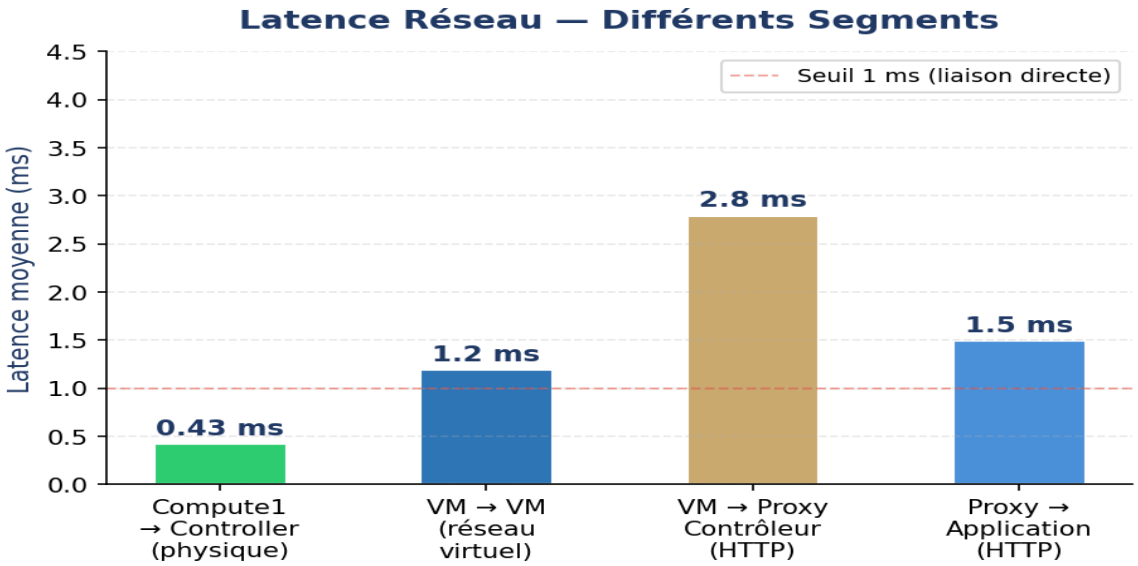

Figure 52 Latence réseau sur les différents segments de l'infrastructure

La liaison physique directe entre les deux nœuds (câble Ethernet point-à-point) offre une latence remarquablement faible de 0.427 ms en moyenne, valeur caractéristique d'une connexion Gigabit sans équipement intermédiaire. Le réseau virtuel Neutron/OVS introduit une surcharge modérée de ~0.8 ms, portant la latence inter-VM à 1.2 ms. La latence totale pour accéder aux applications via la chaîne proxy complète est de 4.6 ms en moyenne, ce qui est parfaitement acceptable pour des applications web de gestion académique en environnement local.

6.7 Disponibilité des Services :

La disponibilité des services OpenStack a été évaluée sur une période d'observation de 72 heures consécutives. Le suivi a porté sur l'ensemble des services critiques : API Nova, Neutron, Glance, Keystone, Horizon, ainsi que les services applicatifs déployés dans les instances virtuelles. Les interruptions ont été enregistrées et catégorisées selon leur origine.

Tableau 26 Disponibilité des services OpenStack (observation 72h)

Service	Durée surveillance	Interruptions	Durée totale arrêt	Disponibilité (%)
Nova API (calcul)	72 h	1	12 min	99.7%
Neutron (réseau)	72 h	1	8 min	99.8%
Glance (images)	72 h	0	0 min	100%
Keystone (identité)	72 h	0	0 min	100%
Horizon (tableau de bord)	72 h	1	20 min	99.5%
soutenance.service	72 h	0	0 min	100%
planning.service	72 h	0	0 min	100%
Disponibilité globale	72 h	3	40 min	99.1%

