

Openstack

Sommaire

1. Conception d'un Cloud OpenStack
2. Gestion des machines virtuelles
3. Gestion du stockage
4. Gestion des images
5. Gestion du réseau
6. Authentification et autorisations
7. Administration du Cloud

Conception d'un Cloud OpenStack

Conception d'un Cloud OpenStack

Apports et spécificités du Cloud

✓ Apports principaux

1. Élasticité et Scalabilité

- Possibilité d'augmenter ou réduire les ressources à la demande.
- Utile pour absorber des pics de charge sans investissement matériel.

2. Réduction des coûts

- Modèle basé sur la consommation (pay-as-you-go).
- Évite les investissements lourds en matériel et en maintenance.

3. Accessibilité

- Accès aux ressources et aux applications depuis n'importe où via Internet.
- Favorise le télétravail et la collaboration internationale.

Conception d'un Cloud OpenStack

Apports et spécificités du Cloud

✓ Apports principaux

4. Innovation accélérée

- Déploiement rapide d'environnements de test, développement et production.
- Large catalogue de services (IA, Big Data, IoT, sécurité, etc.).

5. Sécurité et résilience

- Centres de données hautement sécurisés avec redondance.
- Sauvegardes et plans de reprise après sinistre (PRA) intégrés.

Conception d'un Cloud OpenStack

Types de Cloud

1. Modèles de services

◆ SaaS (Software as a Service)

- **Définition** : Logiciel accessible via Internet, sans installation locale.
- **Exemples** : Gmail, Microsoft 365, Salesforce.
- **Avantages** :
 - Pas de maintenance par l'utilisateur.
 - Mise à jour automatique.
 - Facturation à l'usage/licence.
- **Limite** : Moins de personnalisation.

Conception d'un Cloud OpenStack

Types de Cloud

1. Modèles de services

◆ PaaS (Platform as a Service)

- **Définition** : Plateforme de développement et déploiement fournie par le Cloud.
- **Exemples** : Heroku, Google App Engine, Azure App Service.
- **Avantages** :
 - Gestion simplifiée du cycle de vie des applications.
 - Pas besoin de gérer serveurs ou OS.
- **Limite** : Dépendance au fournisseur (vendor lock-in).

Conception d'un Cloud OpenStack

Types de Cloud

1. Modèles de services

◆ IaaS (Infrastructure as a Service)

- **Définition** : Mise à disposition de ressources matérielles virtuelles (VM, stockage, réseau).
- **Exemples** : Amazon EC2, Microsoft Azure VM, Google Compute Engine.
- **Avantages** :
 - Contrôle complet sur l'infrastructure.
 - Grande flexibilité.
- **Limite** : Nécessite des compétences d'administration système.

Conception d'un Cloud OpenStack

Types de Cloud

2. Modèles de déploiement

◆ Cloud public

- **Définition** : Services partagés entre plusieurs clients sur l'infrastructure du fournisseur.
- **Exemples** : AWS, Azure, Google Cloud.
- **Avantage** : Coût réduit, scalabilité maximale.
- **Limite** : Moins de contrôle, dépendance au fournisseur.

Conception d'un Cloud OpenStack

Types de Cloud

2. Modèles de déploiement

◆ Cloud privé

- **Définition** : Infrastructure dédiée à une seule organisation.
- **Exemples** : VMware vSphere, OpenStack, Azure Stack.
- **Avantage** : Plus de contrôle et de sécurité.
- **Limite** : Coûts élevés (matériel et maintenance).

Conception d'un Cloud OpenStack

Types de Cloud

2. Modèles de déploiement

◆ Cloud hybride

- **Définition** : Combinaison de Cloud public + privé.
- **Exemple** : Une entreprise garde ses données sensibles en privé, mais utilise un public cloud pour ses applications web.
- **Avantage** : Flexibilité et optimisation des coûts.
- **Limite** : Complexité de gestion.

Conception d'un Cloud OpenStack

Types de Cloud

2. Modèles de déploiement

◆ Cloud hybride

- **Définition** : Combinaison de Cloud public + privé.
- **Exemple** : Une entreprise garde ses données sensibles en privé, mais utilise un public cloud pour ses applications web.
- **Avantage** : Flexibilité et optimisation des coûts.
- **Limite** : Complexité de gestion.

Conception d'un Cloud OpenStack

1. Historique & philosophie du projet

Historique

- **2010** : OpenStack est créé par la collaboration entre **NASA** (projet Nebula) et **Rackspace** (cloud public américain).
- But initial : construire une **infrastructure cloud open source**, alternative aux offres propriétaires (Amazon AWS, VMware...).
- **Projet communautaire** : Rapidement adopté par de nombreux acteurs (RedHat, IBM, HP, SUSE, Intel, Canonical, etc.).
- Évolution :
 - 2011 : Premier « release » officiel (Austin)
 - Sorties régulières tous les 6 mois (cycle de développement proche de Linux)
 - Fondation OpenStack créée en 2012 pour structurer la gouvernance et la roadmap.

Conception d'un Cloud OpenStack

1. Historique & philosophie du projet

Philosophie du projet

- **Open Source** : Tout le code est libre (licence Apache 2.0).
- **Communauté** : Contributions ouvertes, processus démocratique, évènements réguliers (OpenStack Summit, PTG...).
- **Modulaire** :
 - Chaque composant est indépendant (Compute, Réseau, Stockage...).
 - Les services communiquent via API REST et message bus (RabbitMQ).
- **Interopérable** :
 - API standardisées, support multi-hyperviseur (KVM, Xen, VMware...)
 - Support de multiples matériels réseau/stockage
- **Scalable** : Conçu pour de très grands datacenters, mais aussi utilisable en démo sur une seule machine.

Conception d'un Cloud OpenStack

2. Cas d'usage d'OpenStack

Cloud privé (Private Cloud)

- Utilisation principale : fournir à une entreprise ou un organisme son propre « cloud » à la AWS mais hébergé sur son infrastructure.
- Avantages : sécurité, maîtrise des données, personnalisation, conformité.

Cloud public / communautaire

- Certains fournisseurs proposent du cloud public basé sur OpenStack (OVH Public Cloud, CityCloud, etc.)
- Offre mutualisée et élastique pour les clients, alternative aux géants du cloud (AWS, Azure).

Conception d'un Cloud OpenStack

2. Cas d'usage d'OpenStack

Cloud hybride

- Intégration d'OpenStack avec d'autres clouds (Azure, AWS, Google Cloud)
- Cas : migration, burst temporaire de charge, reprise d'activité.

Cas d'usage avancés

- Hébergement d'environnements de test, de production, de CI/CD pour les DevOps
- Plateformes d'hébergement de VM pour la recherche (universités, laboratoires)
- Fourniture de services managés pour des clients internes (as a Service : IaaS, PaaS, CaaS)
- Support de conteneurs (Kubernetes avec Magnum, ou OpenShift sur OpenStack)
- Big Data, HPC (calcul intensif)

Conception d'un Cloud OpenStack

3. Écosystème OpenStack

Distributions OpenStack

- Plusieurs acteurs proposent des distributions « clé en main » d'OpenStack :
 - **Red Hat OpenStack Platform (RHOSP)**
 - **Mirantis OpenStack**
 - **Canonical Charmed OpenStack**
 - **SUSE OpenStack Cloud** (fin de support)
 - **OVH Public Cloud**, CityCloud, etc. (services gérés)
- **Installations simplifiées :**
 - **DevStack** (pour les labs/démo, non production)
 - **MicroStack** (Canonical, facile en local)
 - **Packstack, TripleO, Kolla-Ansible** (pour la prod)

Conception d'un Cloud OpenStack

3. Écosystème OpenStack

Projets dérivés et modules complémentaires

- **Projets « Core »** : Keystone (identité), Nova (compute), Glance (images), Neutron (réseau), Cinder (block storage), Swift (object storage), Horizon (dashboard web)
- **Projets additionnels** :
 - **Heat** (orchestration IaaS « Infrastructure as Code »)
 - **Magnum** (Kubernetes as a Service)
 - **Ironic** (bare metal provisioning)
 - **Octavia** (Load Balancer as a Service)
 - **Barbican** (gestion des secrets)
 - **Manila** (partage de fichiers)

Conception d'un Cloud OpenStack

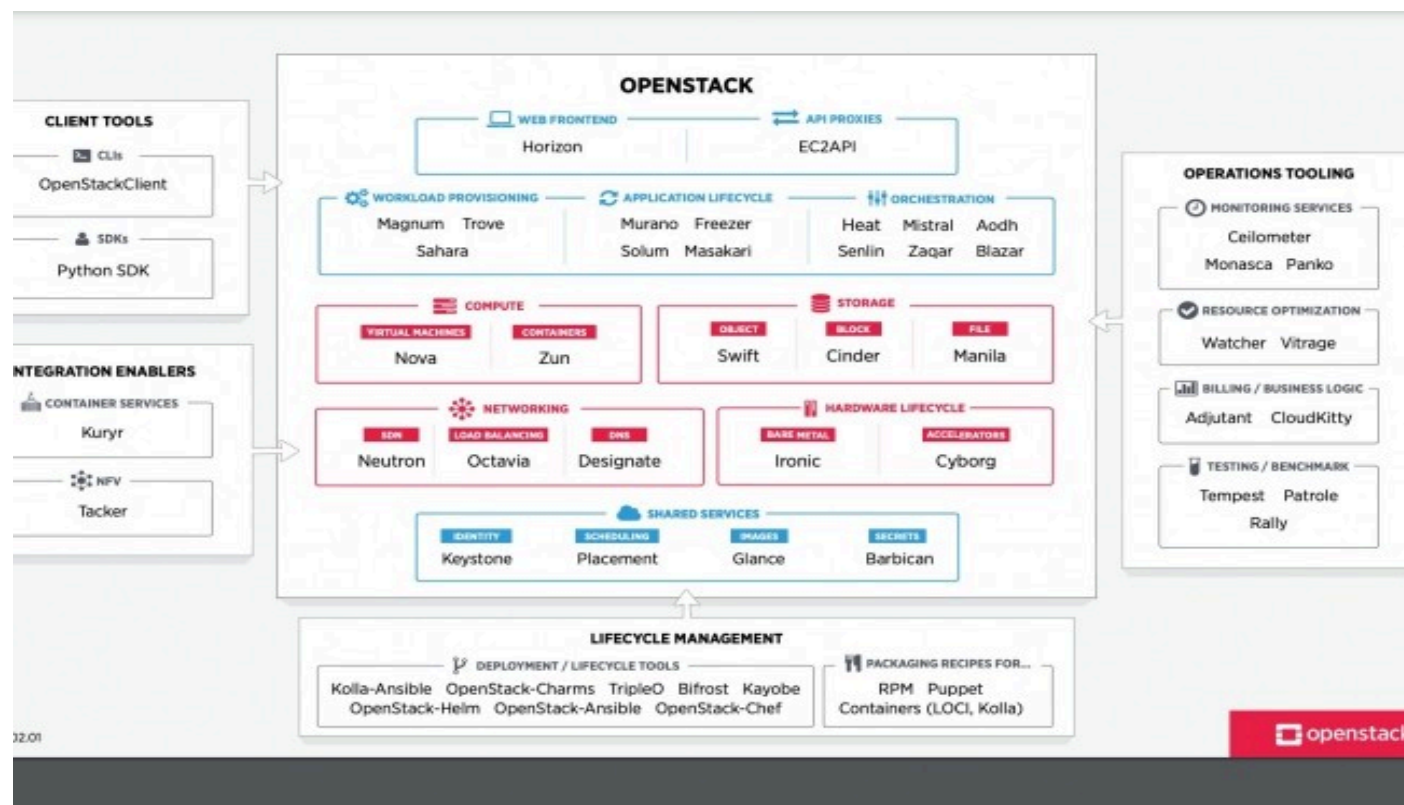
3. Écosystème OpenStack

Intégrations et écosystème

- Support de :
 - **Hyperviseurs** : KVM (par défaut), QEMU, Xen, Hyper-V, VMware ESXi
 - **Stockage** : Ceph, NetApp, EMC, local, S3 compatible
 - **Réseaux** : Cisco, Juniper, Mellanox, Open vSwitch, LinuxBridge
- Intégration possible avec :
 - **Ansible, Terraform, Kubernetes, Prometheus, Grafana**, etc.
 - Outils de supervision et monitoring (Zabbix, ELK...)

