UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Terraform avec OpenStack

Programme

1. Introduction
   1. Objectifs de la formation
   2. Qu’est ce que l’IaC
   3. Les principales solutions d'Infrastructure as Code
   4. Présentation de Terraform
   5. Présentation d'OpenStack
2. Mise en place de l'environnement
   1. Installation de Terraform
   2. Installation du client OpenStack
   3. Configuration des accès API pour OpenStack
3. Concepts de base de Terraform
   1. Les providers
   2. Les ressources
   3. Les variables
   4. Les outputs
   5. Les modules

Programme

4. Provider OpenStack pour Terraform
    1. Configuration du provider OpenStack
    2. Exemples de ressources OpenStack
5. Gestion de l'état de Terraform
    1. Comprendre l'état de Terraform
    2. Utilisation de backends pour stocker l'état
6. Création d'une infrastructure avec Terraform et OpenStack
    1. Création d'un réseau et d'un sous-réseau
    2. Création d'une instance
    3. Création d'un groupe de sécurité
    4. Création d'un volume
    5. Création d'un routeur et connexion au réseau externe
    6. Utilisation des modules pour organiser le code
7. Les meilleures pratiques pour Terraform et OpenStack
    1. Planification des changements
    2. Utilisation de la documentation
    3. Collaboration avec d'autres membres de l'équipe

Programme

8. Approfondissements avec Terraform
    1. Segmentation d’infrastructure pour limiter les périmètres opérationnels.
    2. Organisation des équipes et périmètres de responsabilité

Objectifs de la formation

- Comprendre les concepts clés de Terraform et OpenStack
- Apprendre à créer et gérer une infrastructure cloud avec Terraform et OpenStack
- Maîtriser les meilleures pratiques pour utiliser Terraform avec OpenStack

Qu’est ce que l’IaC

- Infrastructure as Code (IaC) est une approche permettant de gérer et de provisionner des ressources d'infrastructure de manière programmable, en utilisant des fichiers de configuration plutôt que des processus manuels.

- L’IaC permet de traiter l'infrastructure comme du code source, en utilisant des pratiques de développement logiciel telles que la gestion de versions, le contrôle de code source et l'automatisation des tests.

Les principales solutions d'Infrastructure as Code (Open Source)

- **Terraform** : Terraform est un outil open-source populaire pour la création, la modification et la versionnage de l'infrastructure de manière déclarative. Il prend en charge plusieurs fournisseurs de cloud, tels que AWS, Azure, Google Cloud, et permet de décrire l'infrastructure cible dans des fichiers de configuration en langage HCL (HashiCorp Configuration Language).
- **Chef** : Chef est un outil d'automatisation open source qui permet de gérer la configuration et le déploiement d'infrastructures. Il utilise un langage de configuration appelé Chef Infra pour décrire l'état désiré du système. Chef prend en charge une large gamme de systèmes d'exploitation et de fournisseurs de cloud, offrant une grande flexibilité dans la gestion de l'infrastructure.
- **Ansible** : Ansible est un outil d'automatisation open source qui permet la gestion de configuration, le déploiement d'applications et l'orchestration d'infrastructures. Il utilise un langage simple basé sur YAML pour décrire les tâches et les configurations. Ansible peut être utilisé pour provisionner des ressources sur différents environnements, y compris les environnements sur site et cloud.
- **Puppet** : Puppet permet de décrire l'état désiré du système et de gérer la configuration de manière déclarative. Il utilise une syntaxe spécifique appelée "Puppet DSL" (Domain-Specific Language) pour décrire les ressources, les packages, les services, les fichiers de configuration, etc., nécessaires sur un système.
- **SaltStack** : SaltStack est une plateforme open source d'automatisation et de gestion de configuration. Elle utilise un langage appelé SaltStack pour décrire les configurations et les états des systèmes. SaltStack est connu pour sa capacité à gérer des environnements à grande échelle et à exécuter des tâches de manière rapide et efficace.