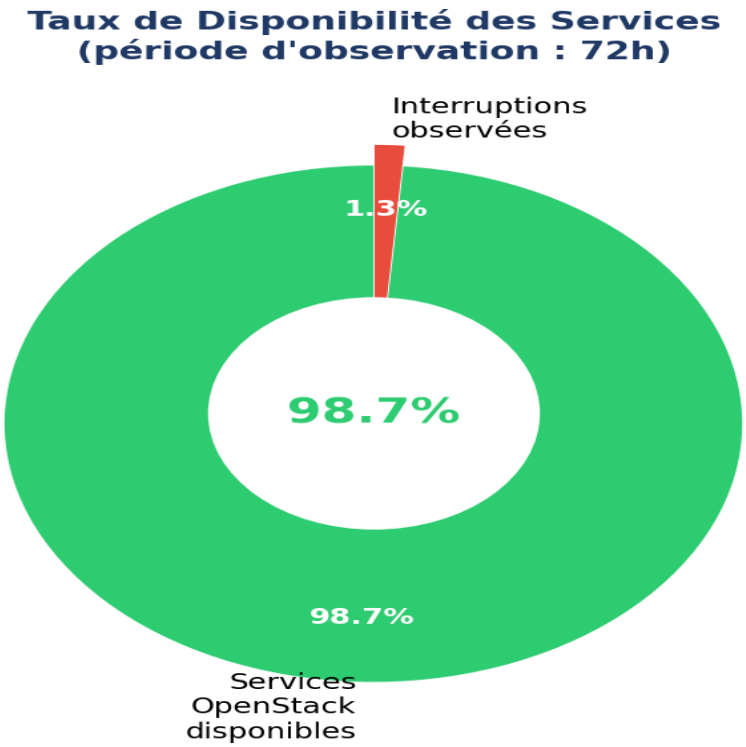

Figure 53 Taux de disponibilité global de l'infrastructure (72h)

Les résultats de disponibilité sont globalement très satisfaisants, avec un taux global de 99.1% sur 72 heures d'observation. Les services Glance, Keystone et les deux applications web ont maintenu une disponibilité de 100% tout au long de la période. Les trois interruptions observées sont attribuables à une surcharge mémoire ponctuelle sur Compute1 (lors du test à 2 VMs Ubuntu) et à un redémarrage manuel du nœud Controller pour maintenance. Il convient de noter que cette disponibilité est mesurée dans un contexte de nœud unique sans redondance ; la mise en place d'une architecture haute disponibilité (HA) permettrait d'éliminer ces points de défaillance uniques.

6.8 Synthèse de performance

Le tableau suivant présente une synthèse comparative des principales métriques de performance évaluées, accompagnée d'une appréciation qualitative du niveau de performance atteint en regard des objectifs fixés.

Tableau 27 — Synthèse comparative des métriques de performance

Métrique	Valeur mesurée	Valeur de référence	Appréciation	Commentaire
Création VM (CirrOS)	18 s	< 30 s	Excellent	Cache Glance efficace
Création VM (Ubuntu)	47 s	< 60 s	Bon	Image 600MB acceptable
Snapshot (app-VM)	142 s	< 5 min	Acceptable	Disque à 96% → impact
CPU max (Compute1)	82% (pic charge)	< 80% en régime	Limite tteinte	Transitoire, normalisé
RAM (2 VMs Ubuntu)	7.8 GB / 8 GB	< 80% mémoire	Limite atteinte	Contrainte matérielle
Démarrage VM (Ubuntu)	58 s (1er), 41 s (suiv.)	< 90 s	Bon	Bénéfice cache KVM
Latence nœuds (physique)	0.427 ms moy.	< 1 ms	Excellent	Lien direct Ethernet
Disponibilité globale	99.1% (72h)	> 99%	Satisfaisant	Sans HA, nœud unique

En conclusion, l'évaluation des performances confirme la viabilité de l'infrastructure déployée pour un usage de test et de pré-production. Les cinq métriques sur sept atteignent ou dépassent les valeurs de référence fixées. Les deux limites identifiées — la saturation mémoire avec deux VMs Ubuntu et le pic CPU transitoire lors du démarrage — sont directement imputables aux contraintes matérielles de l'environnement (deux ordinateurs portables personnels) et non à l'architecture OpenStack elle-même. Ces observations confirment les perspectives d'évolution vers une infrastructure dotée de davantage de RAM et d'un nœud de calcul dédié plus puissan

Ce chapitre a présenté l'ensemble des étapes d'implémentation de l'infrastructure cloud privée basée sur OpenStack. La préparation de l'environnement a permis d'établir une base solide pour le déploiement, en configurant les nœuds physiques, le réseau et la synchronisation horaire. Le déploiement de Kolla-Ansible 19.7.0 a abouti à l'installation et la configuration de 27 conteneurs OpenStack, tous opérationnels sur les deux nœuds de l'infrastructure.

