



# OpenStack

Glenn Y. Rolland  
<[teaching@glenux.net](mailto:teaching@glenux.net)>



# ■ Préambule

# Objectifs de la formation

- Comprendre les concepts clés et les bases techniques d'un Cloud privé
- Appréhender OpenStack et ses différentes composantes
- Concevoir un Cloud privé avec OpenStack
- Maîtriser les méthodes et bonnes pratiques de déploiement d'un Cloud privé
- Savoir administrer un Cloud privé

# ■ Qui êtes vous ?

## **Petit tour de présentation... avec 3 questions**

- Quel est votre "bagage" ? (expérience, compétences, etc.)
- Pourquoi participez vous à cette formation aujourd'hui ?
- Comment utiliserez-vous ces nouvelles compétences d'ici 2 ou 5 ans ?



# Qui suis-je ?

**2021 →  
aujourd'hui**

**Auteur, conférencier et co-fondateur de CRYPTO-CHEMISTS**  
Formation et conseil sur l'impact des Blockchain & des technologies P2P.

**2018 →  
aujourd'hui**

**Directeur technique et co-fondateur de BOLDCODE**  
Développement et audit logiciel, web et mobile, offshoring éthique au Népal.

**2017 → 2022**

**Directeur technique et co-fondateur de DATA-TRANSITION**  
Gestion éthique des données, audit des SI, conformité au RGPD.

**2010 → 2017**

**Gérant et co-fondateur de NETCAT (GNUSIDE)**  
Infrastructures & systèmes en réseau, optimisation de la fiabilité, de la sécurité et de la performance.

**2006 → 2010**

**Ingénieur de recherche chez BEWAN (Pace Group)**  
Conception de systèmes embarqués, et automatisation de la qualité logicielle.

# Déroulement de la formation

## Horaires

- 9h00 - 12h30
- 13h30 - 17h00
- Des pauses le matin et l'après midi

## Le cadre

- Liberté de parole dans le respect des autres et des objectifs de la formation
- Bienveillance, nous sommes dans un espace d'apprentissage
- Confidentialité de l'animateur et des participants sur les échanges



# ■ Introduction

# ■ Cloud computing

## Définition du Cloud Computing

- Mise à disposition de ressources informatiques via Internet
- Infrastructures, plateformes et logiciels en tant que services
- Service mesurable (et facturable)
- Paiement à l'utilisation et élasticité des ressources
- Elasticité



## Intérêt et avantages du cloud

- Réduction des coûts d'investissement et d'exploitation
- Accès à des ressources évolutives
- Flexibilité et rapidité de déploiement
- Facilité de collaboration et de partage de données
- Mises à jour et maintenance simplifiées

## Inconvénients et risques liés au Cloud computing

- Dépendance à l'accès Internet
- Latence et performances réseau
- Risques liés à la sécurité et à la confidentialité des données
- Conformité et réglementations
- Perte de contrôle sur l'infrastructure

## Types de Clouds

### **SaaS (Software as a Service)**

- Applications hébergées et gérées par un fournisseur
- Accès via Internet, généralement via un navigateur Web
- Mises à jour et maintenance prises en charge par le fournisseur

### **PaaS (Platform as a Service)**

- Plateformes de développement et d'exécution d'applications
- Services d'infrastructure, de middleware et de développement
- Permet aux développeurs de se concentrer sur le code

### **IaaS (Infrastructure as a Service)**

- Ressources informatiques virtualisées (serveurs, stockage, réseau)
- Fournit des ressources évolutives et à la demande
- Contrôle et gestion de l'infrastructure par l'utilisateur

## Clouds publics et privés

- Cloud public : ressources partagées et gérées par un fournisseur externe
- Cloud privé : ressources dédiées et gérées en interne ou par un fournisseur
- Contrôle, sécurité et performances différenciés



## Clouds hybrides et communautaires

- Cloud hybride : combinaison de clouds publics et privés
- Cloud communautaire : partage de ressources entre organisations similaires
- Flexibilité, partage des coûts et respect des réglementations spécifiques

## Responsabilités selon les modèles

### **Opérateurs "leaders" actuels du clouds**

- Amazon Web Services (AWS) : leader du marché
- Microsoft Azure : offre étendue de services
- Google Cloud Platform (GCP) : puissance de l'IA et du Big Data
- IBM Cloud : solutions hybrides et multicloud
- Oracle Cloud : optimisé pour les bases de données Oracle
- Alibaba Cloud : leader en Chine et Asie

### **Enjeux et besoin pour un cloud opensource**

- Exploitation de ressources matérielle pré-existantes.
- Indépendance vis-à-vis des fournisseurs et réduction de la dépendance
- Personnalisation et adaptation aux besoins spécifiques de l'organisation
- Coûts réduits grâce à l'utilisation de logiciels libres
- Collaboration et partage des connaissances au sein de la communauté
- Amélioration continue grâce aux contributions de la communauté
- Transparence et sécurité renforcées par l'accès au code source
- Interopérabilité et portabilité entre différentes solutions opensource
- Enjeux de souveraineté

## Le projet OpenStack



## ■ Historique et objectifs du projet

- Créé en 2010 par Rackspace Hosting et la NASA
- Objectif : créer une plateforme de Cloud Computing open source
- Favoriser l'interopérabilité et l'évolutivité
- Offrir une alternative aux solutions propriétaires

## Les acteurs clés et contributeurs

- Fondation OpenStack : organisation à but non lucratif qui gère le projet
- Plus de 750 entreprises membres, dont :
  - Red Hat, IBM, HP, Intel, Cisco, Canonical (Ubuntu), Mirantis, SUSE, etc.
- Communauté mondiale de développeurs et d'opérateurs

 [OpenDev: OpenStack](#)

 [GitHub: OpenStack \(mirror\)](#)

## ■ Organisation et gouvernance du projet

- Gouvernance ouverte et transparente
- Comités techniques et groupes de travail
  - Assure une répartition équilibrée des responsabilités
  - Encourage la collaboration et l'échange d'idées
- Contributions ouvertes à tous les membres de la communauté
  - Favorise l'innovation et l'amélioration continue du projet OpenStack
  - Processus de prise de décision basé sur le consensus
- Méritocratie : les contributeurs actifs et compétents sont reconnus et récompensés

 [OpenStack Documentation: OpenStack Governance](#)

## ■ Organisation et gouvernance du projet (2)

### **Board of Directors:**

- Composé de représentants des entreprises membres
- Responsable de la direction stratégique et financière du projet
- Favorise la collaboration entre les membres et l'évolution d'OpenStack

### **Technical Committee (TC)**

- Composé d'experts techniques élus par la communauté
- Garantit la qualité technique et la cohérence entre les projets
- Encourage l'innovation et les meilleures pratiques

### **Project Teams**

- Équipes dédiées à des projets spécifiques au sein d'OpenStack
- Chaque équipe est dirigée par un Project Team Lead (PTL)
- Favorise la répartition des responsabilités et l'expertise spécialisée

### **Special Interest Groups (SIGs):**

- Groupes de travail axés sur des domaines spécifiques ou des problématiques transversales
- Facilitent la collaboration entre les membres ayant des intérêts communs
- Permettent de partager les connaissances et d'améliorer l'efficacité du projet



## Les différentes versions d'OpenStack

- Nouvelle version tous les 6 mois
- Chaque version porte un nom de code alphabétique
  - Exemples : Austin (première version), Bexar, Cactus, Diablo, ...
- Dernière version en date :
  - 2023.1 Antelope : publiée
  - 2023.2 Bobcat : en développement
- Chaque version apporte des améliorations, des corrections de bugs et de nouvelles fonctionnalités
- Certaines versions peuvent marquer des étapes importantes ou introduire des changements majeurs

 [OpenStack Releases](#)

 [2022-02-10 Release Cadence Adjustment](#)


## ■ Comparaison avec les solutions propriétaires

### Amazon Web Services (AWS)

- Propriétaire et fermée
  - Exploitation des projets opensource
- Très large gamme de services

### Microsoft Azure

- Propriétaire et fermée
- Intégration avec les produits Microsoft

 [OpenStack vs Azure Stack: 5 Key Differences](#)

## ■ Comparaison avec les solutions propriétaires

### **Bilan général**

- Moins d'options "clés en main" pour les services managés
- Un écosystème de partenaires moins étendu
- Certaines fonctionnalités avancées sont absentes ou moins développées (ex: machine learning, data lakes, outils devops intégrés, IoT et services connectés, CDN mondiaux)
- Intégration avec des outils propriétaires peut être plus complexe
- Support technique généralement moins centralisé et structuré

 [CompareCloud](#)