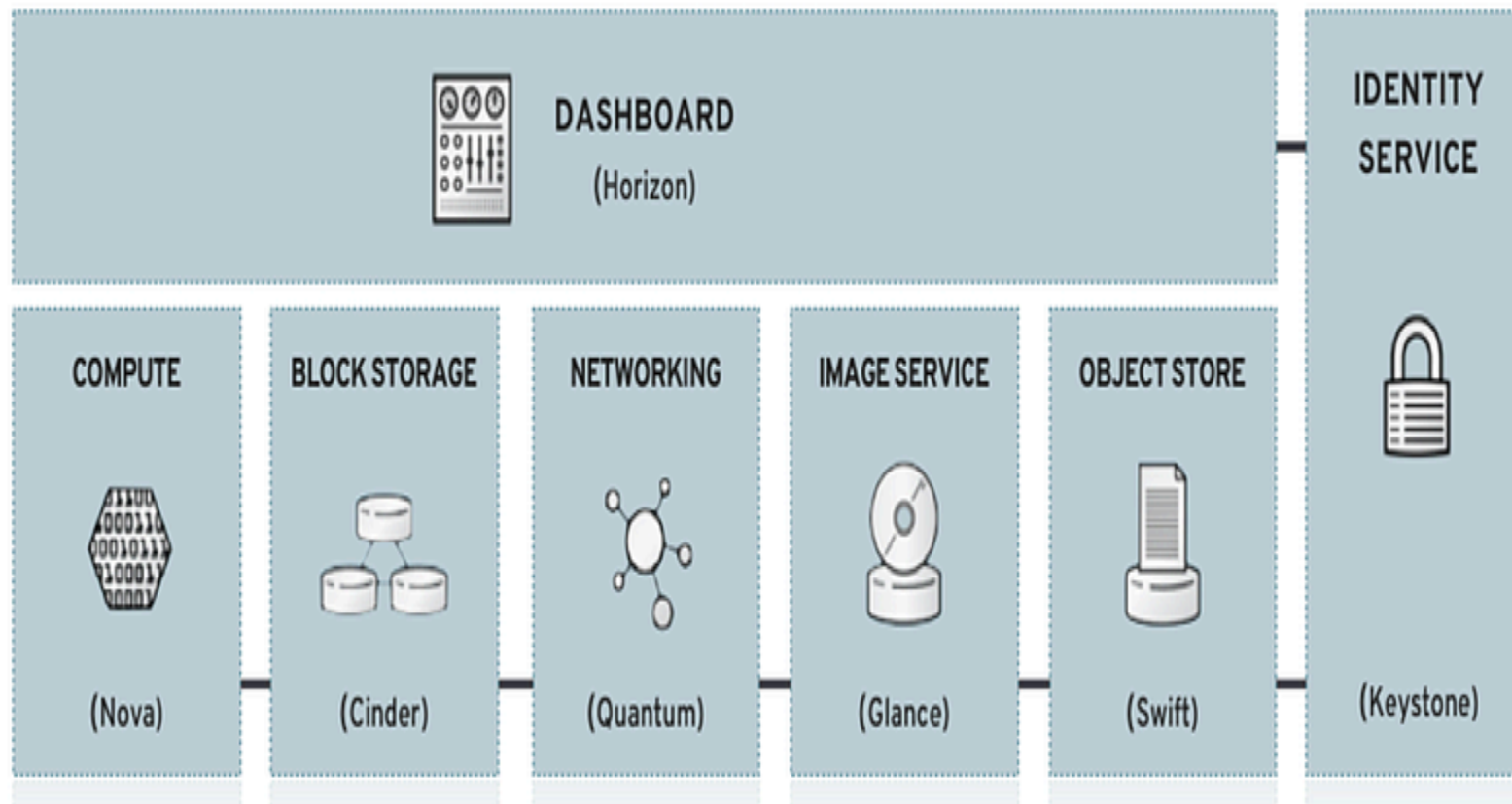
Conception d'un Cloud OpenStack

3. Écosystème OpenStack



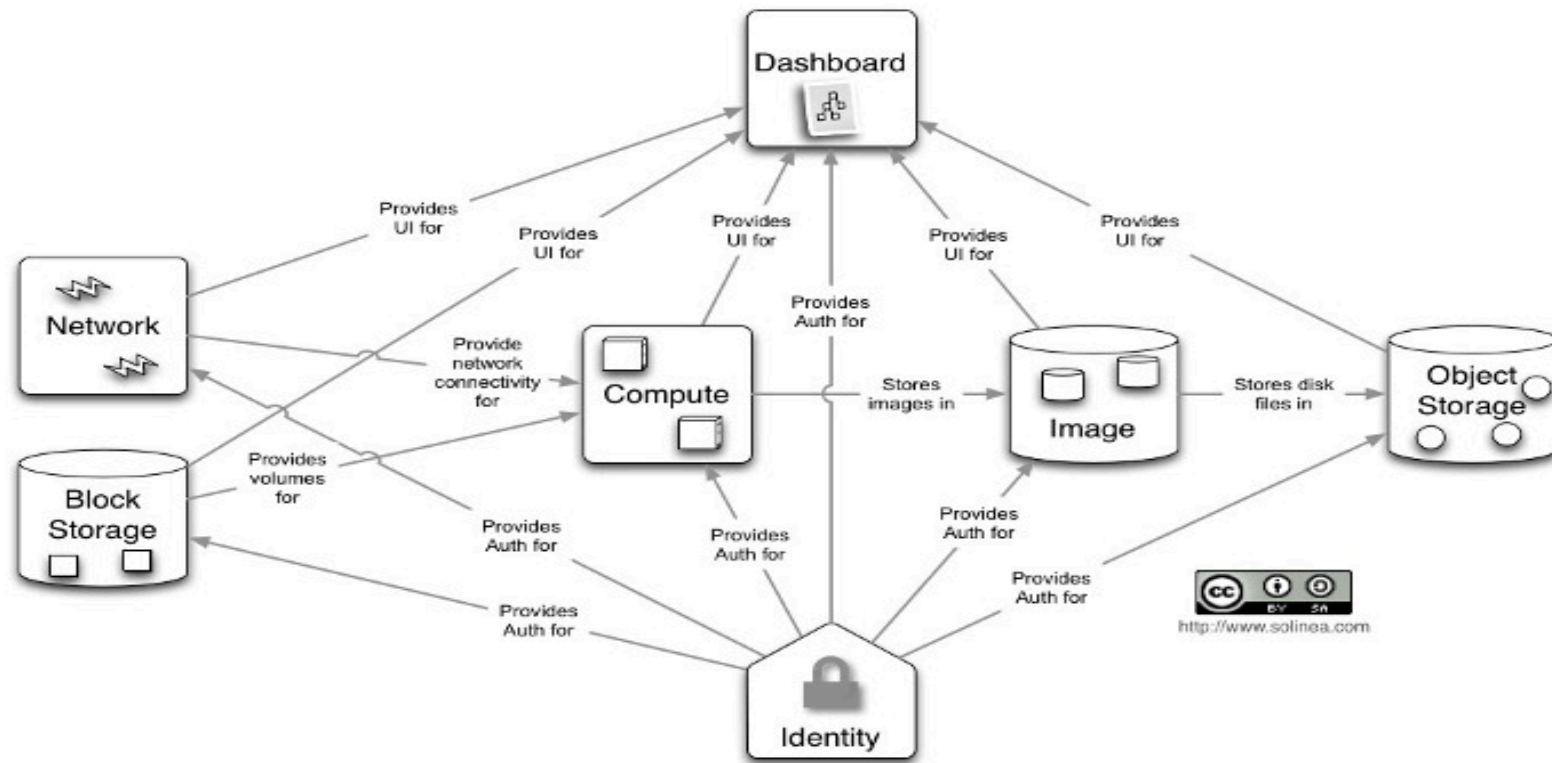
Conception d'un Cloud OpenStack

3. Écosystème OpenStack



Conception d'un Cloud OpenStack

3. Écosystème OpenStack



Conception d'un Cloud OpenStack

1. Les composants principaux (« Core Services »)

Keystone (Service d'identité)

- **Rôle** : Authentification et autorisation (IAM du cloud OpenStack)
- **Fonctions** : Gestion des utilisateurs, rôles, tenants/projets, tokens d'accès
- **Utilisé par** : Tous les autres services pour vérifier les identités

Glance (Service d'images)

- **Rôle** : Gestion des images disques (templates de VM)
- **Fonctions** : Import/export, catalogue d'images, gestion des métadonnées, snapshot d'instances

Conception d'un Cloud OpenStack

1. Les composants principaux (« Core Services »)

Nova (Compute)

- **Rôle** : Orchestration et gestion des machines virtuelles (VM)
- **Fonctions** : Lancer, arrêter, migrer, scheduler des VMs, gestion des flavors (taille des VMs), prise en charge multi-hyperviseur (KVM, Xen, VMware)

Conception d'un Cloud OpenStack

1. Les composants principaux (« Core Services »)

Neutron (Réseau)

- **Rôle** : Gestion avancée des réseaux virtuels pour les instances
- **Fonctions** :
 - Création de réseaux privés/publics
 - Gestion des sous-réseaux, routeurs, DHCP
 - Attribution d'IP flottante (NAT)
 - Groupes de sécurité (pare-feu virtuel)
 - Plugins pour SDN (Open vSwitch, LinuxBridge...)

Conception d'un Cloud OpenStack

1. Les composants principaux (« Core Services »)

Cinder (Block Storage)

- **Rôle** : Fournir des volumes de stockage blocs persistants pour les instances
- **Fonctions** : Création, attachement, détachement de volumes, snapshots de volumes, gestion de backend (LVM, Ceph, NetApp...)

Swift (Object Storage)

- **Rôle** : Stockage d'objets (style S3/Azure Blob)
- **Fonctions** : Upload/download de fichiers, versioning, stockage distribué, haute résilience
- **Usage** : Backup, stockage d'images, partage de fichiers volumineux

Conception d'un Cloud OpenStack

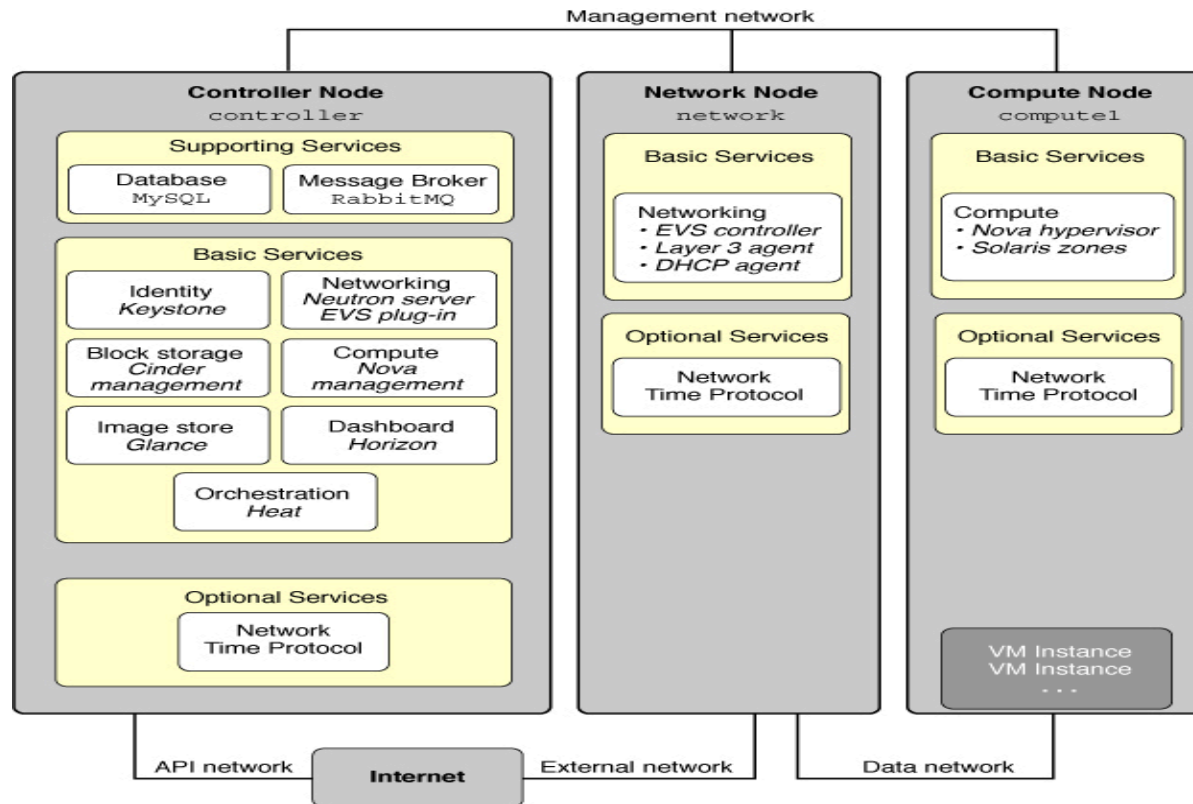
1. Les composants principaux (« Core Services »)