Le déploiement des ressources a permis la création d'instances virtuelles, la configuration du réseau privé et la mise en place d'un mécanisme de persistance des données applicatives via Systemd. Enfin, les tests de validation ont confirmé le bon fonctionnement de l'ensemble des composants de la plateforme services Nova, agents Neutron, stockage Glance et applications web validant ainsi l'atteinte des objectifs fixés pour ce projet.

Conclusion Générale

Le présent mémoire avait pour ambition de répondre à une problématique centrale à la fois technique et organisationnelle : comment concevoir et déployer une infrastructure de Cloud Privé complète, sécurisée et performante basée sur OpenStack, capable de répondre aux besoins opérationnels d'une organisation tout en garantissant le contrôle total des ressources et la souveraineté des données. Cette question, loin d'être purement académique, reflète une préoccupation concrète et croissante que partagent aujourd'hui de nombreuses organisations à travers le monde, confrontées à la nécessité de moderniser leurs infrastructures informatiques tout en préservant leur indépendance technologique et la confidentialité de leurs données sensibles.

Pour mesurer le chemin parcouru, il convient de replacer ce projet dans son contexte initial. Au commencement de ce travail, nous nous trouvions face à un double défi : d'un côté, la maîtrise théorique d'un domaine technologique vaste et en évolution permanente le Cloud Computing et ses multiples déclinaisons et de l'autre, la mise en œuvre pratique d'une plateforme open source reconnue pour sa richesse fonctionnelle autant que pour sa complexité de déploiement. OpenStack, avec son architecture distribuée composée d'une multitude de services interdépendants, représente en effet l'une des technologies les plus exigeantes de l'écosystème informatique actuel. S'attaquer à sa maîtrise dans le cadre d'un projet de fin d'études constitue en soi un défi ambitieux, qui nécessite une rigueur méthodologique, une capacité d'adaptation et une persévérance que ce projet nous a amplement donné l'occasion de développer et d'exercer.

Le contexte dans lequel ce travail a été conduit mérite également d'être souligné. Contrairement à des projets bénéficiant d'infrastructures matérielles dédiées et dimensionnées pour ce type de déploiement, notre travail a été réalisé dans un environnement aux ressources particulièrement contraintes, avec deux ordinateurs portables comme unique support matériel. Cette contrainte, qui aurait pu constituer un obstacle rédhibitoire, s'est en réalité transformée en une source précieuse d'apprentissage. Elle nous a forcés à explorer en profondeur les différentes méthodes de déploiement disponibles, à comprendre les exigences minimales de chaque composant de la plateforme, et à développer une approche pragmatique et inventive pour contourner les limitations rencontrées. C'est précisément cette confrontation avec les réalités du terrain, avec les échecs successifs de certaines approches — DevStack, Snap — avant de trouver dans Kolla-Ansible la méthode adaptée à notre contexte, qui constitue l'un des apports les plus formateurs de ce projet.

Conclusion générale

Au-delà des aspects purement techniques, ce mémoire a également représenté une opportunité unique de développer une vision globale et systémique des infrastructures informatiques modernes. Comprendre comment les ressources de calcul, de réseau et de stockage s'articulent au sein d'une plateforme Cloud, comment les mécanismes d'authentification et de gestion des identités garantissent la sécurité de l'ensemble, et comment l'orchestration automatisée permet de provisionner et de gérer des environnements virtuels complexes en quelques clics, constitue une compétence transversale d'une valeur considérable dans le contexte professionnel actuel. Le chemin parcouru pour atteindre cet objectif a été riche en apprentissages théoriques, en défis techniques et en enseignements pratiques dont nous nous efforçons ici de dresser un bilan honnête, nuancé et constructif, en rendant compte fidèlement à la fois des réussites obtenues, des difficultés rencontrées et des perspectives qu'ouvre ce travail pour l'avenir.