## ■ Comparaison avec les solutions open source

### CloudStack

- Projet créé en 2008 par Cloud.com acheté par Citrix en 2011
- Don en open-source à la fondation Apache, en 2012
- Moins de composants et de fonctionnalités que OpenStack
- Architecture plus simple et plus facile à déployer

 <https://cloudstack.apache.org/>

### Eucalyptus

- Projet open source créé en 2008
- Perte de vitesse depuis 2014 (rachat par HP)
- Compatible avec les API Amazon EC2 et S3
- Moins de fonctionnalités et d'évolutivité que OpenStack

 <https://www.eucalyptus.cloud/>

## ■ Comparaison avec les solutions open source (2)

### OpenNebula


- Projet open source centré sur la gestion des centres de données virtuels
- Moins de fonctionnalités et de composants que OpenStack
- Plus adapté aux petites et moyennes infrastructures
- Utilisé par le StratusLab, Telefonica et Unity Technologies

 <https://opennebula.io/>

# ■ Architecture de la plateforme OpenStack

## Vue d'ensemble des composants

- OpenStack est modulaire et flexible
- Composants appelés "services" ou "projets"
- Chaque service responsable d'une fonction spécifique

 [OpenStack Documentation:](#)  
[OpenStack API documentation](#)

## Interaction entre les composants

- Services communiquent via API
- Messages transmis par bus de messages (ex : RabbitMQ)
- Standardisation des interactions entre services



## Présentation des 6 principales briques

 core services, center

## Nova (Compute)

- Gestion des instances de machines virtuelles
- Supporte divers hyperviseurs (KVM, Hyper-V, ESXi)
- Interactions avec d'autres services (Neutron, Cinder, Glance)

 Nova System Architecture

## Cinder (Block Storage)

- Gestion des volumes de stockage en mode bloc
- Peut être connecté à des instances Nova
- Divers backends supportés (Ceph, LVM, NetApp, etc.)

 [OpenStack Documentation: Cinder, Basic architecture](#)

 [Laying Cinder Block \(Volumes\) In OpenStack](#)

## Neutron (Networking)

- Gestion des réseaux et des adresses IP
- Supporte divers plugins (Open vSwitch, Linux Bridge)
- Fonctionnalités avancées (LBaaS, VPNaaS, FWaaS)

## Glance (Image Service)

- Gestion des images de machines virtuelles
- Stockage et récupération d'images
- Supporte divers formats (QCOW2, VMDK, etc.)


 [OpenStack Documentation: Glance, Basic architecture](#)

## Swift (Object Storage)

- Stockage d'objets distribué et évolutif
- Haute disponibilité et réplication des données
- Supporte API S3 d'Amazon

## Keystone (Identity Service)

- Gestion de l'authentification et des autorisations
- Gestion des utilisateurs, rôles et projets
- Supporte divers backends d'authentification (LDAP, OAuth, etc.)

 OpenStack Keystone  
Workflow & Token Scoping

## Horizon (Dashboard)

- Interface web de gestion d'OpenStack
- Permet de gérer et surveiller les ressources
- Extensible via plugins



## Composants optionnels et extensions

 components, center

## Compute projets

- **Nova**
- **Glance**
- **Ironic** (Bare Metal) : service de gestion des machines physiques, permettant de gérer des machines physiques comme des ressources de calcul dans un environnement OpenStack.
- **Magnum** : provisionning et management de moteurs d'orchestration
- **Zun** (Container service) : service de déploiement et de gestion de conteneurs, permettant de créer et de gérer des conteneurs dans un environnement OpenStack.
- **Storlet** (FaaS) : implémente les Fonctions-as-a-service dans le contexte d'Object Storage.

## ■ Sécurité, Identité et Conformité

- **KeyStone**
- **Barbican** : provisioning dsécurisé, management des secrets (password, certificats X.509, clés de chiffrement, etc.)
- **Congress** - service de gouvernance
- **Mistral** (Workflow service) : service de gestion des flux de travail, permettant de créer et de gérer des flux de travail pour automatiser les tâches dans un environnement OpenStack.

## Stockage, Sauvegarde et Recovery

- **Cinder**
- **Swift**
- **Manila** (Shared File System) : service de partage de fichiers, permettant de créer et de gérer des systèmes de fichiers partagés dans un environnement OpenStack.
- **Kardor** : backup et restauration des data et meta-data d'applications
- **Freezer** : backup et restauration de fichiers et de filesystemes

- **Neutron**
- **Designate** (DNSaaS) : service de gestion des noms de domaine, permettant de créer et de gérer des enregistrements DNS dans un environnement OpenStack.
- **Dragonflow** : controller de SDN distribué
- **Kuryr** : plugin réseau Docker
- **Octavia** (Load Balancer) : service de répartition de charge, permettant de créer et de gérer des équilibreurs de charge dans un environnement OpenStack.
- **Tacker** (NFV Orchestration) : service de gestion des fonctions réseau virtuelles (NFV), permettant de créer et de gérer des fonctions réseau telles que des routeurs, des pare-feu et des passerelles VPN.
- **Tricile** : extension des abstraction réseau à travers plusieurs instances openstack

## Data et Analyse

- **Trove** : DBaaS
- **Sahara** (Data processing) : service de traitement de données, permettant de créer et de gérer des clusters de traitement de données dans un environnement OpenStack.

## Projects applicatifs

- **Heat** (Orchestration) : Gestion de l'infrastructure en tant que code (IaaS), automatisation du déploiement de ressources. Supporte des modèles de déploiement (HOT, AWS CloudFormation)
- **Murano** (Application Catalog) : catalogue d'applications permettant de déployer et de gérer des applications dans un environnement OpenStack.

## Et aussi ...

- **Cyborg** (Accelerator management) : service de gestion des accélérateurs matériels tels que les GPU, permettant de les utiliser pour accélérer les charges de travail dans un environnement OpenStack.
- **Masakari** (High Availability) : service de haute disponibilité, permettant de surveiller les nœuds de l'infrastructure OpenStack et de les récupérer en cas de panne.
- **Monasca** (Monitoring) : service de surveillance des performances de l'infrastructure OpenStack, permettant de collecter et d'analyser des données de performances pour optimiser les ressources et résoudre les problèmes.
- **Rally** (Benchmarking) : outil de benchmarking pour tester les performances d'OpenStack, permettant de simuler des charges de travail et de mesurer les performances de l'infrastructure.



- **Senlin** (Cluster management) : service de gestion de clusters, permettant de créer et de gérer des clusters de machines virtuelles ou physiques dans un environnement OpenStack.
- **Watcher** (Infrastructure optimization) : service d'optimisation de l'infrastructure, permettant d'optimiser les ressources de l'infrastructure OpenStack en fonction des charges de travail et des objectifs métier.

# Méthodes d'installation

## Installation manuelle

- Procédure étape par étape
- Grande flexibilité et contrôle
- Grande quantité de composants - haute complexité
- Temps d'installation très long
- Risque d'erreurs plus élevé
- Idéal pour apprendre et maîtriser OpenStack

## Utilisation de scripts et d'outils d'automatisation

- Gains de temps et de fiabilité
- Réduction des erreurs humaines
- Installation plus rapide
- Nécessite une connaissance préalable des outils

## Packstack

- Outil basé sur Puppet
- Installation rapide d'OpenStack
- Idéal pour les environnements de test et de développement

 [OpenStack Wiki: Packstack](#)

## DevStack

- Outil de développement et de test
- Configuration rapide d'un environnement OpenStack
- Facile à mettre à jour
- Ne convient pas pour les déploiements en production

## Ansible

- Outil d'automatisation et de gestion de configuration
- Playbooks pour déployer OpenStack
- Flexible et extensible
- Convient pour les déploiements en production

## Kolla Ansible

- Déploiement d'OpenStack avec Docker
- Simplifie la gestion et la mise à jour
- Réduction de la complexité de maintenance
- Solution résiliente et évolutive



### Points communs

- Versions pré-configurées d'OpenStack
- Support commercial et maintenance
- Mises à jour et correctifs réguliers
- Coût plus élevé que les installations manuelles

## Red Hat OpenStack Platform

- Basée sur la distribution CentOS
- Support de Red Hat
- Intégration avec d'autres produits Red Hat
- Formation et certification disponibles

## Mirantis Cloud Platform

- Support Kubernetes et Docker
- Gestion des mises à jour en continu
- Formation et services professionnels

## SUSE OpenStack Cloud

- Basée sur la distribution SUSE Linux
- Intégration avec d'autres produits SUSE
- Support et services professionnels

## Ubuntu OpenStack

- Basée sur la distribution Ubuntu
- Intégration avec d'autres produits Canonical
- Support et services professionnels

## Critères de sélection d'une méthode d'installation

- Complexité de l'infrastructure
- Budget et ressources disponibles
- Niveau d'expertise technique
- Besoins en termes de support et de maintenance
- Objectifs à long terme et évolutivité

# ■ Machines virtuelles

# Introduction à Nova



## Qu'est-ce que Nova ?

- Nova est le service de calcul d'OpenStack.
- Nova gère les instances de machines virtuelles (VM) et les ressources de calcul.
- Nova est conçu pour être évolutif et distribué.
- Nova interagit avec d'autres composants d'OpenStack (par exemple, Keystone, Glance, Neutron).