Autres modules importants

- **Horizon** : Tableau de bord Web pour utilisateurs et admins (interface graphique)
- **Heat** : Orchestration de ressources (déploiement automatisé, IaC via templates YAML)
- **Magnum** : Service de déploiement de clusters de conteneurs (Kubernetes, Docker Swarm)

Conception d'un Cloud OpenStack

2. Topologie générale d'un cloud OpenStack



Conception d'un Cloud OpenStack

2. Topologie générale d'un cloud OpenStack

Explications :

- **Contrôleur** : cœur du système (API, gestion centrale, base de données, files de messages)
 - **Nœuds Compute** : hôtes où s'exécutent les VMs
 - **Nœuds Storage** : hôtes qui stockent les volumes (Cinder) et/ou objets (Swift)
 - **Nœuds Réseau** : gèrent la connectivité, les routeurs, le NAT, load balancing
- “ **Note** : Sur de petits labs, tout est parfois sur une seule VM. En production, chaque rôle peut avoir plusieurs serveurs dédiés. ”

Conception d'un Cloud OpenStack

3. Services internes de communication

Explications :

- **RabbitMQ** (ou autre message bus) :
 - Sert de bus de messages asynchrone entre les services (ex : Nova scheduler qui notifie un compute node de lancer une VM)
 - Clé pour la scalabilité et le découplage des composants
- **MariaDB/MySQL** (ou PostgreSQL) :
 - Stockage de toutes les métadonnées (état des ressources, utilisateurs, etc.)
- **etcd** (ou Memcached, Redis, etc.) :
 - Coordination, stockage clé/valeur distribué (utile pour certains services avancés, réseaux, ou haute-disponibilité)

Installation & premiers pas OpenStack

Installation & premiers pas OpenStack

1. Présentation des modes d'installation

A. Solutions tout-en-un (all-in-one)

- **DevStack**

- Script d'installation rapide pour tests, démo, formation.
- *Avantages *: déploiement très simple, installation sur une seule VM (ou machine physique).
- *Limites *: non recommandé pour la production, performances limitées, reset à chaque reboot.
- **Usage : idéal pour la formation, PoC, labs étudiants.**

Installation & premiers pas OpenStack

1. Présentation des modes d'installation

A. Solutions tout-en-un (all-in-one)

- **MicroStack**

- Distribution packagée par Canonical (Ubuntu), très légère, facile à installer.
- *Avantages *: rapide, usage desktop/VM, quelques minutes pour être prêt.
- *Limites *: moins personnalisable, communauté plus petite.
- **Usage : parfait pour tester sur un laptop ou VM.**

Installation & premiers pas OpenStack

1. Présentation des modes d'installation

A. Solutions tout-en-un (all-in-one)

- **Packstack / RDO** (Red Hat)
 - Installation semi-automatisée pour CentOS/RHEL, plus flexible que DevStack, plus adaptée à la pré-prod.
 - **Usage : démonstrations un peu plus proches de la production.**

Installation & premiers pas OpenStack

1. Présentation des modes d'installation

B. Déploiements multi-nœuds (production/réaliste)

- **Kolla-Ansible, TripleO, Juju**
 - Outils d'automatisation pour installer OpenStack sur plusieurs serveurs (production, haute disponibilité).
 - *Avantages *: scalable, modulaire, résilient.
 - *Limites *: plus complexe à mettre en œuvre, nécessite plusieurs machines/VM.
 - **Usage : production, cloud d'entreprise, formations avancées.**

Installation & premiers pas OpenStack

2. Installation rapide : DevStack

Pré-requis

- 1 VM ou machine Ubuntu/Debian (4 Go RAM mini, 2 CPU, 20 Go disque)
- Accès sudo/root
- Connexion Internet

Étapes de base

A. Préparation de la VM

```
# Mise à jour et installation de git  
sudo apt update  
sudo apt install -y git
```

B. Récupération du script DevStack

```
git clone https://opendev.org/openstack/devstack  
cd devstack
```

Installation & premiers pas OpenStack

2. Installation rapide : DevStack

Étapes de base

C. Création d'un fichier de configuration rapide

```
cat > local.conf <<EOF
[[local|localrc]]
ADMIN_PASSWORD=stack
DATABASE_PASSWORD=stack
RABBIT_PASSWORD=stack
SERVICE_PASSWORD=stack
EOF
```

D. Lancement de l'installation

```
./stack.sh
```

- Durée : ~20-30 minutes selon la machine et la connexion.

Installation & premiers pas OpenStack

2. Installation rapide : DevStack

Étapes de base

E. Accès post-installation

- Interface web (Horizon) :
http://<IP_VM>:8080
(ou parfois :80 selon config)
 - user : admin
 - password : celui défini (stack ici)
- Accès CLI :

```
source openrc admin admin  
openstack server list
```

Installation & premiers pas OpenStack

3. Installation rapide : MicroStack (Ubuntu)

A. Installation via snap

```
sudo snap install microstack --devmode --beta
```

B. Initialisation

```
sudo microstack init --auto --control
```

C. Accès

- Interface web : `http://<IP_VM>:80`
- Commandes : `microstack.openstack ...`

Installation & premiers pas OpenStack

4. Connexion au dashboard et à l'API

A. Dashboard Horizon

- Ouvre ton navigateur sur `http://<adresse IP de ta VM>:8080`
- Identifie-toi avec `admin` / `stack` (ou ton mot de passe)

B. Accès en CLI (Terminal)

- Source l'environnement :

```
source ~/devstack/openrc admin admin
```

- Quelques commandes utiles :

```
openstack server list      # Voir les instances
openstack image list       # Voir les images
openstack network list     # Réseaux
openstack flavor list      # Flavors (tailles de VM)
```

C. Accès à l'API (REST)

- Exemple :

```
curl -i http://<IP_VM>:5000/v3 # Vérifie l'API Keystone (identité)
```


Gestion des machines virtuelles

Gestion des machines virtuelles

Présentation de Nova

1. Rôle

- **Nova** est le **service de gestion du calcul** dans OpenStack.
- Il orchestre et pilote :
 - Le **lancement des machines virtuelles (VMs)** sur les hyperviseurs (KVM, QEMU, VMware, Hyper-V...).
 - L'**allocation des ressources** (CPU, RAM, disque).
 - La **planification (scheduling)** des instances sur les nœuds de calcul.
 - L'**interaction avec Neutron** (réseau), **Cinder** (stockage bloc) et **Glance** (images).

Gestion des machines virtuelles

Présentation de Nova

2. Architecture interne

Nova est composé de plusieurs **services** interconnectés via **RabbitMQ** et **API REST** :

- **nova-api** → Reçoit les requêtes des utilisateurs (REST).
- **nova-scheduler** → Choisit le nœud de calcul où déployer la VM.
- **nova-compute** → Déploie la VM sur l'hyperviseur (ex. KVM via libvirt).
- **nova-conductor** → Fait l'intermédiaire entre DB et compute nodes.
- **nova-consoleauth / nova-novncproxy** → Gestion de la console distante (VNC, SPICE).
- **nova-placement** → Service qui gère les ressources disponibles (inventaire CPU/RAM/disk).
- **Base de données (MariaDB/MySQL)** → Stocke l'état des instances et configurations.

Gestion des machines virtuelles

Présentation de Nova

1. Prérequis

- **Services de base déjà installés :**
 - Keystone (identité)
 - Glance (images)
 - Neutron (réseau)
 - RabbitMQ (messagerie)
 - MariaDB/MySQL (base de données)
- **Réseau** configuré (management, provider, tenant).
- **Hyperviseur** : souvent **KVM/QEMU** sur Linux.

Gestion des machines virtuelles

Présentation de Nova

2. Étapes d'installation (exemple sur Ubuntu/Debian)

◆ Sur le contrôleur

1. Créer la base de données Nova

```
CREATE DATABASE nova_api;  
CREATE DATABASE nova;  
GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'localhost' IDENTIFIED BY 'NOVA_PASS';  
GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost' IDENTIFIED BY 'NOVA_PASS';  
FLUSH PRIVILEGES;
```

2. Créer l'utilisateur Nova dans Keystone

```
openstack user create --domain default --password NOVA_PASS nova  
openstack role add --project service --user nova admin  
openstack service create --name nova --description "OpenStack Compute" compute
```

Gestion des machines virtuelles

Présentation de Nova

2. Étapes d'installation (exemple sur Ubuntu/Debian)

◆ Sur le contrôleur

3. Déclarer les endpoints API (public, internal, admin)

```
openstack endpoint create --region RegionOne compute public http://controller:8774/v2.1  
openstack endpoint create --region RegionOne compute internal http://controller:8774/v2.1  
openstack endpoint create --region RegionOne compute admin http://controller:8774/v2.1
```

4. Installer les paquets Nova

```
apt install nova-api nova-conductor nova-novncproxy nova-scheduler
```

Gestion des machines virtuelles

Présentation de Nova

2. Étapes d'installation (exemple sur Ubuntu/Debian)

◆ Sur le contrôleur

5. Configurer Nova (`/etc/nova/nova.conf`)

Exemple (partie importante) :

```
[api_database]
connection = mysql+pymysql://nova:NOVA_PASS@controller/nova_api

[database]
connection = mysql+pymysql://nova:NOVA_PASS@controller/nova

[DEFAULT]
transport_url = rabbit://openstack:RABBIT_PASS@controller
auth_strategy = keystone
my_ip = 10.0.0.11    # IP du contrôleur
```

Gestion des machines virtuelles

Présentation de Nova

2. Étapes d'installation (exemple sur Ubuntu/Debian)

◆ Sur le contrôleur

5. Configurer Nova (`/etc/nova/nova.conf`)

Exemple (partie importante) :

```
[keystone_authtoken]
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = nova
password = NOVA_PASS
```


Gestion des machines virtuelles

Présentation de Nova

2. Étapes d'installation (exemple sur Ubuntu/Debian)

◆ Sur le contrôleur

6. Synchroniser la DB

```
su -s /bin/sh -c "nova-manage api_db sync" nova  
su -s /bin/sh -c "nova-manage db sync" nova
```

7. Démarrer les services

```
systemctl restart nova-api nova-scheduler nova-conductor nova-novncproxy
```

Gestion des machines virtuelles

Présentation de Nova

2. Étapes d'installation (exemple sur Ubuntu/Debian)

◆ Sur le nœud de calcul

1. Installer les paquets

```
apt install nova-compute
```

2. Configurer `/etc/nova/nova.conf`

Exemple :

```
[DEFAULT]
transport_url = rabbit://openstack:RABBIT_PASS@controller
auth_strategy = keystone
my_ip = 10.0.0.31 # IP du compute node
[keystone_auth]
auth_url = http://controller:5000
memcached_servers = controller:11211
username = nova
password = NOVA_PASS
[libvirt]
virt_type = kvm
```

Gestion des machines virtuelles

Présentation de Nova

2. Étapes d'installation (exemple sur Ubuntu/Debian)

◆ Sur le nœud de calcul

3. Redémarrer le service

```
systemctl restart nova-compute
```

Gestion des machines virtuelles

Présentation de Nova

3. Vérification

- Lister les services Nova :

```
openstack compute service list
```

- Lancer une instance :

```
openstack server create --flavor m1.small --image cirros \  
--nic net-id=NETWORK_ID --security-group default --key-name mykey demo-instance
```

Gestion des machines virtuelles

Gestion des images et des instances - Réseau.