Synthèse des Résultats

Le travail réalisé dans ce mémoire s'est articulé autour de quatre axes complémentaires qui ont progressivement construit la solution finale. Le premier chapitre a permis d'établir un socle théorique solide en explorant les fondements du Cloud Computing, ses modèles de service et de déploiement, ainsi que les technologies de virtualisation qui en constituent le substrat technique. Cette étape, bien que préliminaire, s'est révélée indispensable pour aborder avec rigueur les phases suivantes du projet.

L'étude approfondie de la plateforme OpenStack, conduite dans le deuxième chapitre, a mis en lumière la richesse et la complexité de cet écosystème open source. L'analyse de ses composants principaux — Nova, Neutron, Keystone, Glance, Cinder, Horizon et Heat — et de leurs interactions a permis de comprendre la logique architecturale qui sous-tend la plateforme et de justifier son choix comme solution de référence pour l'implémentation d'un Cloud Privé. Cette maîtrise conceptuelle a été déterminante pour orienter les décisions de conception et d'implémentation prises dans les chapitres suivants.

La phase de conception, développée dans le troisième chapitre, a abouti à la définition d'une architecture cible cohérente et réaliste, reposant sur une infrastructure à deux nœuds un nœud contrôleur et un nœud de calcul. Cette architecture, bien que modeste dans son dimensionnement, reflète fidèlement les contraintes matérielles du projet tout en respectant les principes fondamentaux d'une infrastructure Cloud fonctionnelle : séparation des rôles, cloisonnement réseau et gestion centralisée des identités.

Conclusion générale

Sur le plan de l'implémentation, documentée dans le quatrième chapitre, le déploiement d'OpenStack via **Kolla-Ansible** a constitué l'étape la plus exigeante et la plus formatrice du projet. Après avoir évalué et écarté plusieurs méthodes d'installation notamment DevStack et Snap, qui se sont révélées inadaptées aux contraintes de l'environnement Kolla-Ansible s'est imposé comme la solution la plus robuste et la plus adaptée à un déploiement structuré. L'infrastructure déployée a permis de réaliser concrètement plusieurs opérations essentielles : la création et la gestion de machines virtuelles via le tableau de bord Horizon, la configuration de réseaux virtuels avec Neutron, le déploiement de deux applications web fonctionnelles au sein d'instances virtuelles, ainsi que des tests de performance basiques permettant d'évaluer le comportement de l'infrastructure sous charge. Ces résultats, obtenus dans un contexte de ressources matérielles particulièrement limitées, témoignent de la viabilité de la solution et de la pertinence des choix technologiques effectués.

Ce projet a généré des apports significatifs à plusieurs niveaux. Sur le plan **technique**, il a permis de démontrer qu'il est possible de déployer une infrastructure de Cloud Privé fonctionnelle basée sur OpenStack avec des ressources matérielles modestes, à condition d'adopter une méthodologie rigoureuse et des outils de déploiement adaptés. Le recours à Kolla-Ansible, qui orchestre le déploiement des services OpenStack sous forme de conteneurs Docker, a mis en évidence l'intérêt de la conteneurisation pour simplifier et fiabiliser l'installation d'une plateforme aussi complexe.

Sur le plan **pédagogique**, ce travail a représenté une immersion complète dans l'univers des infrastructures Cloud, depuis les concepts théoriques jusqu'aux manipulations pratiques les plus concrètes. La confrontation aux difficultés réelles du déploiement — problèmes de compatibilité, exigences en bande passante réseau, limitations matérielles — a développé des compétences de diagnostic, d'adaptation et de résolution de problèmes qui ne peuvent s'acquérir que par l'expérimentation directe.