## Fonctionnalités et composants de Nova

- Gestion des instances de VM (création, modification, suppression).
- Planification des instances sur les hôtes.
- Gestion des ressources de calcul (CPU, RAM, stockage).
- API RESTful pour l'interaction avec d'autres services et outils.
- Support de plusieurs hyperviseurs (KVM, Xen, VMware, Hyper-V, etc.).
- Composants principaux :
  - nova-api : reçoit et traite les requêtes des utilisateurs.
  - nova-scheduler : sélectionne l'hôte approprié pour les instances.
  - nova-conductor : interagit avec la base de données et les autres services.
  - nova-compute : gère les instances et les ressources de calcul sur les hôtes.
  - nova-db : stocke les informations sur les instances, les hôtes, les réseaux et les volumes. Permet de conserver l'état des ressources et des configurations.

## Gestion des ressources et interactions avec les autres services OpenStack

- Nova est le gestionnaire principal des ressources de calcul dans OpenStack.
- Nova orchestre les cycles de vie des instances, en contrôlant leur création, leur mise à l'échelle, leur suspension, leur redémarrage, leur migration et leur suppression.
- Nova interagit avec les autres services d'OpenStack pour fournir des fonctionnalités intégrées :
  - Keystone pour l'authentification des utilisateurs et la définition des autorisations.
  - Glance pour la récupération des images de VM à partir desquelles les instances sont créées.
  - Neutron pour la configuration des réseaux virtuels et l'attribution des adresses IP aux instances.
  - Cinder pour la fourniture de volumes de stockage en mode bloc aux instances.
  - Swift pour le stockage d'objets, y compris les instantanés d'instance et les images de disque.
  - etc.

## Fonctionnalités avancées et extensibilité de Nova

- La gestion des quotas pour les projets et les utilisateurs.
- La planification des instances en fonction des politiques et des ressources disponibles.
- L'équilibrage de charge entre les hôtes de calcul.
- La haute disponibilité et la récupération après sinistre grâce à la migration en direct et aux instantanés d'instance.
- Nova permet l'intégration avec des outils d'automatisation et d'orchestration, tels que Heat, Terraform et Ansible, via son API RESTful.
- Nova offre également une extensibilité grâce à des plugins et des pilotes pour divers hyperviseurs, matériels et services tiers.

# Gestion des images

## Utilisation des images avec Nova

- Importer des images dans le catalogue d'images Glance
- Choisir une image pour créer une instance
- Lister les images disponibles
  - CLI: `openstack image list`
  - Horizon: *Compute > Images*
- Utiliser des formats d'image pris en charge : qcow2, raw, vmdk, vhd, etc.

# Gestion des gabarits

- Understanding the concept of flavors and how they are used in Nova
- Creating and managing flavors
- Using the Nova command-line client to manage flavors

# ■ Gestion du réseau virtuel



## Comprendre les réseaux virtuels dans OpenStack

- Réseaux virtuels : abstraction des réseaux physiques pour les instances
- Réseau Flat : pas de VLAN, toutes les instances partagent le même réseau
- Réseau VLAN : chaque projet a son propre VLAN, isolation des réseaux
- Réseau VXLAN/GRE : encapsulation des paquets, évolutivité accrue
- Utilisation de Neutron en combinaison avec Nova pour la gestion du réseau

## Configuration du réseau virtuel avec Nova

- Configuration de Nova pour utiliser Neutron ( `neutron.conf` )
- Définition du type de réseau (Flat, VLAN, VXLAN/GRE) dans `nova.conf`
- Configuration du pilote de réseau virtuel (Open vSwitch, Linux Bridge) dans `nova.conf`
- Configuration de l'adresse du serveur Neutron ( `neutron.url` )
- Création et configuration de réseaux virtuels dans Horizon ou via la CLI `openstack network create`
- Association des réseaux virtuels aux instances lors de leur création

## Gestion des adresses IP et des sous-réseaux

- Utilisation des pools d'adresses IP (Floating IP) pour les instances
- Création et configuration des sous-réseaux dans Horizon ou via la CLI `openstack subnet create`
- Attribution des adresses IP aux instances lors de leur création
- Modification des adresses IP et des sous-réseaux associés aux instances en cours d'exécution
- Gestion des adresses IP publiques et privées pour les instances
- Utilisation de la CLI `nova floating-ip-associate` pour associer des adresses IP flottantes aux instances

# Gestion des instances

## Création d'instances

- Sélectionner une image, un type de machine virtuelle (flavor) et d'autres paramètres (réseau, clé SSH, etc.)
- Créer une instance avec openstack server create ou dans Horizon sous "Compute" > "Instances" > "Launch Instance"
- Spécifier des métadonnées pour personnaliser l'instance
- Attacher des volumes à l'instance pour étendre le stockage
- Configurer la sécurité et les groupes de sécurité pour contrôler l'accès à l'instance

## Manipulation des instances

- Arrêter une instance
  - CLI: `openstack server stop <instance_id>`
  - Horizon: *Compute > Instances > Instance Actions" > Stop*
- Redémarrer une instance avec
  - CLI: `openstack server reboot <instance_id>`
  - Horizon: *Compute > Instances > Instance Actions > Reboot*
- Suspendre et reprendre une instance
  - CLI: `openstack server suspend <instance_id>` et `openstack server resume <instance_id>`
  - Horizon: *Compute > Instances > Instance Actions > Suspend / Resume*
- Redimensionner une instance (changer le flavor) avec
  - CLI: `openstack server resize <instance_id> <new_flavor>`
  - Horizon: *Compute > Instances > Instance Actions > Resize*
- Supprimer une instance avec
  - CLI: `openstack server delete <instance_id>`
  - Horizon: *Compute > Instances > Instance Actions > Delete*

## ■ Mise en œuvre et configuration de Nova

## Installation de Nova

- Installer les paquets requis :
  - openstack-nova-api
  - openstack-nova-conductor
  - openstack-nova-consoleauth
  - openstack-nova-novncproxy
  - openstack-nova-scheduler
  - openstack-nova-placement-api
- Activer et démarrer les services :

```
# systemctl enable --now nova-api.service
# systemctl enable --now nova-scheduler.service
# systemctl enable --now nova-conductor.service
# systemctl enable --now nova-consoleauth.service
# systemctl enable --now nova-novncproxy.service
# systemctl enable --now nova-placement-api.service
```



## Configuration initiale de Nova

- Modifier le fichier de configuration `/etc/nova/nova.conf` :
  - Spécifier les informations de connexion à la base de données (section `[database]`)
  - Définir les informations d'authentification avec Keystone (section `[keystone_authtoken]`)
  - Configurer l'API placement (section `[placement]`)
  - Configurer l'accès aux images avec Glance (section `[glance]`)
- Appliquer la configuration et redémarrer les services :

```
# systemctl restart nova-api.service
# systemctl restart nova-scheduler.service
# systemctl restart nova-conductor.service
# systemctl restart nova-consoleauth.service
# systemctl restart nova-novncproxy.service
# systemctl restart nova-placement-api.service
```

- Synchroniser la base de données :

## Personnalisation et optimisation de la configuration

- Configurer l'hyperviseur (section [libvirt]) :
  - Choisir l'hyperviseur (KVM, QEMU, ESXi, etc.)
  - Configurer les paramètres spécifiques à l'hyperviseur
- Optimiser les performances (section [DEFAULT]) :
  - Configurer la taille des pools de travailleurs
  - Personnaliser les paramètres de mise en cache
- Configurer la gestion des réseaux (section [neutron]) :
  - Spécifier les informations d'authentification avec Neutron
  - Configurer les paramètres du réseau
- Gérer les quotas (section [quota]) :
  - Configurer les quotas par défaut pour les instances, les cœurs, la RAM, etc.