Gestion des images (Glance)

Rôle

- **Glance** est le service d'OpenStack dédié à la **gestion des images systèmes** (Linux, Windows, appliances).
- Les images servent de **modèles de VM** à déployer via Nova.

Formats supportés

- RAW, QCOW2 (QEMU/KVM), VMDK (VMware), VHD (Hyper-V), ISO.

Mise en œuvre

1. Téléverser une image

```
openstack image create "Ubuntu-22.04" \  
  --file ubuntu-22.04.qcow2 \  
  --disk-format qcow2 \  
  --container-format bare \
```

Gestion des machines virtuelles

Gestion des images et des instances - Réseau.

Gestion des images (Glance)

Mise en œuvre

2. Lister les images disponibles

```
openstack image list
```

3. Mettre à jour les métadonnées (ex. OS type, archi, taille min RAM/CPU).

Bonnes pratiques

- Utiliser des **images cloud-ready** (Cloud-init installé).
- Gérer un **catalogue d'images validées** par l'entreprise.
- Stocker les images dans **Swift** ou un backend Ceph pour la résilience.

Gestion des machines virtuelles

Gestion des images et des instances - Réseau.

Gestion des instances (Nova)

Rôle

- **Nova** crée et gère les **instances (VMs)**.
- Utilise **Glance** pour l'image, **Neutron** pour le réseau, **Cinder** pour les volumes.

Mise en œuvre

1. Créer une paire de clés SSH

```
openstack keypair create mykey > mykey.pem  
chmod 600 mykey.pem
```

Gestion des machines virtuelles

Gestion des images et des instances - Réseau.

Gestion des instances (Nova)

Mise en œuvre

2. Définir un flavor (gabarit de VM)

```
openstack flavor create --id 1 --ram 2048 --disk 20 --vcpus 2 m1.small
```

3. Lancer une instance

```
openstack server create \  
  --flavor m1.small \  
  --image Ubuntu-22.04 \  
  --nic net-id=NETWORK_ID \  
  --security-group default \  
  --key-name mykey \  
  vm-demo
```


Gestion des machines virtuelles

Gestion des images et des instances - Réseau.

Gestion des instances (Nova)

Mise en œuvre

4. Lister et gérer les instances

```
openstack server list  
openstack server show vm-demo  
openstack server stop vm-demo  
openstack server delete vm-demo
```

Bonnes pratiques

- Automatiser avec **Heat** (orchestration).
- Définir des **quotas** par projet pour contrôler l'usage.
- Sauvegarder des **snapshots** pour réutiliser une VM comme image.

Gestion des machines virtuelles

Gestion des images et des instances - Réseau.

Gestion du réseau virtuel (Neutron)

Rôle

- **Neutron** fournit la **connectivité réseau** aux instances.
- Fonctionne avec des plugins (Open vSwitch, Linux Bridge, OVN, SDN).
- Gère les **réseaux, sous-réseaux, routeurs, flottants, sécurité**.

Composants clés

- **Network** : comme un switch virtuel.
- **Subnet** : plage IP assignée aux instances.
- **Router** : connecte réseaux internes ↔ externes.
- **Security groups** : règles firewall (iptables/OVS).
- **Floating IP** : IP publique NATée vers une VM.

Gestion des machines virtuelles

Gestion des images et des instances - Réseau.

Gestion du réseau virtuel (Neutron)

Mise en œuvre

3. Créer un routeur

```
openstack router create myrouter  
openstack router set myrouter --external-gateway public-net  
openstack router add subnet myrouter private-subnet
```

4. Associer une Floating IP

```
openstack floating ip create public-net  
openstack server add floating ip vm-demo FLOATING_IP
```

Gestion des machines virtuelles

Gestion des images et des instances - Réseau.

Gestion du réseau virtuel (Neutron)

Bonnes pratiques

- Séparer les **réseaux management / data / external**.
- Mettre en place des **Network Policies (security groups)**.
- Utiliser **Octavia** si besoin de load balancers.
- Superviser la charge réseau avec **Ceilometer + Gnocchi**.

Gestion des machines virtuelles

Gestion des images et des instances - Réseau.

Gestion du réseau virtuel (Neutron)

Bonnes pratiques

- Séparer les **réseaux management / data / external**.
- Mettre en place des **Network Policies (security groups)**.
- Utiliser **Octavia** si besoin de load balancers.
- Superviser la charge réseau avec **Ceilometer + Gnocchi**.

Gestion des machines virtuelles

Gestion d'hyperviseurs multiples

1. Rôle des hyperviseurs dans OpenStack

- OpenStack (via **Nova**) **n'exécute pas directement les VM** : il orchestre les **compute nodes** qui utilisent un hyperviseur (libvirt/KVM, ESXi, Hyper-V...).
- Chaque **nœud de calcul** est associé à un hyperviseur.
- Nova interagit avec eux via des **drivers** (appelés *virt drivers*).

Gestion des machines virtuelles

Gestion d'hyperviseurs multiples

2. Hyperviseurs supportés

OpenStack est **agnostique** à l'hyperviseur grâce à ses drivers. Les principaux sont :

- **KVM/QEMU (libvirt)** → le plus courant, performant et open source.
- **VMware ESXi** (via `VMwareVCDriver`) → intégré avec vCenter.
- **Microsoft Hyper-V** (via `HyperVDriver`) → support Windows.
- **Xen/XenServer** (aujourd'hui moins utilisé).
- **Baremetal** (Ironic) → provisionnement direct sans hyperviseur.
- **Containers** (Magnum/Kata) → alternatives modernes.

👉 En pratique : **KVM est le choix par défaut** (Linux + open source), mais OpenStack permet de mélanger.

Gestion des machines virtuelles

Gestion d'hyperviseurs multiples

3. Gestion de plusieurs hyperviseurs dans le même cloud

OpenStack permet de gérer plusieurs types d'hyperviseurs **dans le même cluster**.

- Nova détecte l'hyperviseur du nœud (`nova-compute` sur chaque node).
- L'**administrateur peut définir des "aggregates" et "availability zones"** pour classer les compute nodes (par hyperviseur, CPU, GPU, etc.).
- Lors du déploiement d'une VM, un **flavor** peut être associé à un **extra spec** pour cibler un type d'hyperviseur.

👉 Exemple :

- `flavor1` → VM sur KVM.
- `flavor2` → VM sur ESXi.

Gestion des machines virtuelles

Gestion d'hyperviseurs multiples

4. Configuration des hyperviseurs

a) KVM (par défaut, Linux)

Dans `/etc/nova/nova.conf` :

```
[libvirt]
virt_type = kvm
```

Vérifier que l'hôte supporte la virtualisation matérielle (`egrep -c '(vmx|svm)' /proc/cpuinfo`).

Gestion des machines virtuelles

Gestion d'hyperviseurs multiples

4. Configuration des hyperviseurs

b) VMware ESXi

- Nécessite un **vCenter** ou un cluster ESXi.
- Activer le driver VMware dans `nova.conf` :

```
[DEFAULT]
compute_driver = vmwareapi.VMwareVCDriver

[vmware]
host_ip = <VCENTER_IP>
host_username = <VCENTER_USER>
host_password = <VCENTER_PASS>
cluster_name = <VCENTER_CLUSTER>
```

- Nova utilisera les API vCenter pour gérer les VM.

Gestion des machines virtuelles

Gestion d'hyperviseurs multiples

4. Configuration des hyperviseurs

c) Microsoft Hyper-V

- Installer l'agent **nova-compute-hyperv** sur le serveur Hyper-V (Windows).
- Dans `nova.conf` :

```
[DEFAULT]  
compute_driver = hyperv.nova.driver.HyperVDriver
```

- Nécessite `WinRM` et certains rôles Windows activés.

Gestion des machines virtuelles

Gestion d'hyperviseurs multiples

5. Bonnes pratiques

- **KVM** → recommandé pour la majorité des déploiements (communauté la plus active, meilleure intégration).
- **ESXi** → intéressant si l'entreprise a déjà un parc VMware et veut l'intégrer.
- **Hyper-V** → utile dans les environnements Windows purs.
- **Multi-hyperviseurs** :
 - Créer des **host aggregates** pour regrouper les nœuds par hyperviseur.
 - Utiliser des **flavors avec extra_specs** pour orienter les VM.
 - Exemple :

```
openstack aggregate create kvm_hosts
openstack aggregate set --property hypervisor_type=KVM kvm_hosts

openstack flavor set m1.kvm --property aggregate_instance_extra_specs:hypervisor_type=KVM
```

- **Supervision** : utiliser `openstack hypervisor list` pour voir les hôtes disponibles.

Gestion des machines virtuelles

Gestion d'hyperviseurs multiples

En résumé

- **Nova** gère plusieurs hyperviseurs grâce à des drivers.
- **KVM est le plus utilisé**, mais VMware ESXi et Hyper-V sont supportés.
- On peut mélanger plusieurs hyperviseurs dans le même cloud → grâce aux **aggregates, zones et flavors**.
- Le choix dépend du **contexte entreprise** :
 - Open source/Linux → **KVM**
 - Parc VMware → **ESXi**
 - Environnement Windows → **Hyper-V**

Gestion du stockage

Gestion du stockage

Vue d'ensemble de Swift

OpenStack Swift – Vue d'ensemble

1. Rôle

- **Swift** est le service d'**Object Storage** d'OpenStack.
- Il permet de stocker et de récupérer des objets (fichiers, images, vidéos, backups) dans un espace distribué et hautement disponible.
- Fonctionne sur le même principe qu'**Amazon S3**.
- Utilisé pour :
 - Sauvegardes
 - Archives
 - Distribution de contenus (CDN)
 - Backend de stockage pour Glance (images de VM)

Gestion du stockage

Vue d'ensemble de Swift

OpenStack Swift – Vue d'ensemble

2. Architecture logique

Swift est composé de **deux parties principales** :

a) **Proxy servers**

- Point d'entrée pour les clients.
- Reçoivent les requêtes via l'**API REST** (compatible S3).
- Gèrent l'authentification (via Keystone).
- Distribuent la requête vers le bon nœud de stockage.

Gestion du stockage

Vue d'ensemble de Swift

OpenStack Swift – Vue d'ensemble

2. Architecture logique

Swift est composé de **deux parties principales** :

b) Storage nodes

- Stockent physiquement les objets.
- Trois types de services :
 - **Object server** → stocke les objets.
 - **Container server** → gère la liste des objets (métadonnées).
 - **Account server** → gère les comptes/projets et quotas.

Gestion du stockage

Vue d'ensemble de Swift

OpenStack Swift – Vue d'ensemble

2. Architecture logique

c) Anneaux (rings)

- Fichiers de configuration qui indiquent où stocker/récupérer les objets.
- Gérés par un **ring builder**.
- Assurent la répartition des données (hashing) et la tolérance aux pannes.

Gestion du stockage

Vue d'ensemble de Swift

OpenStack Swift – Vue d'ensemble

3. Fonctionnement

1. Un client envoie un fichier via l'API Swift (`PUT object`).
2. Le proxy server :
 - Authentifie la requête via Keystone.
 - Calcule où placer l'objet dans l'anneau.
 - Redirige vers les storage nodes concernés.
3. L'objet est stocké en **plusieurs copies (réplication)** sur différents nœuds.
4. En cas de panne d'un nœud, une copie est automatiquement restaurée ailleurs.

Gestion du stockage

Vue d'ensemble de Swift

OpenStack Swift – Vue d'ensemble

4. API et utilisation

a) Commandes principales

- **Créer un container** (équivalent à un “bucket”) :

```
openstack container create backups
```

- **Lister les containers** :

```
openstack container list
```

- **Uploader un fichier** :

```
openstack object create backups /home/user/file.txt
```

Gestion du stockage

Vue d'ensemble de Swift

OpenStack Swift – Vue d'ensemble

4. API et utilisation

a) Commandes principales

- **Télécharger un objet :**

```
openstack object save backups file.txt
```

- **Supprimer un objet :**

```
openstack object delete backups file.txt
```

Gestion du stockage

Vue d'ensemble de Swift

OpenStack Swift – Vue d'ensemble

4. API et utilisation

b) Points forts

- API REST (compatible S3).
- Multi-tenants (via Keystone).
- Métadonnées extensibles.