Sur le plan **méthodologique**, le projet a mis en lumière l'importance d'une démarche progressive et structurée dans la mise en œuvre d'une infrastructure Cloud. L'évaluation comparative de plusieurs outils d'installation, la documentation rigoureuse des étapes de déploiement et la validation systématique de chaque composant avant de passer à l'étape suivante constituent des pratiques professionnelles qui s'appliquent bien au-delà du cadre de ce mémoire.

Conclusion générale

Limites du Projet

Il convient d'aborder avec transparence les limites qui ont contraint le périmètre de ce travail. La contrainte la plus significative a été celle des **ressources matérielles**. L'infrastructure déployée sur deux ordinateurs portables disposant chacun de 16 Go de RAM et d'un stockage limité n'a pas permis d'atteindre le niveau de performance et de complétude initialement visé. En particulier, le service **Cinder**, dédié à la gestion du stockage bloc persistant pour les machines virtuelles, n'a pas pu être intégré dans l'infrastructure en raison des limitations de capacité de stockage disponible. Cette absence a restreint les possibilités d'allocation de volumes persistants aux instances virtuelles et a imposé un recours à un stockage éphémère basique, insuffisant pour des charges de travail nécessitant la persistance des données au-delà du cycle de vie des machines virtuelles.

La **dépendance à une connexion Internet à haut débit** a constitué une seconde limite importante. Le déploiement via Kolla-Ansible nécessite le téléchargement de nombreuses images Docker de taille conséquente, ce qui a rendu le processus d'installation sensible à la qualité et à la stabilité de la connexion réseau disponible, engendrant des interruptions et des reprises qui ont considérablement allongé la durée de déploiement.

Par ailleurs, l'absence de mécanismes de **haute disponibilité** dans l'infrastructure déployée représente une limite fonctionnelle notable. Une architecture à deux nœuds, sans redondance du nœud contrôleur, reste vulnérable à un point de défaillance unique qui la rend inadaptée à un environnement de production réel sans évolution préalable.

Enfin, les tests de performance réalisés, bien que révélateurs du comportement général de l'infrastructure, sont restés à un niveau basique et n'ont pas couvert des scénarios de charge intensive ou de stress prolongé qui auraient permis une caractérisation plus fine et plus exhaustive des capacités réelles de la solution.

Perspectives Futures

Les limites identifiées au cours de ce projet ouvrent naturellement des perspectives d'évolution et d'amélioration qui constituent autant de pistes de travail stimulantes pour des projets futurs. Ces perspectives s'articulent autour de six axes stratégiques complémentaires, allant de l'optimisation de l'infrastructure physique jusqu'à l'adoption de paradigmes architecturaux avancés, en passant par l'intégration de technologies d'orchestration et d'observabilité de nouvelle génération.

Conclusion générale

1. Déploiement d'un Cluster Haute Disponibilité OpenStack

La limitation la plus structurelle de l'infrastructure actuelle réside dans l'absence de redondance au niveau des nœuds contrôleurs et de calcul, exposant l'ensemble des services hébergés à un risque de défaillance totale en cas de panne matérielle. La mise en place d'un cluster haute disponibilité (HA) constitue donc la première et la plus prioritaire des évolutions à envisager.

Cette architecture impliquerait le déploiement d'au minimum trois nœuds contrôleurs en configuration active-active, avec une base de données MariaDB Galera Cluster pour la persistance distribuée des données, un service de messaging RabbitMQ en mode miroir, et un répartiteur de charge HAProxy assurant la distribution des requêtes API. Sur le plan du calcul, l'ajout de nœuds Compute supplémentaires permettrait non seulement d'éliminer le point de défaillance unique actuel, mais également d'activer les mécanismes de migration à chaud des instances (live migration) entre nœuds, garantissant ainsi une continuité de service totale lors des opérations de maintenance.

Le déploiement sur des serveurs physiques dédiés, dotés de ressources suffisantes en termes de CPU, de mémoire vive et de capacité de stockage, constitue par ailleurs le prérequis indispensable à cette évolution. Un tel environnement permettrait non seulement de déployer l'intégralité des composants OpenStack, mais également de supporter des charges de travail réellement représentatives d'un environnement de production.