## Gestion d'hyperviseurs multiples

## ■ Introduction aux hyperviseurs supportés

## Configuration d'ESXi avec Nova

- Installer et configurer VMware vSphere et ESXi
- Configurer Nova pour utiliser ESXi comme hyperviseur
  - Modifier le fichier `/etc/nova/nova.conf`
  - Changer l'option `"compute_driver"` à `"vmwareapi"`
  - Renseigner les informations d'authentification et de connexion pour vSphere
- Redémarrer les services Nova après la configuration
- Vérifier la configuration via la CLI nova ou la CLI openstack

## Configuration de KVM avec Nova

- Installer et configurer KVM et libvirt sur les nœuds de calcul
- Configurer Nova pour utiliser KVM comme hyperviseur
  - Modifier le fichier `/etc/nova/nova.conf`
  - Changer l'option `"compute_driver"` à `"libvirt"`
  - Renseigner le type d'hyperviseur à `"kvm"`
- Redémarrer les services Nova après la configuration
- Vérifier la configuration via la CLI nova ou la CLI openstack

## Gestion des ressources entre différents hyperviseurs

- Comprendre l'allocation des ressources dans OpenStack
- Utiliser les filtres d'hôte pour contrôler la répartition des instances
  - Modifier le fichier `/etc/nova/nova.conf`
  - Activer et configurer les filtres d'hôte appropriés Utiliser les agrégats d'hôtes pour regrouper les hôtes par hyperviseur
  - Créer des agrégats via Horizon, la CLI nova ou la CLI openstack
  - Ajouter des hôtes aux agrégats et définir des métadonnées
- Surveiller l'utilisation des ressources par hyperviseur via Horizon, la CLI nova ou la CLI openstack



## Stockage bloc



# Présentation de Cinder

## Introduction à Cinder

- Cinder : service de stockage en mode bloc d'OpenStack
- Fournit des volumes persistants aux instances de machines virtuelles
- Fonctionne avec divers backends de stockage
- Peut être utilisé avec des hyperviseurs compatibles, tels que KVM, ESXi et Hyper-V

## Architecture et composants

- Cinder-API : expose les API pour les opérations de stockage en mode bloc
- Cinder-Scheduler : sélectionne le backend de stockage approprié en fonction des critères de filtrage et de pondération
- Cinder-Volume : gère les opérations de volume, telles que la création, la suppression et l'attachement
- Les drivers de volume : assurent la communication avec les backends de stockage spécifiques
- Les bases de données : stockent les informations sur les volumes et les backends de stockage

## Utilisation et cas d'usage

- Création de volumes persistants pour les instances de machines virtuelles
- Sauvegarde et restauration de volumes
- Snapshots de volumes pour créer des instantanés de données
- Migration de volumes entre différents backends de stockage
- Gestion des quotas de stockage pour les projets et les utilisateurs

## Manipulation de Cinder

Pour les opérations avec Cinder, on peut utiliser :

- Horizon : l'interface web d'OpenStack, qui permet de gérer les volumes dans la section "Volumes" sous "Computing"
- CLI nova (obsolete) : l'interface en ligne de commande d'OpenStack, qui permet de gérer les volumes avec des commandes comme :
  - `nova volume-create`,
  - `nova volume-delete`
  - `nova volume-attach`
- CLI openstack :
  - `openstack volume create`
  - `openstack volume delete`
  - `openstack volume set`

## ■ Mise en œuvre du stockage en mode bloc avec Cinder

## Installation de Cinder

- Installer les paquets nécessaires : cinder-api, cinder-scheduler, cinder-volume
- Configurer la base de données pour Cinder
- Synchroniser la base de données avec la commande : `cinder-manage db sync`

## Configuration des services Cinder

- Modifier le fichier de configuration `/etc/cinder/cinder.conf`
  - Configurer les informations d'authentification et les points d'accès pour les autres services d'OpenStack
  - Configurer les informations du backend de stockage (ex: LVM, Ceph, NFS)
  - Configurer les options spécifiques aux pilotes de stockage (si nécessaire)
- Redémarrer les services de Cinder après la configuration
  - `systemctl restart cinder-api`
  - `systemctl restart cinder-scheduler`
  - `systemctl restart cinder-volume`



## ■ Backend supportés par Cinder

## Les backends de stockage

- Types de backends de stockage pris en charge :
  - Stockage en mode bloc (SAN)
  - Stockage en mode fichier (NAS)
  - Stockage objet
- Exemples de backends pris en charge :
  - Ceph RBD, LVM, NFS, GlusterFS,
  - Dell EMC, HPE 3PAR, IBM, NetApp, Pure Storage, VMware VMDK

## Configuration des backends

- Configuration des backends dans le fichier cinder.conf :
  - Définir plusieurs sections de backends
  - Spécifier les paramètres spécifiques à chaque backend

 [OpenStack Documentation: Cinder Configuration](#)

## Configuration des backends

- Exemple de configuration pour le backend Ceph RBD :
  - `rbd_user`, `rbd_pool`, `rbd_secret_uuid`, `rbd_ceph_conf`
- Exemple de configuration pour le backend LVM :
  - `volume_driver`, `volume_group`, `lvm_type`
- Exemple de configuration pour le backend NFS :
  - `nfs_shares_config`, `nfs_mount_options`
- Utilisation de la CLI:
  - `openstack volume service list`

## Intégration avec les systèmes de stockage existants

- Adapter la configuration de Cinder pour utiliser un système de stockage existant :
  - Utiliser les paramètres spécifiques au backend
  - Adapter les paramètres de connexion (authentification, adresses IP, etc.)
- Migration de volumes entre backends :
  - Utiliser la commande `openstack volume migrate` pour déplacer un volume d'un backend à un autre
  - Migrer les volumes sans interruption de service
- Vérifier l'intégration dans Horizon :
  - Créer un volume avec le nouveau backend sélectionné
  - Attacher le volume à une instance
  - Vérifier le bon fonctionnement du volume avec le nouveau backend

## ■ Stockage objet

# ■ Présentation de Swift

## Introduction à Swift

- Swift est un système de stockage d'objets distribué et hautement disponible
- Fait partie intégrante du projet OpenStack
- Conçu pour stocker et récupérer de grandes quantités de données non structurées
- *Scaling* horizontal pour supporter des petaoctets de données
- Réplication automatique des données pour assurer la durabilité et la disponibilité



## Utilisation et cas d'usage

- Stockage d'objets volumineux comme des images, vidéos, sauvegardes, archives, etc.
- Idéal pour les applications nécessitant une grande capacité de stockage et une disponibilité élevée
- Utilisé en conjonction avec d'autres services OpenStack comme Glance pour stocker les images de machines virtuelles
  - Gestion des conteneurs et objets dans le panneau *Project > Object Store > Containers*

## Utilisation et cas d'usage

- Intégration avec la CLI OpenStack pour la gestion des conteneurs et des objets
  - Création et suppression des conteneurs :
    - `openstack container create <container_name>`
    - `openstack container delete <container_name>`
  - Listing des conteneurs et des objets avec
    - `openstack container list`
    - `openstack object list <container_name>`
  - Téléchargement et téléversement des objets avec
    - `openstack object save <container_name> <object_name>`
    - `openstack object create <container_name> <object_name>`
- Visualisation et gestion des conteneurs et objets via l'interface Web Horizon

## Architecture et composants

- Composants principaux : serveurs Proxy, serveurs de stockage et anneau (Ring)
- Serveurs Proxy : gèrent les requêtes des clients et interagissent avec les serveurs de stockage
- Serveurs de stockage : responsables du stockage et de la récupération des objets
- Anneau (Ring) : structure de données logique qui répartit les objets sur les serveurs de stockage
- Réplication et rééquilibrage automatiques pour assurer la cohérence des données

## Fonctionnement du Ring Builder dans Swift

- Création des rings
  - Trois types de rings : account, container, object
  - Chaque ring gère une partie spécifique du stockage
- Partitionnement
  - Division de l'espace de stockage en partitions égales
  - Les partitions sont des unités de base pour la distribution des données
- Répartition des partitions
  - Attribution des partitions aux nœuds de stockage selon capacité et poids
  - Équilibrage des réplicas sur différents nœuds et zones de stockage

## Fonctionnement du Ring Builder dans Swift (2)

- Génération du fichier de configuration du ring
  - Fichier de configuration pour chaque type de ring (ring.gz)
  - Distribution des fichiers aux nœuds du cluster Swift
  - Utilisation pour déterminer l'emplacement des objets et réplicas
- Mise à jour des rings
  - Utilisation du ring builder pour mettre à jour les rings en cas de changements
  - Redistribution et rééquilibrage des modifications sur le cluster
- Commandes swift-ring-builder
  - Utilisées pour créer, gérer et mettre à jour les rings
  - Planification et configuration importantes pour optimiser performance et disponibilité

## ■ Mise en œuvre et configuration de Swift

## Installation de Swift

- Préparer l'environnement système
  - Installer les paquets requis
  - Configurer les dépôts logiciels appropriés
- Installer les services Swift
  - Proxy Server
  - Account Server
  - Container Server
  - Object Server
- Configuration des services Swift
  - Configurer le fichier [/etc/swift/proxy-server.conf](#)
  - Configurer le fichier [/etc/swift/account-server.conf](#)
  - Configurer le fichier [/etc/swift/container-server.conf](#)
  - Configurer le fichier [/etc/swift/object-server.conf](#)