Gestion du stockage

Vue d'ensemble de Swift

OpenStack Swift – Vue d'ensemble

5. Avantages de Swift

- **Scalabilité horizontale** (ajout de nouveaux nœuds sans interruption).
- **Haute disponibilité** grâce à la réplication automatique.
- **Pas de SPOF (single point of failure)** : architecture distribuée.
- **Multi-pétabyte possible.**
- **Stockage orienté objet** → pratique pour fichiers statiques, médias, archives.

Gestion du stockage

Vue d'ensemble de Swift

OpenStack Swift – Vue d'ensemble

6. Limites

- Pas adapté pour du **stockage bloc** (VM, bases de données → utiliser **Cinder**).
- Pas de système de fichiers classique (pas de hiérarchie type ext4/NTFS).
- Performances moindres que du stockage local pour les petites I/O.

Gestion du stockage

Vue d'ensemble de Swift

OpenStack Swift – Vue d'ensemble

En résumé

- **Swift = stockage objet** distribué, tolérant aux pannes, scalable.
- Composé de **proxy servers, storage nodes et rings**.
- Permet de gérer **containers et objets** via une API REST.
- Idéal pour **sauvegardes, archives, médias, CDN**.
- Peut servir de **backend pour Glance** (stockage des images VM).

Gestion du stockage

Mise en œuvre de Swift

1. Prérequis

- Un cluster OpenStack déjà installé avec **Keystone** (authentification).
- **RabbitMQ** et **MariaDB** opérationnels.
- Plusieurs serveurs (ou disques) disponibles pour stocker les objets.
- Résolution DNS ou fichier `/etc/hosts` configuré (ex : `controller`, `swift-storage1`, etc.).

Gestion du stockage

Mise en œuvre de Swift

2. Architecture cible

- **Controller node :**
 - Proxy Server (entrée des requêtes REST)
 - Ring builder (génère les anneaux)
- **Storage nodes :**
 - Account server
 - Container server
 - Object server
 - Stockage physique des objets (disques / partitions montées sur `/srv/node/`)

Gestion du stockage

Mise en œuvre de Swift

3. Étapes d'installation

a) Sur le controller (proxy node)

1. Créer la base de données pour Swift

```
CREATE DATABASE swift;  
GRANT ALL PRIVILEGES ON swift.* TO 'swift'@'localhost' IDENTIFIED BY 'SWIFT_PASS';  
FLUSH PRIVILEGES;
```

2. Créer l'utilisateur dans Keystone

```
openstack user create --domain default --password SWIFT_PASS swift  
openstack role add --project service --user swift admin  
openstack service create --name swift --description "OpenStack Object Storage" object-store  
openstack endpoint create --region RegionOne object-store public http://controller:8080/v1/AUTH_%(project_id)s  
openstack endpoint create --region RegionOne object-store internal http://controller:8080/v1/AUTH_%(project_id)s  
openstack endpoint create --region RegionOne object-store admin http://controller:8080/v1
```

Gestion du stockage

Mise en œuvre de Swift

3. Étapes d'installation

a) Sur le controller (proxy node)

3. Installer les paquets Swift Proxy

```
apt install swift swift-proxy python3-swiftclient \  
python3-keystoneclient python3-keystonemiddleware \  
memcached
```

Gestion du stockage

Mise en œuvre de Swift

3. Étapes d'installation

a) Sur le controller (proxy node)

4. Configurer `/etc/swift/proxy-server.conf`

Exemple minimal :

```
[DEFAULT]
bind_port = 8080
user = swift
swift_dir = /etc/swift

[pipeline:main]
pipeline = catch_errors gatekeeper healthcheck proxy-logging cache authtoken keystoneauth proxy-logging proxy-server

[app:proxy-server]
use = egg:swift#proxy
account_autocreate = true
```

Gestion du stockage

Mise en œuvre de Swift

3. Étapes d'installation

b) Sur les storage nodes

1. Installer Swift et dépendances

```
apt install swift swift-account swift-container swift-object xfsprogs rsync
```

2. Préparer les disques

Exemple avec `/dev/sdb` :

```
mkfs.xfs /dev/sdb  
mkdir -p /srv/node/sdb  
echo "/dev/sdb /srv/node/sdb xfs noatime,nodiratime,nobarrier,logbufs=8 0 0" >> /etc/fstab  
mount -a
```

Gestion du stockage

Mise en œuvre de Swift

3. Étapes d'installation

b) Sur les storage nodes

3. Configurer rsync (`/etc/rsyncd.conf`) :

```
uid = swift
gid = swift
[account]
path = /srv/node/
read only = false
[container]
path = /srv/node/
read only = false
[object]
path = /srv/node/
read only = false
```


Gestion du stockage

Mise en œuvre de Swift

3. Étapes d'installation

c) Création des rings (depuis le controller)

- **Account ring :**

```
swift-ring-builder account.builder create 10 3 1
swift-ring-builder account.builder add --region 1 --zone 1 --ip 10.0.0.21 --port 6002 --device sdb --weight 100
swift-ring-builder account.builder rebalance
```

- **Container ring :**

```
swift-ring-builder container.builder create 10 3 1
swift-ring-builder container.builder add --region 1 --zone 1 --ip 10.0.0.21 --port 6001 --device sdb --weight 100
swift-ring-builder container.builder rebalance
```

Gestion du stockage

Mise en œuvre de Swift

3. Étapes d'installation

c) Création des **rings** (depuis le controller)

- **Object ring :**

```
swift-ring-builder object.builder create 10 3 1  
swift-ring-builder object.builder add --region 1 --zone 1 --ip 10.0.0.21 --port 6000 --device sdb --weight 100  
swift-ring-builder object.builder rebalance
```

👉 Les fichiers générés (`*.ring.gz`) doivent être copiés sur **tous les nœuds Swift** (`/etc/swift/`).

Gestion du stockage

Mise en œuvre de Swift

3. Étapes d'installation

d) Démarrer les services

- Sur le **proxy** :

```
systemctl restart swift-proxy
```

- Sur les **storage nodes** :

```
systemctl restart swift-account swift-container swift-object
```

Gestion du stockage

Mise en œuvre de Swift

4. ♦ Vérification

- **Lister les containers (buckets) :**

```
openstack container list
```

- **Créer un container :**

```
openstack container create backups
```

- **Uploader un objet :**

```
openstack object create backups /etc/hosts
```

- **Télécharger un objet :**

```
openstack object save backups hosts
```

Gestion du stockage

Mise en œuvre de Swift

En résumé

- **Swift proxy** : point d'entrée, configuré sur le controller.
- **Storage nodes** : stockent réellement les objets (account, container, object).
- **Rings** : déterminent la répartition des données → doivent être identiques sur tous les nœuds.
- **Keystone** : gère l'authentification des utilisateurs.

Gestion du stockage

Mise en œuvre et configuration.

En résumé

- **Swift proxy** : point d'entrée, configuré sur le controller.
- **Storage nodes** : stockent réellement les objets (account, container, object).
- **Rings** : déterminent la répartition des données → doivent être identiques sur tous les nœuds.
- **Keystone** : gère l'authentification des utilisateurs.

Gestion du stockage

Mise en œuvre de Swift

3. Étapes d'installation

b) Sur les storage nodes

4. **Configurer Swift sur chaque service** (`/etc/swift/account-server.conf`, `/etc/swift/container-server.conf`, `/etc/swift/object-server.conf`) :

Exemple pour object :

```
[DEFAULT]
devices = /srv/node
mount_check = true
bind_ip = 0.0.0.0
bind_port = 6000
user = swift
swift_dir = /etc/swift
```

Gestion du stockage

Pools de stockage dans OpenStack

Cela touche surtout deux briques :

- **Cinder (Block Storage)** → gestion des **pools de volumes** (LVM, Ceph, NetApp, etc.)
- **Swift (Object Storage)** → gestion via **anneaux (rings)** et **réplication**
- (Accessoirement **Nova/Glance** peuvent utiliser ces backends, mais ne gèrent pas directement les pools)

Gestion du stockage

Pools de stockage dans OpenStack

1. Pools côté Cinder (Block Storage)

Qu'est-ce qu'un pool de stockage ?

- Un **pool** regroupe un ensemble de disques ou un backend de stockage.
- Cinder peut avoir **plusieurs backends** → chacun devient un pool.
- Exemple :
 - `pool_hdd` (LVM sur HDD → grande capacité, lent)
 - `pool_ssd` (Ceph SSD → rapide)
 - `pool_netapp` (NAS ou SAN externe)

Gestion du stockage

Pools de stockage dans OpenStack

1. Pools côté Cinder (Block Storage)

Déclaration dans `/etc/cinder/cinder.conf`

Exemple avec deux backends :

```
[DEFAULT]
enabled_backends = lvm,ceph

[lvm]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_group = cinder-volumes
volume_backend_name = LVM_POOL

[ceph]
volume_driver = cinder.volume.drivers.rbd.RBDDriver
rbd_pool = volumes
rbd_user = cinder
rbd_ceph_conf = /etc/ceph/ceph.conf
volume_backend_name = CEPH_POOL
```

Gestion du stockage

Pools de stockage dans OpenStack

1. Pools côté Cinder (Block Storage)

Association avec des volumes

Quand on crée un volume, on peut cibler un pool précis :

```
openstack volume create --size 20 --type fast_vol myvolume
```

Ici `fast_vol` est un **volume type** lié au pool `CEPH_POOL`.

Bonnes pratiques

- Créer des **volume types** pour orienter les workloads (ex. `gold` = SSD, `silver` = HDD).
- Surveiller la capacité avec :

```
openstack volume service list  
cinder pool-list
```

Gestion du stockage

Pools de stockage dans OpenStack

2. Pools côté Swift (Object Storage)

Swift n'utilise pas le mot "pool", mais fonctionne avec un concept équivalent : **les rings et la réplication**.

◆ Fonctionnement

- Les objets sont stockés sur plusieurs **storage nodes**.
- Le **ring** (anneau) détermine où un objet doit être placé.
- Chaque objet est **répliqué N fois** (par défaut 3 copies) sur différents disques/nœuds/zones.
- Cela assure tolérance aux pannes → si un disque tombe, une copie est recrée ailleurs.

Gestion du stockage

Pools de stockage dans OpenStack

2. Pools côté Swift (Object Storage)

Swift n'utilise pas le mot "pool", mais fonctionne avec un concept équivalent : **les rings et la réplication**.

Exemple de réplication

- Un objet est stocké sur :
 - `swift-storage1:/srv/node/sdb`
 - `swift-storage2:/srv/node/sdc`
 - `swift-storage3:/srv/node/sdd`

Gestion du stockage

Pools de stockage dans OpenStack

2. Pools côté Swift (Object Storage)

Swift n'utilise pas le mot "pool", mais fonctionne avec un concept équivalent : **les rings et la réplication**.

Commandes de gestion

- Construire un anneau (ring) :

```
swift-ring-builder object.builder add --region 1 --zone 1 \  
  --ip 10.0.0.21 --port 6000 --device sdb --weight 100  
swift-ring-builder object.builder rebalance
```

- Voir l'état des réplicas :

```
swift-recon --replication
```

Gestion du stockage

Pools de stockage dans OpenStack

Bonnes pratiques

- Placer les réplicas dans des **zones différentes** (serveurs ou racks différents).
- Ajuster le **weight** pour équilibrer la charge entre disques.
- Utiliser un **erasure coding** (au lieu de réplication) pour optimiser la capacité sur gros clusters.

Gestion du stockage

Pools de stockage dans OpenStack

3. Pools dans une architecture OpenStack typique

- **Nova** (Compute) → ne gère pas de pool, mais utilise les volumes de **Cinder**.
- **Glance** (Images) → peut stocker ses images dans Swift ou Cinder.
- **Cinder** (Block Storage) → pools = backends (LVM, Ceph, SAN).
- **Swift** (Object Storage) → pools = rings + réplication.

Gestion du stockage

Pools de stockage dans OpenStack

En résumé

- **Cinder** : pools = backends de stockage (LVM, Ceph, NetApp, etc.).
 - On les associe à des **volume types** pour orienter les VM.
- **Swift** : pools implicites via **rings et réplication**.
 - Chaque objet est stocké sur plusieurs disques/nœuds.
- Gestion des pools = **stratégie de placement + tolérance aux pannes + performance**.