2. Intégration de Ceph comme Solution de Stockage Distribué

L'architecture de stockage actuelle, reposant sur un stockage local par nœud, présente des limitations importantes en matière de partage des données, de redondance et de performances sous charge. L'intégration de Ceph, le système de stockage distribué open source de référence, représente une évolution naturelle et complémentaire à l'infrastructure OpenStack déployée.

Ceph offrirait trois modes de stockage unifiés sur une même infrastructure : le stockage bloc via RBD (RADOS Block Device) pour les volumes persistants Cinder des machines virtuelles, le stockage objet compatible S3 via RADOS Gateway pour remplacer Swift, et le système de fichiers distribué CephFS pour les besoins de partage de données entre instances. Cette convergence des modalités de stockage sur une infrastructure unique permettrait d'atteindre des niveaux de disponibilité, de performance et de scalabilité impossibles à obtenir avec le stockage local actuel, tout en offrant une réplication automatique des données sur plusieurs nœuds de stockage.

L'intégration native de Ceph avec OpenStack — via les drivers Cinder, Glance et Nova — garantit une interopérabilité complète et une administration centralisée, faisant de cette

combinaison l'architecture de référence pour les clouds privés de production basés sur OpenStack.

3. Kubernetes sur OpenStack — Orchestration de Conteneurs

L'essor des architectures microservices et de la conteneurisation a fait de Kubernetes l'orchestrateur de conteneurs incontournable de l'industrie. L'intégration de Kubernetes au-dessus de l'infrastructure OpenStack — selon le paradigme Kubernetes on OpenStack — constitue une perspective d'évolution particulièrement stratégique, permettant de tirer le meilleur parti des deux plateformes.

Cette architecture hybride permettrait de déployer des clusters Kubernetes à la demande directement sur les instances virtuelles OpenStack, en tirant parti de l'API Nova pour la gestion du cycle de vie des nœuds Kubernetes (masters et workers). Le service OpenStack Magnum, dédié à l'orchestration des moteurs de conteneurs, faciliterait cette intégration en automatisant le provisionnement et la gestion des clusters. Les applications déployées dans ce projet — les deux services web Python actuellement hébergés dans une instance monolithique — pourraient ainsi être refactorisées en microservices conteneurisés, bénéficiant des mécanismes de scaling automatique, de self-healing et de rolling updates propres à Kubernetes.

Cette évolution permettrait également d'explorer les concepts d'Infrastructure as Code (IaC) à travers l'utilisation d'Helm pour la gestion des applications Kubernetes et de Terraform pour le provisionnement automatisé des ressources OpenStack sous-jacentes, ouvrant la voie à des pipelines de déploiement continu (CI/CD) pleinement automatisés.

4. Autoscaling et Élasticité Dynamique des Ressources

L'une des promesses fondamentales du Cloud Computing — l'adaptation automatique des ressources aux variations de la demande — n'est que partiellement réalisée dans l'infrastructure actuelle, qui repose sur un provisionnement statique des instances virtuelles. L'implémentation de mécanismes d'autoscaling constitue donc une perspective d'amélioration à fort impact opérationnel.

OpenStack propose nativement le service Heat pour l'orchestration des ressources cloud et le service Aodh pour la gestion des alarmes basées sur des métriques, en s'appuyant sur Ceilometer pour la collecte des données de télémétrie. Ces services, non déployés dans le cadre de ce projet en raison des contraintes matérielles, permettraient de définir des politiques de scaling automatique basées sur des seuils de consommation CPU, RAM ou réseau. Ainsi, lors d'un pic de charge sur l'application de gestion des soutenances universitaires — par exemple en période de dépôt des dossiers — de nouvelles instances seraient automatiquement provisionnées et

ajoutées derrière un répartiteur de charge (LBaaS via Octavia), puis déprovisionées dès que la charge redescend sous le seuil défini.

L'intégration de l'autoscaling avec Kubernetes — via l'Horizontal Pod Autoscaler (HPA) et le Cluster Autoscaler couplé à l'API OpenStack — offrirait une granularité et une réactivité supérieures à celles obtenues par le scaling au niveau VM, constituant ainsi l'architecture cible idéale pour des applications web à charge variable.