## Configuration des services Swift

- Configurer le Proxy Server
  - Définir les options d'authentification
  - Spécifier le pipeline WSGI
  - Configurer les paramètres du réseau
- Configurer les services Account, Container et Object
  - Définir les paramètres du réseau
  - Configurer les chemins de stockage
  - Spécifier les politiques de réplication et les paramètres de temps de réconciliation
- Configurer les services complémentaires
  - Activer le service de dispersion (pour vérifier la répartition des données)
  - Configurer les quotas de conteneurs et d'objets



## Paramétrage des politiques de stockage

- Créer des politiques de stockage
  - Éditer le fichier /etc/swift/swift.conf
  - Définir les politiques de stockage avec les noms et les indices uniques
- Appliquer les politiques de stockage
  - Ajouter les politiques aux fichiers de configuration des services (proxy-server.conf, account-server.conf, container-server.conf, object-server.conf)
  - Redémarrer les services Swift pour appliquer les modifications
    - `systemctl restart swift-proxy`
    - `swift-init restart all`

## Paramétrage des politiques de stockage (2)

- Gérer les politiques de stockage
  - Dans la configuration, au niveau des serveurs swift
  - Pas dans Horizon
- Utiliser les politiques de stockage via la CLI
  - Utiliser les commandes openstack pour créer, lister et supprimer des conteneurs avec différentes politiques de stockage
    - `openstack container create --storage-policy <policy_name> <container_name>`
    - `openstack container list --long`
  - Utiliser les commandes openstack pour afficher les détails et les statistiques des politiques de stockage
    - `swift stat -v`

## ■ Gestion des pools de stockage avec Swift

## Création et gestion des pools

- Création de pools de stockage avec la CLI Swift
  - Utiliser swift-ring-builder pour créer un anneau (ring)
  - Ajouter des devices avec swift-ring-builder
- Configuration des zones de stockage
  - Définir les zones pour une répartition optimale des données
- Modification et suppression des pools
  - Utiliser swift-ring-builder pour modifier ou supprimer un device

## Répartition des données

- Fonctionnement de la répartition des données dans Swift
  - Utilisation d'algorithmes de répartition (consistent hashing)
  - Réplication des données pour garantir la durabilité
- Contrôle de la répartition des données
  - Configurer le nombre de réplicas
  - Choisir le niveau de durabilité souhaité
- Optimisation de la répartition des données
  - Ajuster les poids des devices dans l'anneau (ring)

## Supervision et maintenance des pools

- Surveillance des pools de stockage
  - Utiliser les outils de monitoring (par exemple: Swift Healthcheck)
  - Vérifier l'état des services Swift avec `systemctl` ou `service`
- Maintenance des pools de stockage
  - Remplacement d'un device défaillant
  - Mise à jour de la configuration de l'anneau (ring) avec `swift-ring-builder`

 [OpenStack Documentation: Swift Ring builder](#)



## Gestion du réseau

# ■ Introduction à Neutron



## ■ Historique et évolution

- Projet Quantum lancé en 2011 pour gérer le réseau dans OpenStack
- Renommé en Neutron en 2013 pour éviter les conflits de marque
- Evolution continue pour répondre aux besoins changeants du cloud computing
- Intégration avec d'autres technologies de virtualisation réseau (SDN, NFV)

## Fonctionnalités principales

- Gestion des réseaux virtuels
- Gestion des sous-réseaux et des adresses IP
- Gestion des routeurs virtuels et de la connectivité L3
- Support pour les réseaux VLAN, VXLAN et GRE
- Intégration avec Open vSwitch pour les switchs virtuels
- Extension de fonctionnalités via des plugins et des drivers
- API RESTful pour l'automatisation et l'intégration avec d'autres outils

## Composants de Neutron

- neutron-server : service central pour gérer les requêtes API
- neutron-plugin : plugin pour le backend spécifique (ex: Open vSwitch, Linux Bridge)
- neutron-dhcp-agent : agent pour gérer les services DHCP
- neutron-l3-agent : agent pour gérer les routeurs virtuels et la connectivité L3
- neutron-metadata-agent : agent pour fournir les métadonnées aux instances

## Utilisation

Informations dans Horizon :

- Onglet "Réseaux" pour gérer les réseaux, sous-réseaux, et routeurs
- Onglet "Instances" pour connecter les instances aux réseaux

Commandes CLI :

- Utiliser `openstack network` et `openstack subnet` pour gérer les réseaux et sous-réseaux
- Utiliser `openstack router` pour gérer les routeurs virtuels
- Utiliser `nova boot` avec l'option `--nic` pour connecter une instance à un réseau spécifique

# ■ Switchs virtuels avec Open vSwitch

## Présentation d'Open vSwitch

- Open vSwitch (OVS) : switch virtuel multilayer open source
- Fonctionne sur Linux et d'autres systèmes d'exploitation
- Conçu pour supporter les réseaux Cloud
- Compatible avec les normes de gestion de réseaux SDN (Software-Defined Networking)
- Gestion avancée du trafic réseau entre instances virtuelles et le réseau physique

## Installation et configuration

- Installer Open vSwitch sur les noeuds réseau :
  - `apt-get install openvswitch-switch` (Debian/Ubuntu)
  - `yum install openvswitch` (CentOS/RHEL)
- Créer un bridge OVS :
  - `ovs-vsctl add-br br-int`
- Ajouter des interfaces au bridge :
  - `ovs-vsctl add-port br-int eth1`
- Configurer les VLANs si nécessaire :
  - `ovs-vsctl set port eth1 tag=VLAN_ID`

## Intégration avec Neutron

- Configurer Neutron pour utiliser OVS comme mécanisme de gestion des réseaux virtuels
  - Modifier le fichier `/etc/neutron/plugins/ml2/ml2_conf.ini`
  - Changer le type de mécanisme à `mechanism_drivers = openvswitch`
- Redémarrer les services Neutron pour prendre en compte les modifications
  - `systemctl restart neutron-server`
  - `systemctl restart neutron-openvswitch-agent`
- Vérifier l'intégration d'OVS avec Neutron
  - Utiliser Horizon : aller dans le tableau de bord administrateur, puis Réseaux > Agents
  - Utiliser la CLI openstack : `openstack network agent list`
  - Rechercher l'agent Open vSwitch dans les résultats



# ■ Topologies de réseau Cloud

## Réseau plat (Flat Network)

- Pas de segmentation, toutes les machines virtuelles sur le même réseau physique
- Simplicité de mise en œuvre et de gestion
- Limite la scalabilité et augmente les risques de sécurité
- Configuration dans Neutron : choix du mécanisme de type "flat"
- Visualisation dans Horizon : onglet "Réseaux", section "Réseaux"

## Réseau VLAN (Virtual LAN)

- Segmentation du réseau physique en plusieurs réseaux logiques
- Améliore la sécurité et l'isolation entre les machines virtuelles
- Utilise des tags 802.1Q pour identifier les réseaux logiques
- Configuration dans Neutron : choix du mécanisme de type "vlan"
- Visualisation dans Horizon : onglet "Réseaux", section "Réseaux"
- CLI : `openstack network create --provider-network-type vlan`

## Réseau VXLAN (Virtual Extensible LAN)

- Encapsulation de trames Ethernet dans des paquets IP
- Permet de créer des réseaux overlay indépendants de l'infrastructure physique
- Améliore la scalabilité par rapport au VLAN, jusqu'à 16 millions d'identifiants de réseau
- Configuration dans Neutron : choix du mécanisme de type "vxlan"
- Visualisation dans Horizon : onglet "Réseaux", section "Réseaux"
- CLI : `openstack network create --provider-network-type vxlan`

## Réseau GRE (Generic Routing Encapsulation)

- Encapsulation de paquets de divers protocoles dans des paquets IP
- Permet de créer des réseaux overlay indépendants de l'infrastructure physique
- Moins performant que VXLAN en raison de l'absence d'optimisation matérielle
- Configuration dans Neutron : choix du mécanisme de type "gre"
- Visualisation dans Horizon : onglet "Réseaux", section "Réseaux"
- CLI : `openstack network create --provider-network-type gre`

## ■ Daemon de routage (L3)

## Principe du routage L3

- Routage L3 : interconnexion de réseaux IP à différents niveaux hiérarchiques
- Fonctions principales :
  - Transmission de paquets entre sous-réseaux
  - Gestion de la table de routage
  - Implémentation de politiques de routage
- Utilisation de routeurs virtuels pour le routage dans OpenStack
- Gestion des adresses IP publiques et privées
- Support de NAT (Network Address Translation) pour les instances