Les principales solutions d'Infrastructure as Code (Cloud Provider)

- **AWS CloudFormation** : AWS CloudFormation est un service d'Amazon Web Services (AWS) qui permet de déployer et de gérer des ressources AWS à l'aide de fichiers de modèle JSON ou YAML. Il offre une prise en charge complète des services AWS et permet de créer des stacks d'infrastructure de manière déclarative.
- **Azure Resource Manager (ARM)** : Azure Resource Manager est la solution d'Infrastructure as Code de Microsoft Azure. Il permet de déployer et de gérer des ressources Azure en utilisant des fichiers de modèle JSON. ARM fournit une prise en charge complète des services Azure et permet de déployer des ressources dans des groupes de ressources.
- **Google Cloud Deployment Manager** : Google Cloud Deployment Manager est l'outil d'IaC de Google Cloud Platform (GCP). Il permet de déployer et de gérer des ressources GCP à l'aide de fichiers de configuration YAML ou Python. Deployment Manager offre une prise en charge complète des services GCP et permet de créer des déploiements reproductibles.

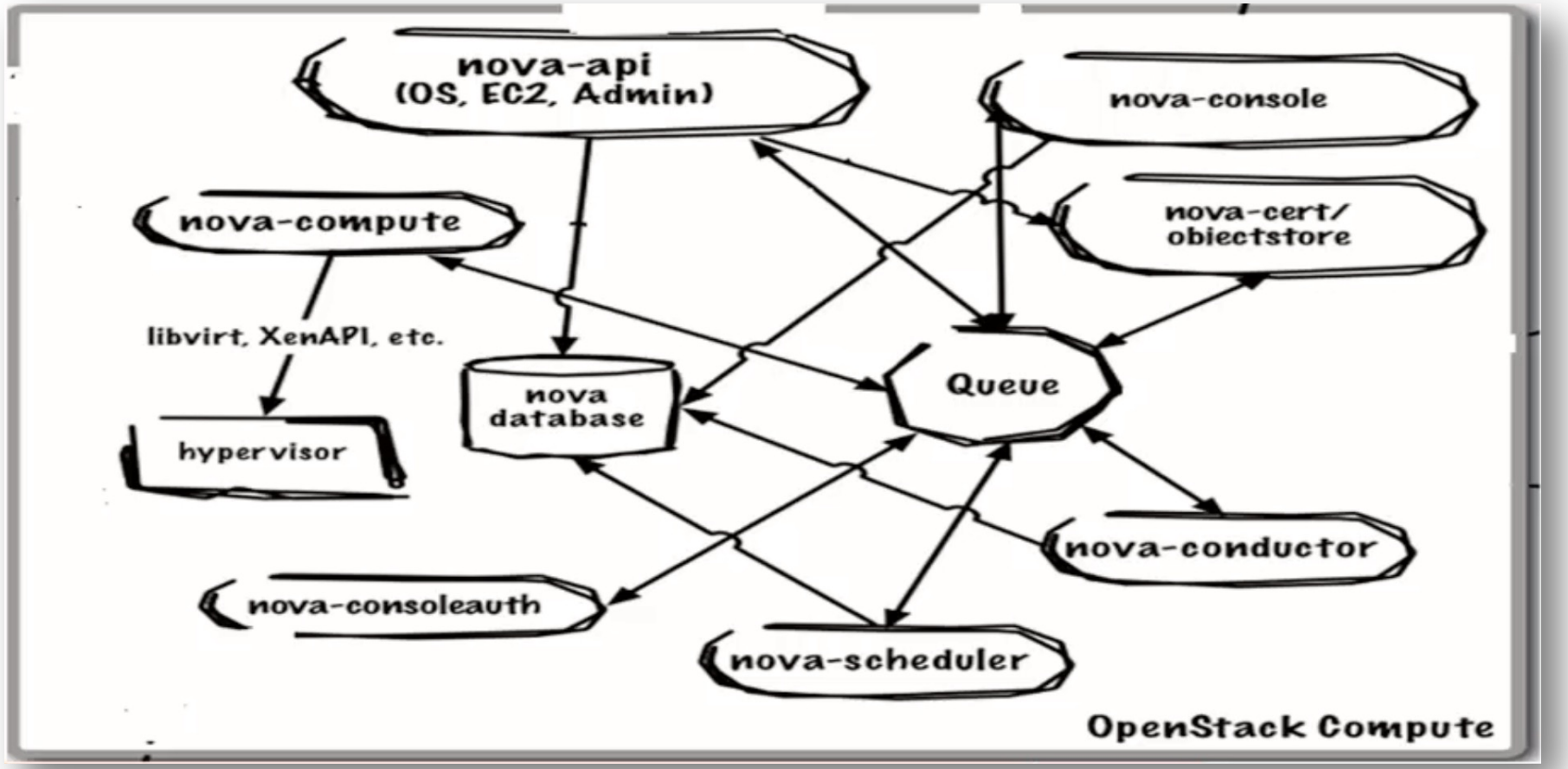
Présentation de Terraform

- **Définition de Terraform** :
  - **Terraform** est un outil open-source développé par HashiCorp qui permet de définir, provisionner et gérer des ressources d'infrastructure en tant que code (IaC) dans divers fournisseurs de cloud, tels qu'OpenStack.
  - **Terraform** utilise son propre langage de configuration déclaratif appelé HCL (HashiCorp Configuration Language) pour décrire les ressources souhaitées, puis génère un plan d'exécution pour atteindre l'état souhaité.
- **Avantages de l'Infrastructure as Code (IaC)** :
  - Versionnage : Le code de l'infrastructure peut être versionné et contrôlé dans des systèmes de gestion de version (comme Git), permettant un suivi clair des modifications.
  - Reproductibilité : Les infrastructures peuvent être déployées de manière cohérente et reproductible, réduisant les erreurs humaines et les différences entre les environnements.
  - Automatisation : Le processus de déploiement de l'infrastructure peut être automatisé, réduisant les coûts et les efforts manuels.