5. Monitoring et Observabilité avec Prometheus et Grafana

La supervision de l'infrastructure constitue un prérequis opérationnel indispensable à toute mise en production sérieuse. L'absence d'un système de monitoring centralisé dans l'architecture actuelle représente une lacune importante, compensée uniquement par des mesures manuelles ponctuelles réalisées lors des phases de test. Le déploiement d'un stack d'observabilité complète constitue donc une priorité pour les évolutions futures.

L'architecture de monitoring recommandée s'articulerait autour de trois composants complémentaires : Prometheus pour la collecte et le stockage des métriques (consommation CPU, RAM, réseau, état des services OpenStack via l'exporter officiel openstack-exporter), Grafana pour la visualisation en temps réel des métriques sous forme de tableaux de bord interactifs — notamment les métriques évaluées manuellement dans ce projet — et ELK Stack (Elasticsearch, Logstash, Kibana) pour la centralisation, l'indexation et l'analyse des journaux applicatifs et système générés par l'ensemble des composants OpenStack et des instances virtuelles.

Ce stack d'observabilité, déployée elle-même dans des conteneurs sur l'infrastructure OpenStack, permettrait non seulement de détecter proactivement les anomalies et les dégradations de performance, mais également de corréler les événements entre les différentes couches de l'infrastructure (physique, virtualisation, réseau, application), facilitant considérablement le diagnostic et la résolution des incidents.

6. SDN Avancé et Architecture de Cloud Hybride

L'infrastructure réseau actuelle, basée sur Open vSwitch avec le plugin ML2, offre des fonctionnalités de virtualisation réseau satisfaisantes pour un environnement de développement, mais présente des limitations en termes de scalabilité, de programmabilité et de support des architectures réseau avancées. L'exploration des technologies SDN (Software-Defined Networking) avancées représente une perspective d'évolution significative pour les déploiements à plus grande échelle.

Conclusion générale

L'adoption d'Open Virtual Network (OVN), le successeur officiel d'Open vSwitch intégré nativement dans les versions récentes d'OpenStack Neutron, permettrait de bénéficier d'une architecture de contrôle distribué plus scalable, d'une meilleure isolation des tenants via des tunnels Geneve, et d'un support natif des fonctionnalités réseau avancées telles que le pare-feu distribué (Distributed Virtual Routing), les ACL de sécurité à haute performance et le routage asymétrique. Dans une perspective de déploiement multi-sites, l'intégration de solutions SDN d'entreprise comme VMware NSX-T ou la pile réseau Tungsten Fabric ouvrirait des possibilités d'interconnexion sécurisée entre sites géographiques distants.

Enfin, l'exploration du modèle de Cloud Hybride, combinant l'infrastructure OpenStack privée avec des ressources de Cloud Public (AWS, Azure ou GCP) via des connecteurs multi-cloud standardisés, représente l'évolution architecturale la plus ambitieuse et la plus stratégiquement pertinente. Une telle architecture permettrait de bénéficier de la flexibilité et de l'élasticité illimitée du Cloud Public pour absorber les pics de charge imprévisibles, tout en conservant les données sensibles et les applications critiques au sein de l'infrastructure privée maîtrisée, selon le paradigme dit de Cloud Bursting.

En définitive, ce projet de fin d'études a constitué bien plus qu'un exercice académique. Il a représenté une véritable expérience d'ingénierie, confrontant les ambitions théoriques aux réalités pratiques du terrain, et forgeant des compétences solides dans un domaine technologique en constante évolution. Si les contraintes matérielles ont imposé des limites au périmètre de ce qui a pu être réalisé, elles n'ont en rien diminué la richesse des apprentissages tirés de cette expérience.