## Mise en œuvre avec Neutron

- Neutron fournit le service L3 pour le routage et la gestion des routeurs virtuels
- Création de routeurs virtuels dans OpenStack via Neutron
- Association des routeurs virtuels avec des réseaux externes et internes
- Configuration des interfaces de routeur pour les sous-réseaux
- Utilisation des commandes CLI neutron et openstack pour gérer les routeurs
- Accès aux routeurs et gestion du routage via l'interface Horizon



## Configuration du routage

- Création d'un routeur virtuel :
  - Horizon : Réseau > Routeurs > Créer un routeur
  - CLI : `openstack router create ROUTER_NAME`
- Association d'un routeur à un réseau externe :
  - Horizon : Réseau > Routeurs > Définir le réseau externe
  - CLI : `neutron router-gateway-set ROUTER_ID EXTERNAL_NETWORK_ID`
- Ajout d'une interface de routeur pour un sous-réseau :
  - Horizon : Réseau > Routeurs > Ajouter une interface
  - CLI : `openstack router add subnet ROUTER_ID SUBNET_ID`
- Configuration des règles de routage et NAT :
  - Horizon : Réseau > Routeurs > Gérer les règles
  - CLI : Utiliser `neutron security-group-rule-create` pour créer des règles

## Vérification du routage

- Vérification de la configuration du routage :
  - Horizon : *Network > Topology*
  - CLI : `openstack router show ROUTER_ID` et `openstack router list`



# Mise en œuvre et configuration de Neutron

## Installation des paquets Neutron

- Installer les paquets nécessaires:
  - apt-get install neutron-server neutron-plugin-ml2 Installer les paquets pour les agents L3 et DHCP:
  - apt-get install neutron-l3-agent neutron-dhcp-agent
- Installer les paquets pour Open vSwitch:
  - apt-get install neutron-plugin-openvswitch-agent
- Installer les paquets pour le contrôleur de réseau:
  - apt-get install openvswitch-controller

## Configuration des fichiers de Neutron

- Configurer le fichier `/etc/neutron/neutron.conf` :
  - Paramétrer la connexion à la base de données
  - Configurer l'authentification avec Keystone
- Configurer le fichier `/etc/neutron/plugins/ml2/ml2_conf.ini` :
  - Paramétrer les mécanismes de pilotes (ex: openvswitch, linuxbridge)
  - Configurer les types de réseau (ex: flat, vlan, vxlan)
  - Configurer les gammes d'adresses pour les réseaux VXLAN
- Configurer le fichier `/etc/neutron/l3_agent.ini` :
  - Paramétrer l'interface externe pour le routage L3
- Configurer le fichier `/etc/neutron/dhcp_agent.ini` :
  - Activer l'agent DHCP
  - Configurer l'interface pour l'agent DHCP
- Configurer le fichier `/etc/neutron/plugin.ini` :
  - Spécifier le fichier de configuration ML2

## Démarrage des services Neutron

- Redémarrer les services après la configuration:
  - `systemctl restart neutron-server`
  - `systemctl restart neutron-plugin-openvswitch-agent`
  - `systemctl restart neutron-l3-agent`
  - `systemctl restart neutron-dhcp-agent`
- Vérifier l'état des services Neutron:
  - `systemctl status neutron-server`
  - `systemctl status neutron-plugin-openvswitch-agent`
  - `systemctl status neutron-l3-agent`
  - `systemctl status neutron-dhcp-agent`

## Trouver les informations sur le réseau

- Trouver les informations dans Horizon:
  - Aller dans le tableau de bord "Réseau"
  - Vérifier les configurations de réseau et les agents
- Utiliser la CLI nova:
  - nova network-list
  - nova floating-ip-list
- Utiliser la CLI openstack:
  - openstack network list
  - openstack subnet list
  - openstack router list



# ■ Authentification et autorisations



## Présentation de la brique Keystone

## ■ Introduction à Keystone

- Keystone : service d'authentification et d'autorisation d'OpenStack
- Gère les identités et les accès aux ressources d'OpenStack
- Fournit un point central pour la gestion des utilisateurs, projets et services
- Utilise des jetons pour authentifier les requêtes

## Fonctionnalités principales de Keystone

- Authentification : vérifie les identifiants des utilisateurs
  - Utilisation de la CLI OpenStack ou de l'interface Horizon pour s'authentifier
- Autorisation : détermine les permissions d'accès aux ressources
  - Basé sur les rôles attribués aux utilisateurs
  - Possibilité de définir des politiques de sécurité personnalisées
- Gestion des services : enregistre et découvre les services d'OpenStack
  - Catalogue des services disponibles dans l'infrastructure
  - Accessible via Horizon ou la CLI OpenStack
- Gestion des jetons : émet et valide les jetons d'authentification
  - Jetons temporaires utilisés pour authentifier les requêtes API

## Composants de Keystone

- API : interface pour interagir avec le service Keystone
  - Supporte les requêtes HTTP et les réponses au format JSON
  - Utilisation de la CLI OpenStack ou de l'interface Horizon pour accéder aux fonctionnalités
- Endpoint : point d'accès aux services d'OpenStack
  - URL spécifique pour chaque service et région
  - Permet d'interagir avec les autres services d'OpenStack
- Backend : stockage des données de Keystone
  - Utilisation de bases de données relationnelles comme MySQL ou PostgreSQL
  - Peut également utiliser des solutions de stockage LDAP pour la gestion des utilisateurs et des groupes
- Middleware : composant logiciel intermédiaire pour valider les jetons
  - Intégré aux autres services d'OpenStack pour sécuriser l'accès aux API
  - Utilise les jetons pour vérifier les permissions et authentifier les requêtes

## ■ Création des utilisateurs, projets et rôles

## Utilisateurs dans Keystone

- Objectif des utilisateurs : représenter des individus ou des systèmes dans OpenStack
- Authentification : identifiants (login/mot de passe) ou autres méthodes (tokens)
- Association aux domaines et projets pour déterminer les droits d'accès

### Utilisation

- Horizon : *Identity > Users*
- Commandes CLI :
  - `openstack user create`
  - `openstack user list`
  - `openstack user delete`

## Projets et domaines

- Projets : unités organisationnelles pour regrouper et isoler les ressources
- Domaines : conteneurs pour gérer plusieurs projets et utilisateurs
- Multi-tenancy : isolation des ressources et des politiques entre différents projets
- Horizon : *Identity > Projects et Domains*
- Commandes CLI :
  - `openstack project create`
  - `openstack domain create`

## Rôles et assignations de rôles

- Rôles : ensemble de droits et permissions pour les utilisateurs
- Assignation de rôles : lien entre utilisateurs, projets/domaines et rôles
- Rôles courants : admin, member, reader
- Commandes CLI :
  - `openstack role create`
  - `openstack role assignment create`
- Interface Horizon : *Identity > Roles et Role Assignments*



## Processus de création d'utilisateurs, projets et rôles

- Étape 1 : créer un domaine (optionnel, si multi-domaines requis)
- Étape 2 : créer un projet
- Étape 3 : créer un utilisateur et l'associer à un projet
- Étape 4 : créer un rôle
- Étape 5 : assigner un rôle à un utilisateur pour un projet/domaine spécifique
- Utilisation des commandes CLI et de l'interface Horizon pour chaque étape

## ■ Configuration des utilisateurs, projets et rôles

## Gestion des utilisateurs

- Création d'utilisateurs avec la CLI openstack: `openstack user create`
- Modification d'utilisateurs: `openstack user set`
- Suppression d'utilisateurs: `openstack user delete`
- Listage des utilisateurs: `openstack user list`
- Assignment d'un rôle à un utilisateur: `openstack role add`
- Horizon: *Identity > Users*

## Gestion des projets et domaines

- Création de projets: `openstack project create`
- Modification de projets: `openstack project set`
- Suppression de projets: `openstack project delete`
- Listage des projets: `openstack project list`
- Création de domaines: `openstack domain create`
- Modification de domaines: `openstack domain set`
- Suppression de domaines: `openstack domain delete`
- Horizon: *Identity > Projects et Domains*

## Gestion des rôles et assignations

- Création de rôles: `openstack role create`
- Modification de rôles: `openstack role set`
- Suppression de rôles: `openstack role delete`
- Listage des rôles: `openstack role list`
- Assignment de rôles aux utilisateurs: `openstack role add --user --project`
- Retrait de rôles aux utilisateurs: `openstack role remove --user --project`
- Horizon: *Identity > Roles*

## Utilisation des politiques de sécurité

- Définition des politiques de sécurité dans le fichier `policy.json`
- Contrôle d'accès basé sur les rôles (RBAC)
- Configuration de politiques pour les services OpenStack
- Horizon: *Identity > Policies*

```
{  
  "identity:create_user": "role:admin"  
}
```

 [OpenStack Documentation: The policy.json file](#)

## Mise en œuvre et configuration



## Installation de Keystone

- Installer les paquets nécessaires
  - apt-get install keystone
- Vérifier les dépendances (ex: Python)
- Consulter la documentation officielle pour les autres systèmes d'exploitation



## Configuration des fichiers de Keystone

- Modifier le fichier de configuration principal
  - `/etc/keystone/keystone.conf`
- Configurer la connexion à la base de données
  - `[database]`
  - `connection = mysql+pymysql://keystone:KEYSTONE_DBPASS@controller/keystone`
- Configurer le token Fernet
  - `[token]`
  - `provider = fernet`

 [OpenStack Documentation: KeyStone - Install and configure \(OBS\)](#).  [OpenStack Documentation: KeyStone - Install and configure \(Ubuntu\)](#).