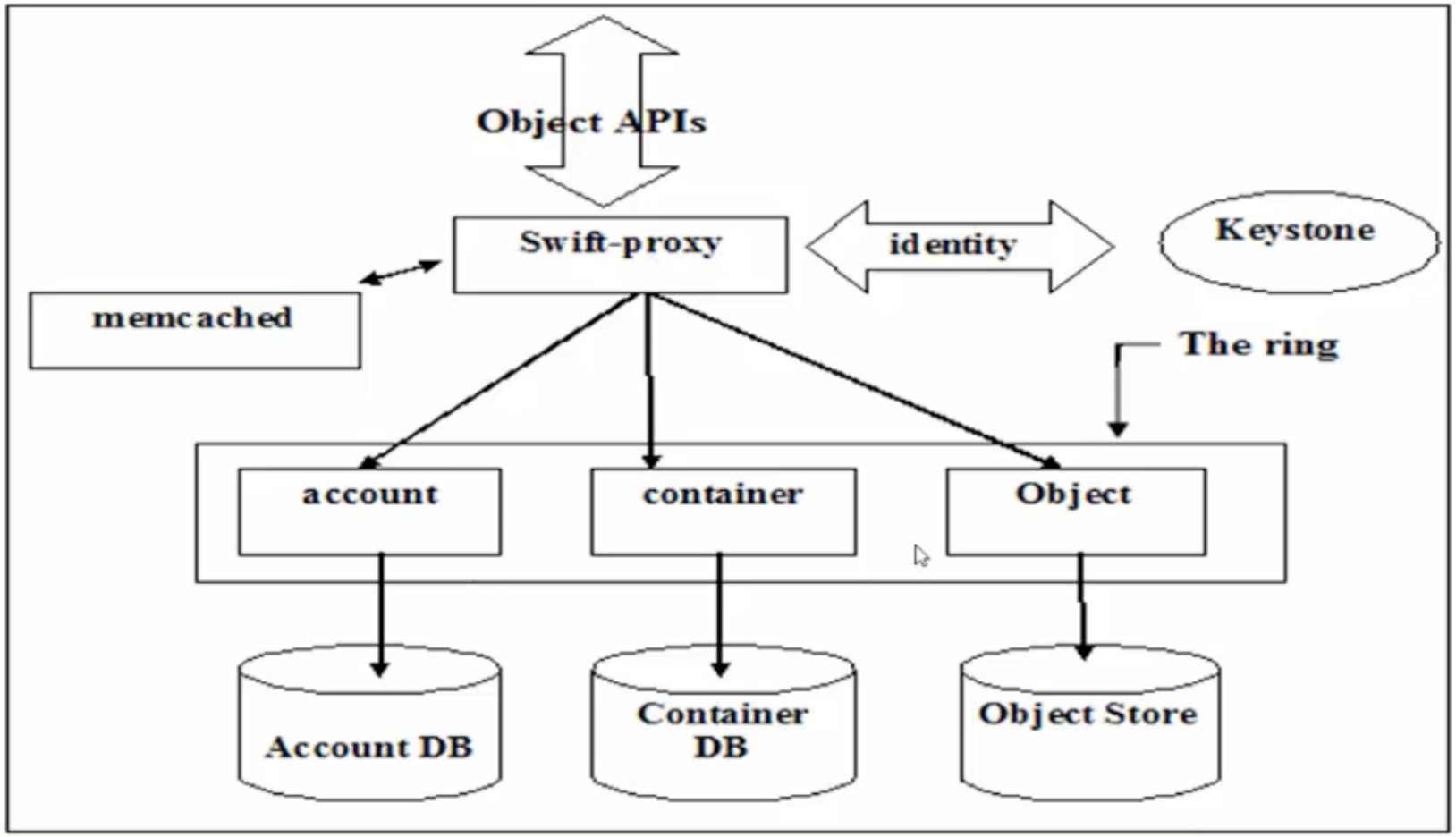
Présentation d'OpenStack

- OpenStack est un logiciel open-source qui permet de créer et gérer des infrastructures de cloud computing.
- Composants principaux d'OpenStack :
  - Nova (compute),
  - Swift (stockage objet),
  - Cinder (stockage bloc),
  - Neutron (réseau)
  - Keystone (identité).