Gestion du stockage

Cinder – Mise en œuvre du stockage bloc

1. Rôle

- Fournir des **volumes persistants** (comme des disques durs virtuels).
- Géré par le **Volume Service (cinder-volume)**.
- Les volumes peuvent être :
 - **attachés/détachés** à des instances (Nova).
 - **clonés ou sauvegardés**.
 - basés sur différents **backends** (LVM, Ceph, NetApp, SAN, etc.).

Gestion du stockage

Cinder – Mise en œuvre du stockage bloc

2. Architecture Cinder

- **cinder-api** → reçoit les requêtes REST.
- **cinder-scheduler** → choisit le backend/pool approprié.
- **cinder-volume** → gère les volumes sur le backend.
- **cinder-backup** (optionnel) → sauvegarde des volumes.
- **Base de données (MariaDB)** → stocke la config et état des volumes.
- **RabbitMQ** → bus de messages entre services.

Gestion du stockage

Cinder – Mise en œuvre du stockage bloc

3. Mise en œuvre (exemple avec LVM)

a) Prérequis

- Un nœud de stockage avec un disque libre (/dev/sdb).
- Keystone, RabbitMQ, MariaDB déjà en place.

b) Sur le controller node

1. Créer la DB Cinder

```
CREATE DATABASE cinder;  
GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'localhost' IDENTIFIED BY 'CINDER_PASS';  
GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%' IDENTIFIED BY 'CINDER_PASS';  
FLUSH PRIVILEGES;
```

Gestion du stockage

Cinder – Mise en œuvre du stockage bloc

3. Mise en œuvre (exemple avec LVM)

b) Sur le controller node

2. Créer l'utilisateur Keystone et le service

```
openstack user create --domain default --password CINDER_PASS cinder
openstack role add --project service --user cinder admin
openstack service create --name cinderv2 --description "OpenStack Block Storage" volumev2
openstack service create --name cinderv3 --description "OpenStack Block Storage" volumev3
```

3. Créer les endpoints

```
openstack endpoint create --region RegionOne volumev3 public http://controller:8776/v3/%(project_id)s
openstack endpoint create --region RegionOne volumev3 internal http://controller:8776/v3/%(project_id)s
openstack endpoint create --region RegionOne volumev3 admin http://controller:8776/v3/%(project_id)s
```

Gestion du stockage

Cinder – Mise en œuvre du stockage bloc

b) Sur le controller node

4. Installer les paquets Cinder

```
apt install cinder-api cinder-scheduler
```

5. Configurer `/etc/cinder/cinder.conf` (controller)

```
[database]
connection = mysql+pymysql://cinder:CINDER_PASS@controller/cinder

[DEFAULT]
transport_url = rabbit://openstack:RABBIT_PASS@controller
auth_strategy = keystone
my_ip = 10.0.0.11
enabled_backends = lvm
glance_api_servers = http://controller:9292
```

Gestion du stockage

Cinder – Mise en œuvre du stockage bloc

b) Sur le controller node

5. Configurer `/etc/cinder/cinder.conf` (controller)

```
[lvm]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_group = cinder-volumes
iscsi_protocol = iscsi
iscsi_helper = tgtadm
volume_backend_name = LVM_POOL
```

6. Init DB et redémarrage

```
su -s /bin/sh -c "cinder-manage db sync" cinder
systemctl restart cinder-api cinder-scheduler
```

Gestion du stockage

Cinder – Mise en œuvre du stockage bloc

c) Sur le storage node

1. Installer les paquets

```
apt install cinder-volume lvm2 tgt
```

2. Préparer le VG LVM

```
pvcreate /dev/sdb  
vgcreate cinder-volumes /dev/sdb
```


Gestion du stockage

Cinder – Mise en œuvre du stockage bloc

c) Sur le storage node

3. Configurer `/etc/cinder/cinder.conf` (storage node)

```
[DEFAULT]
transport_url = rabbit://openstack:RABBIT_PASS@controller
auth_strategy = keystone
my_ip = 10.0.0.31
enabled_backends = lvm

[lvm]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_group = cinder-volumes
iscsi_protocol = iscsi
iscsi_helper = tgtadm
volume_backend_name = LVM_POOL
```

4. Démarrer le service

```
systemctl restart cinder-volume tgt
```

Gestion du stockage

Cinder – Mise en œuvre du stockage bloc

4. Vérification

- Vérifier les services :

```
openstack volume service list
```

- Créer un volume :

```
openstack volume create --size 1 test-vol
```

- Attacher à une instance :

```
openstack server add volume VM_ID test-vol
```

- Vérifier :

```
openstack volume list
```

Gestion du stockage

Cinder – Mise en œuvre du stockage bloc

5. Bonnes pratiques

- Utiliser **Ceph** au lieu de LVM en production (scalabilité, redondance).
- Créer plusieurs **volume types** pour orienter les workloads (SSD, HDD, gold, silver).
- Activer **cinder-backup** pour sauvegarder les volumes vers Swift/Ceph.
- Surveiller avec :

```
cinder list  
cinder pool-list
```

``

Gestion du stockage

Cinder – Mise en œuvre du stockage bloc

En résumé

- **Cinder = stockage bloc** → volumes persistants, attachables aux VM.
- **Services principaux** : cinder-api, cinder-scheduler, cinder-volume.
- **Backends possibles** : LVM (simple), Ceph (production), NetApp/SAN (enterprise).
- Mise en œuvre = configuration du **controller (API + scheduler) + storage node (volume backend)**.
- Les volumes sont ensuite consommés par **Nova (VMs)**.

Gestion du stockage

Backend supportés par Cinder

1. Backends logiciels (open source)

- **LVM (Logical Volume Manager)**
 - Backend par défaut, simple à mettre en place.
 - Fournit des volumes via iSCSI.
 - Usage : **tests, labs**, pas recommandé en production à grande échelle.
- **Ceph RBD (RADOS Block Device)**
 - Backend le plus utilisé en production.
 - Fournit des volumes distribués, redondants, scalables.
 - Intégration native avec Nova, Glance, Cinder.
 - Usage : **production cloud privé/public**, haute disponibilité.

Gestion du stockage

Backend supportés par Cinder

2. Backends matériels (storage enterprise)

Cinder supporte de nombreux **baies de stockage** via des drivers fournis par les constructeurs.

Exemples :

- **NetApp** (ONTAP, SolidFire, E-Series).
- **EMC / Dell EMC** (VNX, VMAX, PowerMax, XtremIO).
- **IBM** (Storwize, Spectrum Scale, DS8000).
- **Hitachi** (VSP).
- **Fujitsu ETERNUS**.
- **Pure Storage FlashArray**.

👉 Ces drivers permettent à OpenStack d'exposer des volumes sur ces systèmes via **iSCSI, FC, NVMe-oF**.

Gestion du stockage

Backend supportés par Cinder

3. Backends cloud & virtuels

- **NFS (Network File System)**
 - Stockage via partage NFS.
 - Utilisé pour des besoins simples, pas toujours optimal pour VM.
- **GlusterFS**
 - Système de fichiers distribué, utilisable comme backend bloc.
- **Sheepdog** (moins utilisé aujourd'hui).
- **DRBD (Distributed Replicated Block Device)**
 - Réplication synchrone des volumes entre nœuds.

Gestion du stockage

Backend supportés par Cinder

4. Protocoles de connexion supportés

- **iSCSI** (le plus courant).
- **Fibre Channel (FC)** pour environnements datacenter.
- **NVMe over Fabrics (NVMe-oF)** pour très haute performance.
- **RBD (Ceph)** pour clusters distribués.

Gestion du stockage

Backend supportés par Cinder

5. Multi-backends

Cinder peut gérer plusieurs backends **en parallèle** :

- Exemple dans `/etc/cinder/cinder.conf` :

```
[DEFAULT]
enabled_backends = lvm,ceph,netapp

[lvm]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_group = cinder-volumes
volume_backend_name = LVM_POOL

[ceph]
volume_driver = cinder.volume.drivers.rbd.RBDDriver
rbd_pool = volumes
rbd_user = cinder
volume_backend_name = CEPH_POOL

[netapp]
volume_driver = cinder.volume.drivers.netapp.common.NetAppDriver
netapp_storage_protocol = iscsi
netapp_backend_name = NETAPP_POOL
```

Gestion du stockage

Backend supportés par Cinder

5. Multi-backends

- On peut créer des **volume types** liés à chaque backend :

```
openstack volume type create ceph_vol  
openstack volume type set ceph_vol --property volume_backend_name=CEPH_POOL
```

Gestion du stockage

Backend supportés par Cinder

En résumé

- **Backends logiciels** : LVM (simple), Ceph (scalable, production).
- **Backends matériels** : NetApp, EMC, IBM, Pure Storage, etc.
- **Backends réseau/FS** : NFS, GlusterFS, DRBD.
- **Protocoles supportés** : iSCSI, FC, NVMe-oF, RBD.
- **Multi-backends** possibles → orienter les workloads via des **volume types**.

Gestion du stockage

Backend supportés par Cinder

En résumé

- **Backends logiciels** : LVM (simple), Ceph (scalable, production).
- **Backends matériels** : NetApp, EMC, IBM, Pure Storage, etc.
- **Backends réseau/FS** : NFS, GlusterFS, DRBD.
- **Protocoles supportés** : iSCSI, FC, NVMe-oF, RBD.
- **Multi-backends** possibles → orienter les workloads via des **volume types**.

Gestion des images

Gestion des images

Qu'est-ce qu'une image ?

1. Définition

- Une **image** est un **modèle de machine virtuelle** :
 - Un fichier qui contient un **système d'exploitation préinstallé** (Linux, Windows, BSD, etc.), éventuellement avec des logiciels ou configurations spécifiques.
 - C'est la **base** utilisée par **Nova (Compute)** pour créer une **instance** (VM).
- Dans OpenStack, les images sont gérées par le service **Glance**.

Gestion des images

Qu'est-ce qu'une image ?

2. Rôle d'une image

- **Point de départ d'une VM** : quand tu lances une instance, Nova prend l'image choisie (par ex. "Ubuntu 22.04 cloud") et la déploie sur l'hyperviseur.
- **Standardisation** : permet d'avoir des environnements identiques pour tous les utilisateurs.
- **Gain de temps** : pas besoin d'installer l'OS manuellement.
- **Automatisation** : certaines images sont préparées avec **Cloud-init**, ce qui permet de personnaliser (hostname, clés SSH, scripts) au boot.

Gestion des images

Qu'est-ce qu'une image ?

3. Formats d'images supportés

OpenStack/Glance supporte plusieurs formats :

- **QCOW2** (QEMU/KVM) → très utilisé, supporte la compression et snapshots.
- **RAW** → format brut, rapide mais volumineux.
- **VMDK** → utilisé par VMware.
- **VHD / VHDX** → utilisé par Hyper-V.
- **ISO** → installation classique (comme un CD/DVD).

Gestion des images

Qu'est-ce qu'une image ?

4. Types d'images

- **OS de base** : Ubuntu, CentOS, Windows, Debian, etc.
- **Images customisées** : ajout d'applications (ex : "Ubuntu + Apache + MySQL").
- **Snapshots** : captures d'une instance existante → réutilisable comme nouvelle image.

Gestion des images

Qu'est-ce qu'une image ?

5. Exemple de gestion (CLI)

- **Lister les images disponibles :**

```
openstack image list
```

- **Importer une image** (ex : Ubuntu QCOW2) :

```
openstack image create "Ubuntu-22.04" \  
  --file ubuntu-22.04.qcow2 \  
  --disk-format qcow2 \  
  --container-format bare \  
  --public
```

- **Utiliser une image pour lancer une VM :**

```
openstack server create --flavor m1.small --image Ubuntu-22.04 \  
  --nic net-id=private-net vm1
```

Gestion des images

Qu'est-ce qu'une image ?

Synthèse

- Une **image** = **modèle de VM** contenant un système d'exploitation (et éventuellement des logiciels).
- Les images sont gérées par **Glance**.
- Elles servent à **créer des instances** via Nova.
- Formats courants : **QCOW2, RAW, VMDK, VHD, ISO**.
- On peut créer ses propres images ou utiliser des images officielles.