## Initialisation de la base de données Keystone

- Créer la base de données MySQL
  - `CREATE DATABASE keystone;`
- Accorder les privilèges nécessaires
  - `GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' IDENTIFIED BY 'KEYSTONE_DBPASS';`
  - `GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' IDENTIFIED BY 'KEYSTONE_DBPASS';`
- Initialiser la base de données
  - `keystone-manage db_sync`

## Démarrage du service Keystone

- Démarrer et activer le service Keystone
  - `systemctl enable keystone`
  - `systemctl start keystone`
- Vérifier le statut du service
  - `systemctl status keystone`
- Configurer les variables d'environnement
  - `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`
- Utiliser la CLI OpenStack pour vérifier la configuration
  - `openstack token issue`



## Gestion des orchestration et Infrastructure As Code (Heat)

# ■ Infrastructure as Code, Heat and Terraform

## Définition de l'Infrastructure as Code (IaC)

- Infrastructure as Code (IaC) : approche pour gérer et provisionner les ressources informatiques
- Utilisation de fichiers de configuration textuels pour décrire l'état souhaité de l'infrastructure
- Mise en place de l'infrastructure automatisée avec des outils et des scripts

## Avantages de l'IaC

- Rapidité de déploiement
  - Provisionnement automatisé des ressources
  - Réduction du temps consacré à la configuration manuelle
  - Accélération du processus de déploiement des infrastructures
- Répétabilité
  - Uniformisation des environnements de développement, de test et de production
  - Réduction des erreurs humaines lors du provisionnement
  - Facilité de mise à l'échelle

## Avantages de l'IaC (2)

- Traçabilité
  - Suivi des modifications de l'infrastructure grâce au versionnement du code
  - Meilleure collaboration entre les équipes
  - Audit facilité des changements d'infrastructure
- Simplification de la gestion des environnements
  - Centralisation des configurations et des paramètres
  - Définition claire des responsabilités
  - Réutilisation des configurations pour différents projets



## Les principes de base de l'IaC

- Codification de l'infrastructure
  - Description de l'infrastructure avec un langage déclaratif
  - Utilisation de templates pour décrire les ressources et leurs relations
- Versionnement du code
  - Utilisation de systèmes de gestion de version (ex. Git)
  - Suivi des modifications et collaboration entre les équipes

## Les principes de base de l'IaC

- Tests et validation
  - Vérification de la syntaxe et des erreurs dans les fichiers de configuration
  - Tests automatisés pour valider le comportement de l'infrastructure
  - Utilisation de l'intégration et du déploiement continu (CI/CD)
- Automatisation du déploiement et de la gestion
  - Utilisation d'outils de déploiement (ex. Heat, Terraform)
  - Gestion des modifications d'infrastructure avec des pipelines automatisés
  - Contrôle des accès et des autorisations via des outils comme Keystone et Horizon

# ■ Présentation de Heat et Terraform pour OpenStack

## Introduction à Heat

- Origines et objectifs
  - Projet OpenStack dédié à l'orchestration
  - Automatiser et gérer le déploiement d'infrastructures complexes
- Composants principaux
  - heat-api: interface RESTful pour interagir avec Heat
  - heat-engine: cœur de l'orchestrateur, traite les templates et exécute les actions
  - heat-dashboard: plugin Horizon pour gérer Heat depuis l'interface web
- Fonctionnement de Heat
  - Utilisation de templates YAML (Heat Orchestration Template, HOT)
  - Description des ressources, paramètres et dépendances
  - Déploiement, mise à jour et suppression des stacks

## Introduction à Terraform

- Origines et objectifs
  - Outil IaC multi-cloud développé par HashiCorp
  - Provisionner, modifier et détruire des infrastructures de manière déclarative
- Composants principaux
  - Terraform Core: interprète les fichiers de configuration et gère les ressources
  - Providers: extensions pour interagir avec différentes plateformes (OpenStack, AWS, etc.)
  - Modules: réutilisation de configurations pour faciliter la gestion de l'infrastructure
- Fonctionnement de Terraform
  - Utilisation de fichiers de configuration HCL (HashiCorp Configuration Language)
  - Description des ressources, variables et dépendances
  - Planification et exécution des changements d'infrastructure

## ■ Comparaison entre Heat et Terraform

- Points communs et différences
  - Les deux permettent l'automatisation et la gestion d'infrastructures
  - Heat est spécifique à OpenStack, Terraform est multi-cloud
- Avantages et inconvénients de chaque solution
  - Heat: intégration native avec OpenStack, moins polyvalent que Terraform
  - Terraform: prise en charge de plusieurs plateformes, courbe d'apprentissage plus longue
- Cas d'utilisation typiques
  - Heat: déploiements OpenStack purs, gestion des ressources OpenStack spécifiques
  - Terraform: déploiements multi-cloud, infrastructure hybride

## Intégration de Heat et Terraform avec OpenStack

- Interaction avec les briques OpenStack
  - Heat: intégration native avec Nova, Swift, Glance, Neutron, etc.
  - Terraform: utilisation du provider OpenStack pour interagir avec les briques
- Configuration et authentification
  - Heat: configuration via openrc.sh, authentification avec Keystone
  - Terraform: configuration du provider OpenStack, authentification avec les variables d'environnement
- Exemples d'utilisation pour OpenStack
  - Heat: déploiement de VMs, création de réseaux, ajout de stockage, etc.
  - Terraform: provisionnement de VMs, allocation d'adresses IP, configuration de réseaux, etc.

## Présentation de la brique Heat



## Objectifs et fonctionnalités de Heat

- Automatiser le déploiement et la gestion des ressources OpenStack.
- Gérer l'évolutivité des ressources en fonction des besoins.
- Faciliter l'infrastructure As Code avec des templates.

## Composants clés de Heat

- Heat API : service RESTful permettant d'interagir avec Heat.
- Heat Engine : interprète et exécute les templates HOT.
- Heat CLI : interface en ligne de commande pour interagir avec Heat.
- Dashboard Heat (intégré à Horizon) : interface graphique pour gérer les templates et les stacks.

## ■ Intégration avec les autres briques d'OpenStack

- Interaction avec Keystone pour l'authentification et la gestion des autorisations.
- Gestion des instances via Nova.
- Configuration et gestion des réseaux avec Neutron.
- Gestion du stockage des images avec Glance
- Idem pour les autres ressources (swift, etc.)
- Heat s'intègre dans Horizon pour faciliter la gestion des stacks et des templates.