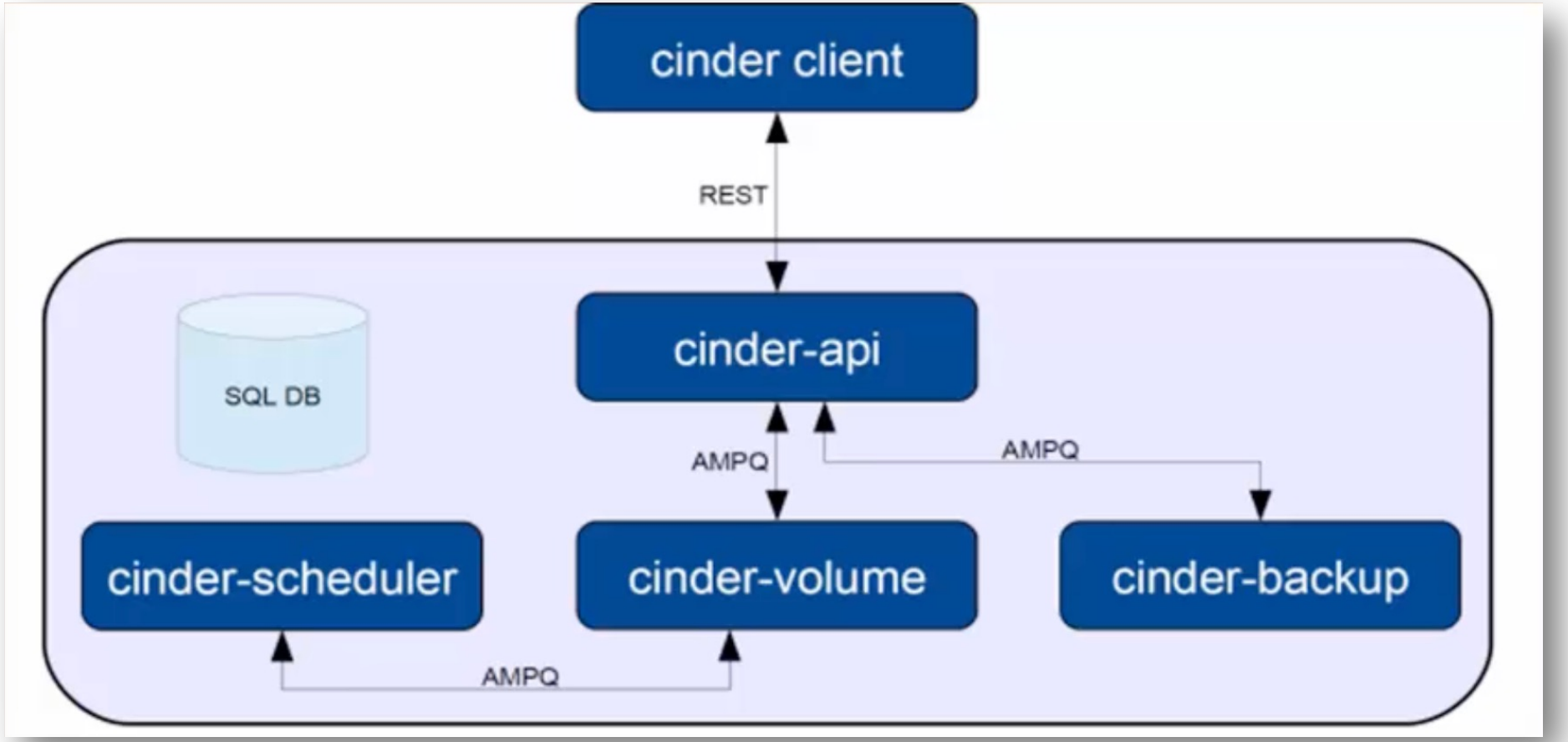
Le composant Nova



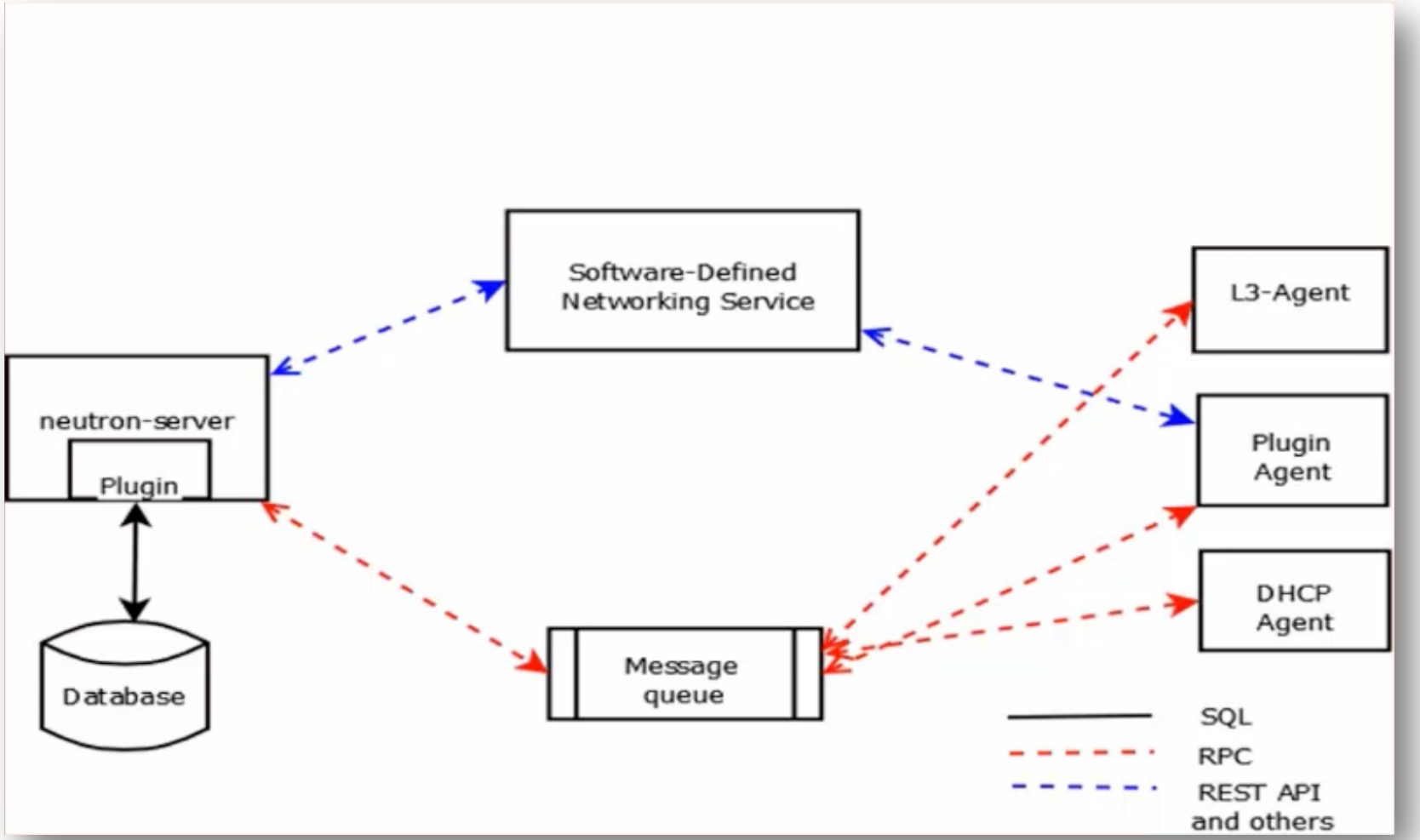
Le composant Swift



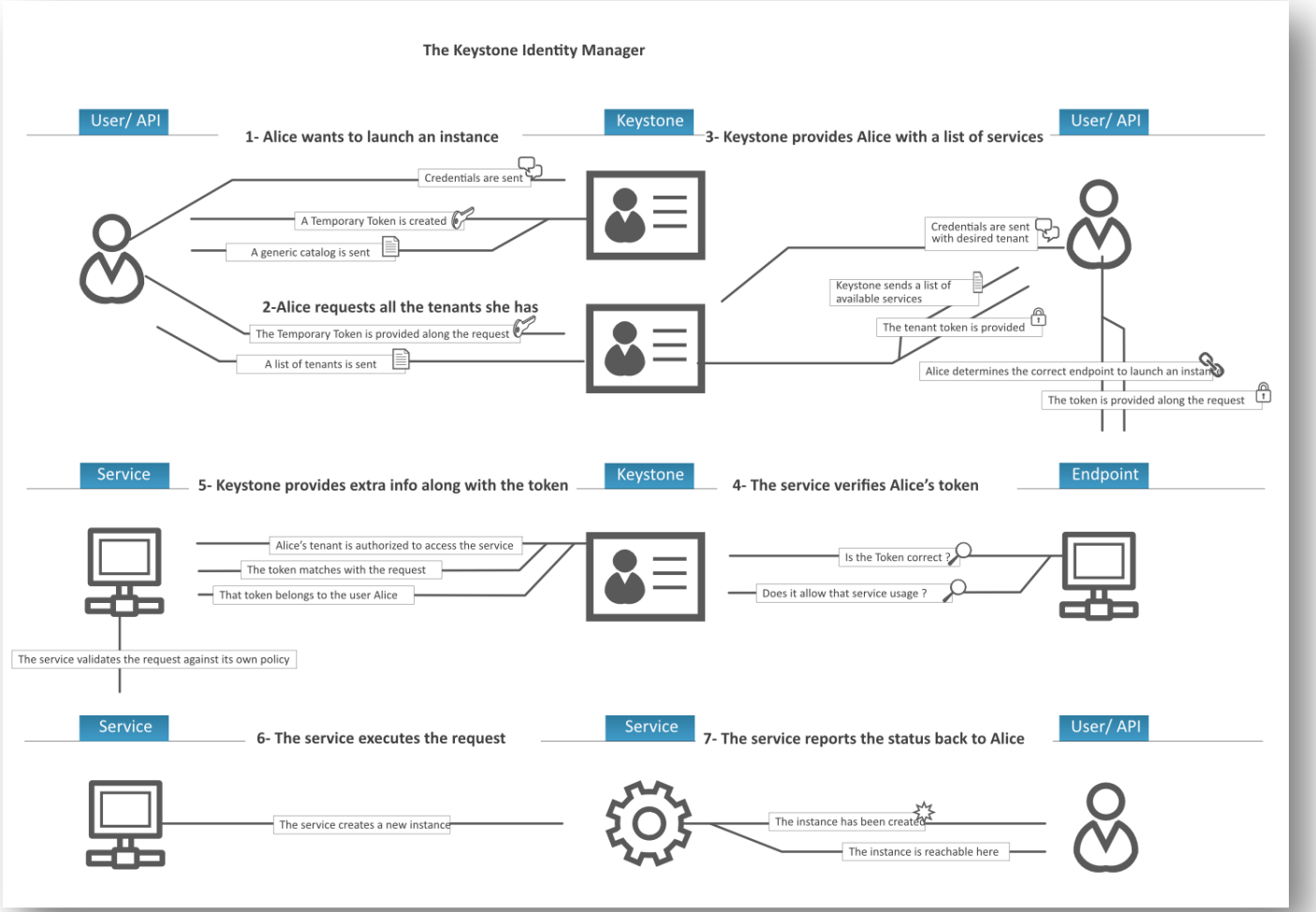
Le composant Cinder



Le composant Neutron



Le composant Keystone



Installation de Terraform

- Télécharger Terraform à partir du site officiel : https://www.terraform.io/downloads.html
- Installer Terraform en suivant les instructions pour votre système d'exploitation

Installation du client OpenStack

- Installer le client OpenStack en utilisant la commande :

```
pip install python-openstackclient
```

- Vérifier l'installation en exécutant la commande :

```
openstack --version
```