Gestion des images

OpenStack Glance – Gestion des images

1. Rôle de Glance

- **Glance** est le service OpenStack chargé de la **gestion des images disque** (modèles de VM).
 - Il fournit une **API REST** pour :
 - Stocker, retrouver et distribuer des images.
 - Servir de source pour **Nova (Compute)** lorsqu'une instance est créée.
 - Être backend pour **Cinder (snapshots)** ou **Swift (stockage objet)**.
- 👉 Sans Glance, Nova ne saurait pas d'où prendre les systèmes d'exploitation pour créer les instances.

Gestion des images

OpenStack Glance – Gestion des images

2. ♦ Fonctionnalités principales

- **Gestion des images OS** (Ubuntu, CentOS, Windows, etc.).
- **Support de plusieurs formats** : QCOW2, RAW, VMDK, VHD, ISO.
- **Métadonnées** : chaque image contient des infos (taille min RAM, architecture CPU, OS type, etc.).
- **Snapshots** : possibilité de capturer l'état d'une VM et de l'enregistrer comme image réutilisable.
- **Partage** : images privées (par projet) ou publiques (pour tous).
- **Backends multiples** :
 - Swift (Object Storage)
 - Cinder (Block Storage)
 - Filesystem (local)
 - Ceph RBD

Gestion des images

OpenStack Glance – Gestion des images

3. Architecture de Glance

Glance est composé de plusieurs services :

- **glance-api** → reçoit les requêtes des utilisateurs (upload, download, list).
- **glance-registry** (*déprécié*) → stockait les métadonnées (intégré dans API désormais).
- **Base de données** → enregistre les métadonnées des images (nom, format, taille, propriétaire).
- **Backend de stockage** → stocke réellement les fichiers (Swift, Ceph, LVM, FS).

Gestion des images

OpenStack Glance – Gestion des images

4. Exemple de cycle de vie d'une image

1. L'administrateur **importe une image** (ex : Ubuntu-22.04.qcow2) dans Glance.
2. Glance **stocke le fichier** dans un backend (ex : Swift).
3. L'utilisateur demande à Nova de **lancer une VM** avec cette image.
4. Nova récupère l'image auprès de Glance → la déploie sur l'hyperviseur.
5. L'utilisateur peut faire un **snapshot de la VM** → enregistré comme nouvelle image dans Glance.

Gestion des images

OpenStack Glance – Gestion des images

5. Commandes principales (CLI)

- Lister les images disponibles

```
openstack image list
```

- Créer une nouvelle image

```
openstack image create "Ubuntu-22.04" \  
  --file ubuntu-22.04.qcow2 \  
  --disk-format qcow2 \  
  --container-format bare \  
  --public
```

- Montrer les détails d'une image

```
openstack image show Ubuntu-22.04
```

- Supprimer une image

```
openstack image delete Ubuntu-22.04
```


Gestion des images

OpenStack Glance – Gestion des images

6. Cas d'usage

- **Catalogue d'OS validés** pour les projets (ex : Ubuntu, CentOS, Windows Server).
- **Snapshots utilisateurs** pour sauvegarder ou cloner une VM.
- **Golden images** préinstallées avec logiciels d'entreprise (middleware, DB, outils de dev).
- **CI/CD** : préparation d'images customisées pour automatiser les déploiements.

Gestion des images

OpenStack Glance – Gestion des images

En résumé

- **Glance = brique de gestion des images dans OpenStack.**
- Stocke, catalogue et distribue les images pour Nova, Cinder, Swift.
- Supporte plusieurs **formats** et **backends** (Swift, Ceph, FS, Cinder).
- Gère **images OS, snapshots et golden images.**

Gestion des images

Mise en œuvre et configuration de Glance

2. Création de l'utilisateur et du service dans Keystone

Toujours sur le **Controller** :

```
# Créer l'utilisateur glance
openstack user create --domain default --password GLANCE_PASS glance

# Lui donner les droits admin sur le projet "service"
openstack role add --project service --user glance admin

# Créer le service Glance
openstack service create --name glance --description "OpenStack Image" image

# Créer les endpoints (public, internal, admin)
openstack endpoint create --region RegionOne image public http://controller:9292
openstack endpoint create --region RegionOne image internal http://controller:9292
openstack endpoint create --region RegionOne image admin http://controller:9292
```

Gestion des images

Mise en œuvre et configuration de Glance

3. ♦ Installation des paquets

Sur le **Controller node** :

```
apt install glance
```

Gestion des images

Mise en œuvre et configuration de Glance

4. ♦ Configuration de Glance

Éditer `/etc/glance/glance-api.conf` :

```
[database]
connection = mysql+pymysql://glance:GLANCE_PASS@controller/glance

[keystone_authtoken]
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = glance
password = GLANCE_PASS

[paste_deploy]
flavor = keystone

[glance_store]
stores = file,http
default_store = file
filesystem_store_datadir = /var/lib/glance/images/
```

Gestion des images

Mise en œuvre et configuration de Glance

5. Synchronisation de la base de données

```
su -s /bin/sh -c "glance-manage db_sync" glance
```

6. Démarrage et activation des services

```
systemctl restart glance-api  
systemctl enable glance-api
```

Gestion des images

Mise en œuvre et configuration de Glance

7. ♦ Vérification

- **Lister les images** (doit être vide au début) :

```
openstack image list
```

- **Ajouter une image (ex : Cirros, petite image de test) :**

```
wget http://download.cirros-cloud.net/0.5.2/cirros-0.5.2-x86_64-disk.img
```

```
openstack image create "cirros" \  
  --file cirros-0.5.2-x86_64-disk.img \  
  --disk-format qcow2 \  
  --container-format bare \  
  --public
```

- **Vérifier :**

```
openstack image list
```

Gestion des images

Mise en œuvre et configuration de Glance

En résumé

1. **Créer la DB MySQL/MariaDB** pour Glance.
2. **Créer l'utilisateur/service** dans Keystone + endpoints.
3. **Installer Glance** et configurer `glance-api.conf`.
4. **Choisir un backend de stockage** (filesystem, Swift, Ceph, etc.).
5. **Synchroniser la DB** et démarrer le service.
6. **Vérifier avec une image de test** (Cirros).

Gestion des images

Gestion du stockage des images dans OpenStack (Glance)

1. Où sont stockées les images ?

Glance ne stocke pas toujours les images lui-même : il sert de **catalogue** et délègue le stockage à des backends appelés **Glance Stores**.

Backends supportés :

- **File (Filesystem local)** → simple, utilisé pour labs/tests
→ `/var/lib/glance/images/` sur le controller.
- **Swift (Object Storage)** → scalable, HA, production.
- **Ceph RBD** → backend recommandé en production, intégré avec Nova et Cinder.
- **Cinder** → stockage bloc (moins utilisé comme backend d'images).
- **HTTP/HTTPS** → images disponibles via une URL distante.

👉 Le choix dépend du **niveau de production** et du **volume attendu**.

Gestion des images

Gestion du stockage des images dans OpenStack (Glance)

2. ♦ Gestion dans Glance

- **Import** d'une image : via CLI/API, Glance stocke l'image dans son backend et enregistre ses métadonnées.
- **Catalogage** : chaque image possède des métadonnées (OS type, architecture, min RAM/CPU, propriétaire, visibilité publique/privée).
- **Distribution** : quand Nova lance une instance, il demande l'image à Glance → l'hyperviseur la récupère du backend.
- **Snapshots** : une VM existante peut être transformée en image Glance pour réutilisation.

Exemple :

```
openstack image create "Ubuntu-22.04" \  
  --file ubuntu-22.04.qcow2 \  
  --disk-format qcow2 \  
  --container-format bare \  
  --public
```

Gestion des images

Gestion du stockage des images dans OpenStack (Glance)

3 - Gestion des images EC2 (AMI)

1. Rappel : qu'est-ce qu'une AMI ?

- Dans AWS EC2, une **AMI (Amazon Machine Image)** est l'équivalent d'une **image Glance** :
 - Un modèle d'OS (ex. Amazon Linux, Ubuntu, Windows).
 - Contient des métadonnées : type d'archi, volume root, permissions.
- Permet de lancer des **instances EC2**.

Gestion des images

Gestion du stockage des images dans OpenStack (Glance)

3 - Gestion des images EC2 (AMI)

2. Différences avec Glance

- **Glance (OpenStack)** → supporte des formats génériques : QCOW2, RAW, VMDK, ISO.
- **EC2 (AWS)** → AMI est liée à EBS (Elastic Block Store) ou à un snapshot S3.
- **Visibilité :**
 - AMI peut être privée, partagée à un compte, ou publique.
 - Glance fait pareil : `--public`, `--private`, `--shared`.

Gestion des images

Gestion du stockage des images dans OpenStack

3 - Gestion des images EC2 (AMI)

3. ♦ Compatibilité OpenStack ↔ EC2

Historiquement, OpenStack proposait une **EC2 API** (deprecated), permettant :

- D'importer/exporter des AMI.
- D'offrir une compatibilité partielle avec des outils AWS.

Aujourd'hui :

- On peut **convertir une AMI en QCOW2 ou RAW** pour l'utiliser avec Glance.
- Outils utilisés : `qemu-img`

```
qemu-img convert -f vmdk -O qcow2 my-ec2-ami.vmdk my-openstack-image.qcow2
```

- Puis importer dans Glance :

```
openstack image create "EC2-imported" \  
  --file my-openstack-image.qcow2 \  
  --disk-format qcow2 \  
  --container-format bare \  
  --
```

Gestion des images

Gestion du stockage des images dans OpenStack

En résumé

- **Glance** gère les images OpenStack et s'appuie sur des **backends** (**file**, **Swift**, **Ceph**, **Cinder**) pour leur stockage.
- Une **image Glance** = modèle de VM (équivalent d'une **AMI EC2**).
- Les deux assurent :
 - **Import/export d'OS**
 - **Stockage des métadonnées**
 - **Gestion des permissions**
- Différence clé :
 - **OpenStack Glance** → multi-formats, multi-backends.
 - **AWS AMI** → format propriétaire, étroitement lié à EBS/S3.

Gestion du réseau

Gestion du réseau

Vue d'ensemble de la brique Neutron

- **Neutron** est la **brique réseau** d'OpenStack (anciennement Quantum).
- Fournit du **Network-as-a-Service (NaaS)** aux autres services OpenStack.
- Objectif : permettre aux utilisateurs de créer et gérer **leurs propres réseaux virtuels** de manière isolée et multi-tenant.

Fonctionnalités

- Création de **réseaux privés et externes**.
 - Gestion des **sous-réseaux** (subnets, DHCP, DNS).
 - **Routage virtuel** (L3 agent).
 - **Security groups** (pare-feu distribué).
 - **Floating IPs** pour exposer une VM.
 - Support de **VLAN, VXLAN, GRE** pour l'isolation.
 - Extensions : **LBaaS (Octavia), FWaaS, VPNaaS**.
- 👉 **Neutron = le SDN d'OpenStack.**

Gestion du réseau

Switchs virtuels avec Open vSwitch (OVS)

- **Open vSwitch (OVS)** est le switch logiciel utilisé par Neutron (plugin ML2).
- Rôle : connecter les interfaces des VMs, gérer les VLAN/VXLAN, appliquer les règles de sécurité.

Bridges principaux

- **br-int** → switch central, connecte les VM locales.
- **br-tun** → gère les tunnels VXLAN/GRE entre nœuds compute.
- **br-ex** → connecte les réseaux internes vers l'extérieur (Internet, LAN).

Fonctionnement

1. La VM crée une interface virtuelle **tap-xxx** connectée à `br-int`.
2. OVS commute le trafic local ou l'envoie via **VXLAN/GRE** sur `br-tun`.
3. Si destination = Internet → passe par `br-ex`.

👉 OVS rend possible la **connectivité multi-tenant** sans interférence entre projets.

Gestion du réseau

Topologies de réseau Cloud

Flat Network

- Toutes les VMs partagent le même réseau physique.
- Simple, mais pas d'isolation.
- Usage : lab, démo.

VLAN

- Isolation via VLAN tags (802.1Q).
- Limité à **4096 VLANs**.
- Usage : cloud privé classique.

Gestion du réseau

Topologies de réseau Cloud

Overlay (VXLAN/GRE)

- Encapsulation L2 sur L3.
- Jusqu'à **16 millions de réseaux isolés**.
- Usage : clouds publics/multi-tenant (standard OpenStack).