## ■ Le langage de modélisation HOT (Heat Orchestration Template)

## Introduction au langage HOT

- Heat Orchestration Template (HOT) : format YAML pour décrire les ressources et les configurations
- HOT facilite la gestion et l'automatisation des déploiements d'infrastructure
- Compatible avec les autres services OpenStack
- Utilisation de la CLI Heat pour déployer et gérer les templates

## Structure d'un template HOT

- En-tête avec la version du template et description (optionnelle)
- Section "resources" pour décrire les ressources
- Section "outputs" pour définir les sorties (optionnelle)
- Section "parameters" pour définir les paramètres d'entrée (optionnelle)

## En-tête

- La première ligne spécifie la version du langage à utiliser dans le template.
- La description est optionnelle et fournit des informations sur le but du template.

```
heat_template_version: 2018-03-02  
description: Deploy a simple web server
```

## Section "resources"

- Décrit les ressources
- Liste des ressources à déployer, avec leurs types, noms et propriétés.

```
resources:  
  my_instance:  
    type: OS::Nova::Server  
    properties:  
      image: "Ubuntu 18.04"  
      flavor: "m1.small"
```



## Section "outputs"

- Définit les sorties (optionnelle)
- Les sorties fournissent des informations sur les ressources déployées (ex: adresses IP, URL).

```
outputs:  
  instance_ip:  
    description: The IP address of the deployed instance  
    value: { get_attr: [my_instance, first_address] }
```

## Section "parameters"

- Définit les paramètres d'entrée (optionnelle) :
- Les paramètres permettent de personnaliser le déploiement (ex: choix de l'image, de la taille de l'instance).

```
parameters:  
  image_name:  
    type: string  
    default: "Ubuntu 18.04"  
    description: Name of the image to use for the instance
```

## Exemple complet

```
heat_template_version: 2018-03-02
description: Deploy a simple web server

parameters:
  image_name:
    type: string
    default: "Ubuntu 18.04"
    description: Name of the image to use for the instance

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      image: { get_param: image_name }
      flavor: "m1.small"

outputs:
  instance_ip:
    description: The IP address of the deployed instance
    value: { get_attr: [my_instance, first_address] }
```

## Ressources, propriétés et attributs

- Ressources : éléments d'infrastructure déployés par Heat (ex: instances, réseaux)
- Propriétés : spécifications et configurations des ressources
- Attributs : informations dynamiques sur les ressources, accessibles après déploiement
- Liste complète des ressources et propriétés dans la documentation OpenStack

## Fonctions intrinsèques

- Fonctions pour manipuler et traiter les données du template
- Elles facilitent la gestion des ressources, des attributs et des paramètres dans un template HOT.
- Exemples : `get_attr`, `get_param`, `get_resource`, `ref`
- Permettent d'éviter la duplication de code et de simplifier la maintenance
- Documentation complète des fonctions intrinsèques dans la documentation OpenStack

 [OpenStack Documentation: Fonctions intrinsèques](#)

## get\_attr

- Utilisée pour récupérer la valeur d'un attribut d'une ressource
- Syntaxe : `{ get_attr: [ resource_name, attribute_name ] }`
- Exemple :

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      ...

outputs:
  instance_ip:
    description: The IP address of the instance
    value: { get_attr: [ my_instance, first_address ] }
```

## get\_param

- Utilisée pour récupérer la valeur d'un paramètre d'entrée du template
- Syntaxe : `{ get_param: parameter_name }`
- Exemple :

```
parameters:
  image_id:
    type: string
    description: ID of the image to use for the instance

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      image: { get_param: image_id }
      ...
```

## get\_resource

- Utilisée pour récupérer le nom d'une ressource
- Syntaxe : `{ get_resource: resource_name }`
- Exemple :

```
resources:  
  my_instance:  
    type: OS::Nova::Server  
    properties:  
      ...  
  
  my_volume:  
    type: OS::Cinder::Volume  
    properties:  
      ...  
  
  my_volume_attachment:  
    type: OS::Cinder::VolumeAttachment  
    properties:  
      instance_uuid: { get_resource: my_instance }  
      volume_id: { get_resource: my_volume }
```



## ref

- Utilisée pour référencer une ressource, généralement pour récupérer son ID
- Syntaxe : `{ ref: resource_name }`
- Exemple :

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      ...

  my_security_group:
    type: OS::Neutron::SecurityGroup
    properties:
      ...

  my_instance_security_group:
    type: OS::Nova::ServerSecurityGroup
    properties:
      server: { ref: my_instance }
      security_group: { ref: my_security_group }
```

## Commandes de gestion de stack

- Déploiement d'une stack

```
openstack stack create \  
-t my_stack.yaml my_stack
```

- Voir la stack créée

```
openstack stack show my_stack
```

- Lister les stack

```
openstack stack list
```

- Lister les evenements d'une stack

```
openstack stack event list my_stack
```

- Lister les ressources d'une stack

```
openstack stack resource list my_stack
```

- Supprimer une stack

```
openstack stack delete my_stack
```

## Gestion des instances dans Heat

## Création d'instances via Heat

- Utilisation de la ressource OS::Nova::Server pour définir une instance
- Spécification des propriétés essentielles : `image` , `flavor` , `key_name`

```
heat_template_version: 2018-03-02

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      image: "cirros-0.3.5-x86_64-disk"
      flavor: "m1.tiny"
      key_name: "my_key"
```

 [OpenStack Documentation: Software configuration](#)

## Configuration et personnalisation des instances

- Utilisation de `user_data` pour exécuter des scripts lors du lancement de l'instance
- Utilisation des métadonnées pour définir des valeurs personnalisées
- Passage de paramètres via Heat pour personnaliser l'instance lors de la création

## User data

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      image: "Ubuntu 20.04"
      flavor: "m1.small"
      key_name: "my_keypair"
      user_data: |
        #!/bin/bash
        apt-get update
        apt-get install -y apache2
```

## Cloud init

```
cloud-config
package_upgrade: true
packages:
  - apache2
write_files:
  - path: /var/www/html/index.html
    content: |
      <html>
        <head>
          <title>Welcome to my website!</title>
        </head>
        <body>
          <h1>Hello, World!</h1>
        </body>
      </html>
runcmd:
  - systemctl enable apache2
  - systemctl start apache2
```

## Meta données

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      image: "Ubuntu 20.04"
      flavor: "m1.small"
      key_name: "my_keypair"
      metadata:
        environment: "production"
        app_name: "my_app"
```



## Paramètres

```
parameters:
  instance_name:
    type: string
    description: Name of the instance

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      name: {get_param: instance_name}
      image: "Ubuntu 20.04"
      flavor: "m1.small"
      key_name: "my_keypair"
```

```
openstack stack create -t my_template.yaml --parameter "instance_name=my_custom_instance" my_stack
```

## Gestion des réseaux

## Création et configuration de réseaux virtuels via Heat

- Utilisation de la ressource `OS::Neutron::Net` pour créer un réseau virtuel
- Spécification des propriétés : `name` , `shared` , `tenant_id`

```
resources:  
  my_network:  
    type: OS::Neutron::Net  
    properties:  
      name: "my_network"  
      shared: false
```

```
openstack stack create -t network_template.yaml my_network_stack
```

## Gestion des sous-réseaux et des routeurs

- Utilisation de la ressource `OS::Neutron::Subnet` pour créer un sous-réseau
- Spécification des propriétés : `network_id`, `cidr`, `gateway_ip`, `allocation_pools`
- Utilisation de la ressource `OS::Neutron::Router` pour créer un routeur
- Spécification des propriétés : `external_gateway_info`, `name`

## Gestion des sous-réseaux et des routeurs

```
resources:
  my_net:
    type: OS::Neutron::Net
    properties:
      name: my_net

  my_subnet:
    type: OS::Neutron::Subnet
    properties:
      network_id: { get_resource: my_net }
      cidr: 192.168.1.0/24

  my_router:
    type: OS::Neutron::Router
    properties:
      name: my_router
      external_gateway_info:
        network: { get_param: external_network_id }

  my_router_interface:
    type: OS::Neutron::RouterInterface
    properties:
      router_id: { get_resource: my_router }
      subnet_id: { get_resource: my_subnet }
```

## Gestion des volumes

## Création de volumes via Heat

- Utilisation de la ressource `OS::Cinder::Volume` pour créer un volume
- Spécification des propriétés : `size`, `image`, `name`

```
resources:  
  my_volume:  
    type: OS::Cinder::Volume  
    properties:  
      size: 10  
      name: MyVolume  
      image: <image_id>
```

## Attachement de volumes

- Utilisation de la ressource `OS::Cinder::VolumeAttachment` pour attacher un volume à une instance
- Spécification des propriétés : `instance_uuid`, `volume_id`, `mountpoint`

```
resources:
  my_volume_attachment:
    type: OS::Cinder::VolumeAttachment
    properties:
      instance_uuid: { get_resource: my_instance }
      volume_id: { get_resource: my_volume }
      mountpoint: /dev/vdb
```



## Configuration avancée des volumes

- Utilisation de la ressource `OS::Cinder::Volume` pour configurer un volume
- Spécification des propriétés : `availability_zone`, `metadata`, `volume_type`

```
my_volume:  
  type: OS::Cinder::Volume  
  properties:  
    size: 10  
    availability_zone: nova  
    metadata:  
      role: database  
    volume_type: ssd
```

## Utilisation des snapshots

- Utilisation de la ressource `OS::Cinder::VolumeSnapshot` pour créer un snapshot
- Spécification des propriétés : `volume_id`, `name`, `description`
- Exemple de création et configuration d'un snapshot dans un template HOT

```
my_volume_snapshot:  
  type: OS::Cinder::VolumeSnapshot  
  properties:  
    volume_id: { get_resource: my_volume }  
    name: "Database Volume Snapshot"  
    description: "Snapshot of the database volume"
```



**Merci pour votre  
attention !**