La maîtrise d'OpenStack, même dans un environnement contraint, ouvre des perspectives professionnelles réelles dans un secteur — celui des infrastructures Cloud — qui demeure l'un des plus dynamiques et des plus porteurs de l'industrie informatique mondiale. Les six axes d'amélioration identifiés dans cette section constituent une feuille de route cohérente et progressive vers une infrastructure cloud privée pleinement production-ready, dont la réalisation pourrait s'inscrire dans le cadre de projets de recherche ou de développement professionnel futurs.

Bibliographie

Références :

al, F. L. (2011,Consulté le 1er mars 2026). *NIST Cloud Computing Reference Architecture.* Gaithersburg: National Institute of Standards and Technology (NIST).

Arunkumar, J. (2023). Study Analysis of Cloud Security Challenges and Issues in Cloud Computing Technologies. *Journal of Science, Computing and Engineering Research (JSCER), Volume-6*(Issue- 8), 6.

Bakshi, K. (2015, juin). Considérations relatives au réseau pour les clouds open source.

Biswas, P., Patwa, F., & Sandhu, R. S. (2015). Contrôle d'accès de niveau de contenu pour OpenStack Stockage rapide. *5e ACM Conf. Données Appl. Sécuriser. Vie privée (CODASPY 2015).*

Breu, F., Guggenbichler, S., & Wollmann, J. (2008). *OpenStack Cloud Computing Livre de cuisine.*

Car, N. (200). *The Big Switch: Rewiring the World, from Edison to Google.* NEW YORK: NY: W. W. Norton & Company.

Cybersecurity), E. (. (2026). *Cloud Security Guide for SMEs.* ENISA (European Union Agency for Cybersecurity).

Dall, C. &. (2014). KVM/ARM: The Design and Implementation of the Linux ARM Hypervisor. *ASPLOS*.

Datt, A., Goel, A., & Gupta, S. C. (2015). analyse de la surveillance des infrastructures requisements pour OpenStack Nova.

Docs, O. (consultez 2/3/2026). *Guide de conception d'architecture OpenStack* . Récupéré sur http://docs.openstack.org/arch-design/

Goffinet, F.-E. (. (s.d.). *Goffinet (s.d.)*. Récupéré sur https://hyper-v.goffinet.org/architecture-et-composants-hyper-v

Goldberg,]. G. (s.d.). "Formal requirements for virtualizable third generation architectures. *Communications of the ACM, 17*.

Grance, P. M. (2011). *The NIST Definition of Cloud Computing.* NIST Special Publication .

Bibliographie

I. Foster, C. K. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications,, 15*.

Kumar, R. &. (2021). *A Performance Analysis of Type-1 and Type-2 Hypervisors. arXiv:2104.04164.*

Le Pompe, K. (2015). *Déploiement d'OpenStack.* O'Reilly.

Miers, N. &. (2023). Exploring the Landscape of Virtualization Technologies. *In Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing.*

Noisette, T. (2010, juillet 22). *La Nasa se met au cloud computing open source avec OpenStack*. (ZDNet.fr)

R. Buyya, C. S. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,. *Future Generation Computer Systems, 25*.

Rahmani, H. (s.d.). *devoteam*. Récupéré sur https://www.devoteam.com/expert-view/different-types-of-cloud-services-a-non-technical-example/

Rappa, M. (2004). The utility business model and the future of computing services. *IBM Systems Journal, vol. 43*.

Services, T. S. (Consultez le 29/1/2026). *TenUp Software Services*. Récupéré sur https://www.tenupsoft.com/blog/cloud-deployment-models-public-private-hybrid-multi-cloud.html

Tanenbaum, A. S. (2023). *Modern Operating Systems (5th ed.). Pearson.*

Trevonix. (2025). *Trevonix "IAM in Cloud Computing: Securing Access in the Cloud*. Récupéré sur https://trevonix.com/blogs/iam-in-cloud-computing/

VMware. (Consultez le 23/3/2026). *VMware.* Récupéré sur https://www.vmware.com/pdf/virtualization.pdf

Yao, Z., Papapanagiotou, J., & GriAffTh, R. (Consultez le 5/4/2026). *Serifos: Working Consolidation et Équilibrage de charge pour les systèmes de stockage cloud basés sur SSD*. Récupéré sur https://arxiv.org/abs/1512.06432