Modèle typique OpenStack

- **Réseau privé tenant** (isolé par VXLAN).
- **Routeur Neutron** pour accéder à un **réseau externe**.
- **Floating IP** pour exposer une VM au public

Gestion du réseau

Daemon de routage (L3 agent)

Rôle

- Le **neutron-l3-agent** fournit :
 - Routage entre subnets.
 - NAT (SNAT/DNAT) pour Internet.
 - Gestion des Floating IP.

Fonctionnement

- Chaque routeur Neutron = **namespace Linux** (`qrouter-xxx`).
- Interfaces internes → réseaux privés.
- Interface externe → réseau provider/public.
- iptables = NAT et règles firewall.

Gestion du réseau

Daemon de routage (L3 agent)

Modes

- **Legacy** : tout le routage sur un **network node central**.
- **DVR (Distributed Virtual Routing)** : routage et NAT sur chaque compute → supprime SPOF.

Gestion du réseau

Mise en œuvre et configuration

a) Installer les paquets

Sur un **network node** :

```
apt install neutron-l3-agent neutron-dhcp-agent neutron-metadata-agent
```

b) Configurer l'agent L3 (`/etc/neutron/l3_agent.ini`)

```
[DEFAULT]  
interface_driver = openvswitch  
external_network_bridge =  
agent_mode = legacy
```

c) Configurer Open vSwitch

```
ovs-vsctl add-br br-ex  
ovs-vsctl add-port br-ex eth1    # eth1 = interface connectée au réseau public
```

Gestion du réseau

Mise en œuvre et configuration

d) Redémarrer les services

```
systemctl restart neutron-l3-agent neutron-dhcp-agent neutron-metadata-agent
```

e) Créer un réseau et un routeur

```
# Réseau privé
openstack network create private-net
openstack subnet create --network private-net --subnet-range 192.168.10.0/24 private-subnet

# Réseau externe (provider)
openstack network create public-net --external --provider-network-type flat --provider-physical-network physnet1
openstack subnet create --network public-net --subnet-range 203.0.113.0/24 --no-dhcp --gateway 203.0.113.1 public-subnet

# Routeur Neutron
openstack router create myrouter
openstack router set myrouter --external-gateway public-net
openstack router add subnet myrouter private-subnet
```

Gestion du réseau

Mise en œuvre et configuration

f) Associer une Floating IP

```
openstack floating ip create public-net  
openstack server add floating ip vm1 FLOATING_IP
```


Gestion du réseau

En résumé

- **Neutron** = service réseau SDN d'OpenStack.
- **OVS** fournit les switchs virtuels (`br-int`, `br-tun`, `br-ex`).
- **Topologies supportées** : flat, VLAN, VXLAN (multi-tenant).
- **L3 agent** gère le routage/NAT/Floating IP.
- **Mise en œuvre** : configuration des agents (L3, DHCP, metadata), OVS, création de réseaux et routeurs.

Authentification et autorisations

Authentification et autorisations

Présentation de la brique Keystone

Rôle

- Keystone est le **service d'identité et d'accès** d'OpenStack.
- Il fournit :
 - **Authentification (AuthN)** : vérifier qui se connecte (utilisateurs, services).
 - **Autorisation (AuthZ)** : vérifier ce qu'ils peuvent faire (via rôles).
 - **Service Catalog** : liste des services OpenStack (Nova, Neutron, Glance, etc.).
 - **Gestion des tokens** : tickets temporaires pour accéder aux APIs.

Authentification et autorisations

Présentation de la brique Keystone

Concepts clés

- **User** : une identité (personne ou service).
 - **Project (tenant)** : conteneur logique de ressources.
 - **Role** : définit les permissions.
 - **Domain** : regroupe projets et utilisateurs.
 - **Token** : jeton d'accès généré par Keystone après login.
- 👉 **Keystone = le SSO (Single Sign-On) d'OpenStack.**

Authentification et autorisations

Création des utilisateurs, projets et rôles

a) Créer un projet

```
openstack project create --domain default --description "Projet de démonstration" demo
```

b) Créer un utilisateur

```
openstack user create --domain default --project demo --password DEMO_PASS demo
```

c) Créer un rôle

```
openstack role create member
```

d) Assigner un rôle à un utilisateur dans un projet

```
openstack role add --project demo --user demo member
```

👉 Résultat : l'utilisateur **demo** a le rôle **member** dans le projet **demo**.

Authentification et autorisations

Mise en œuvre et configuration de Keystone

a) Base de données

```
CREATE DATABASE keystone;  
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' IDENTIFIED BY 'KEYSTONE_DBPASS';  
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' IDENTIFIED BY 'KEYSTONE_DBPASS';  
FLUSH PRIVILEGES;
```

b) Installation

```
apt install keystone apache2 libapache2-mod-wsgi-py3
```

c) Configuration `/etc/keystone/keystone.conf`

```
[database]  
connection = mysql+pymysql://keystone:KEYSTONE_DBPASS@controller/keystone  
  
[token]  
provider = fernet
```

Authentification et autorisations

Mise en œuvre et configuration de Keystone

d) Initialisation

```
su -s /bin/sh -c "keystone-manage db_sync" keystone
keystone-manage fernet_setup --keystone-user keystone --keystone-group keystone
keystone-manage credential_setup --keystone-user keystone --keystone-group keystone

keystone-manage bootstrap --bootstrap-password ADMIN_PASS \
--bootstrap-admin-url http://controller:5000/v3/ \
--bootstrap-internal-url http://controller:5000/v3/ \
--bootstrap-public-url http://controller:5000/v3/ \
--bootstrap-region-id RegionOne
```

e) Configuration Apache

```
echo "ServerName controller" >> /etc/apache2/apache2.conf
systemctl restart apache2
```

Authentification et autorisations

Mise en œuvre et configuration de Keystone

f) Variables d'environnement (admin)

Créer `admin-openrc` :

```
export OS_USERNAME=admin
export OS_PASSWORD=ADMIN_PASS
export OS_PROJECT_NAME=admin
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_DOMAIN_NAME=Default
export OS_AUTH_URL=http://controller:5000/v3
export OS_IDENTITY_API_VERSION=3
```

Charger :

```
source admin-openrc
```


Authentification et autorisations

Configuration des utilisateurs, projets et rôles

a) Projet service (pour les services OpenStack)

```
openstack project create --domain default --description "Service Project" service
```

b) Projet demo + utilisateur demo

```
openstack project create --domain default --description "Demo Project" demo  
openstack user create --domain default --project demo --password DEMO_PASS demo
```

c) Créer et attribuer le rôle member

```
openstack role create member  
openstack role add --project demo --user demo member
```

d) Vérification

```
openstack project list  
openstack user list  
openstack role assignment list --user demo --project demo --names
```

Authentification et autorisations

Configuration des utilisateurs, projets et rôles

e) Fichier openrc pour l'utilisateur demo

Créer `demo-openrc` :

```
export OS_PROJECT_NAME=demo
export OS_USERNAME=demo
export OS_PASSWORD=DEMO_PASS
export OS_AUTH_URL=http://controller:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
```

Authentification et autorisations

En résumé

- **Keystone** = gestion centralisée des identités et autorisations.
- On définit **utilisateurs, projets, rôles** → combinaison = droits.
- **Mise en œuvre** = DB + installation Keystone + bootstrap admin.
- **Configuration** = création de projets (admin, service, demo), utilisateurs et rôles.
- Accès final via fichiers **openrc** (variables d'environnement).

Administration du Cloud

Administration du Cloud

Vue d'ensemble du client Web Horizon

- **Horizon** est le **tableau de bord web officiel d'OpenStack**, basé sur Django (Python).
- Permet aux **utilisateurs** et **administrateurs** de gérer les ressources sans CLI.

Administration du Cloud

Vue d'ensemble du client Web Horizon

Fonctions principales :

- **Pour les utilisateurs :**
 - Créer, démarrer, arrêter, supprimer des **instances Nova**.
 - Gérer les **volumes (Cinder)** et **images (Glance)**.
 - Créer des **réseaux privés (Neutron)**, configurer routeurs, floating IP, security groups.
 - Définir des **clés SSH** et **politiques de sécurité**.
- **Pour les administrateurs :**
 - Gérer les **projets, utilisateurs, rôles** (Keystone).
 - Définir les **quotas** (nombre de VM, IPs, volumes par projet).
 - Superviser l'état des services.

☞ Horizon s'appuie sur les **API REST** de Keystone, Nova, Neutron, Cinder, Glance, etc.

Administration du Cloud

Automatisation avec l'API REST.

OpenStack est **API-first** : toutes les actions passent par des appels REST.
Les clients CLI, SDK et Horizon utilisent ces APIs.

1. Authentification via Keystone → obtenir un token.

```
curl -X POST http://controller:5000/v3/auth/tokens \  
-H "Content-Type: application/json" \  
-d '{  
  "auth": {  
    "identity": {  
      "methods": ["password"],  
      "password": {  
        "user": {  
          "name": "admin",  
          "domain": {"id": "default"},  
          "password": "ADMIN_PASS"  
        }  
      }  
    },  
    "scope": {  
      "project": {  
        "name": "admin",  
        "domain": {"id": "default"}  
      }  
    }  
  }  
}
```

Administration du Cloud

Automatisation avec l'API REST.

2. Consommer un service (ex. Nova, Neutron, Glance) avec ce token :

- Lister les images Glance :

```
curl -H "X-Auth-Token: $TOKEN" http://controller:9292/v2/images
```

- Créer une instance Nova :

```
curl -X POST http://controller:8774/v2.1/servers \  
-H "X-Auth-Token: $TOKEN" -H "Content-Type: application/json" \  
-d '{"server":{"name":"demo","imageRef":"IMAGE_ID","flavorRef":"FLAVOR_ID"}}'
```

👉 API REST = base pour **Terraform, Ansible, CI/CD**.

Administration du Cloud

Présentation des API Amazon EC2 et S3

EC2 (Elastic Compute Cloud)

- API pour gérer les **machines virtuelles** (instances).
- Fonctions principales :
 - Lancer/arrêter/terminer des instances.
 - Associer des volumes (EBS).
 - Gérer des **sécurités (security groups)** et **Elastic IPs**.
- **Équivalent OpenStack : Nova (Compute) + Neutron (réseau) + Cinder (stockage bloc).**

S3 (Simple Storage Service)

- API orientée **stockage objet**.
- Fonctions principales :
 - Créer des **buckets**.
 - Upload/download d'objets.
 - Gestion des ACL et politiques d'accès.
- **Équivalent OpenStack : Swift (Object Storage).**

Administration du Cloud

Automatisation avec Cloud-init

- **Cloud-init** est un agent présent dans les images cloud (Ubuntu, CentOS, Windows via Cloudbase-init).
- Il permet de **personnaliser une VM au premier boot** en lisant des métadonnées fournies par Nova.

Fonctions :

- Configurer le **hostname, utilisateurs, clés SSH**.
- Installer des **packages** (Apache, MySQL, etc.).
- Déployer des fichiers, lancer des scripts.
- Déclencher des outils de config (Ansible, Puppet, Chef).

Administration du Cloud

Automatisation avec Cloud-init

Exemple de script `user-data` :

```
#cloud-config
hostname: web01
users:
  - name: devuser
    ssh-authorized-keys:
      - ssh-rsa AAAAB3Nz...xyz
packages:
  - apache2
  - mysql-client
write_files:
  - path: /var/www/html/index.html
    content: |
      <h1>Bienvenue sur mon serveur web 🚀</h1>
runcmd:
  - systemctl enable apache2
  - systemctl start apache2
```

Administration du Cloud

Automatisation avec Cloud-init

Déploiement via OpenStack CLI :

```
openstack server create \  
  --flavor m1.small \  
  --image Ubuntu-22.04 \  
  --nic net-id=private-net \  
  --user-data init.yaml \  
web01
```

👉 Au premier démarrage, Cloud-init applique la config → VM prête automatiquement.

Administration du Cloud

Automatisation avec Cloud-init

Synthèse

- **Horizon** = interface web pour administrer le cloud.
- **API REST** = cœur d'OpenStack, base de toute automatisation (Terraform, Ansible, CI/CD).
- **API Amazon (EC2 & S3)** = standards cloud → Nova/Neutron/Cinder et Swift sont les équivalents OpenStack.
- **Cloud-init** = outil d'automatisation dans l'instance → personnalisation au boot (utilisateurs, packages, services).