Configuration des accès API pour OpenStack

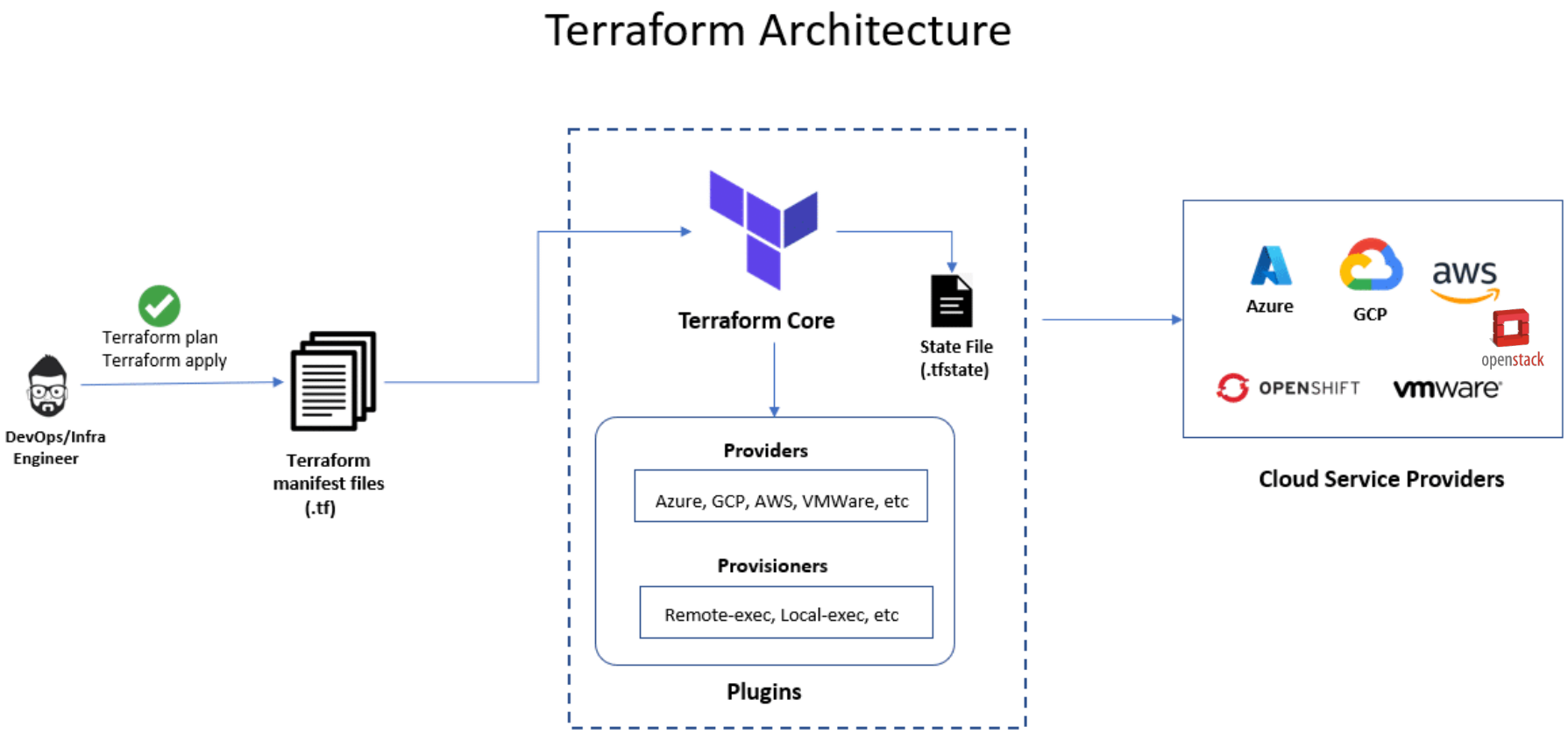
- Récupérer le fichier de configuration OpenStack (souvent nommé openrc.sh) auprès de votre administrateur OpenStack ou via le tableau de bord
- Charger le fichier de configuration en exécutant :

```
source openrc.sh
```

Exercice 1 - Installation et configuration

- Installer Terraform et le client OpenStack sur votre machine
- Configurer l'accès API pour OpenStack en utilisant un fichier openrc.sh

Schéma simplifié



Les providers

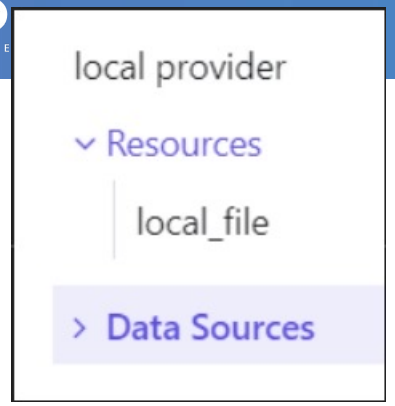
- Terraform est cloud agnostic.
- Les providers sont des plugins qui permettent à Terraform de communiquer avec des services externes
- Le provider OpenStack permet à Terraform de gérer les ressources OpenStack

Les ressources

- Les ressources représentent des éléments d'infrastructure, tels que des instances, des réseaux ou des volumes
- Exemple de ressource OpenStack :

```
resource "openstack_compute_instance_v2" "example"
```

Example fichier



.tf Extension

Block

Resource Type - Local

Resource Name

Arguments

main.tf

```
1  resource local_file sample_res {
2    filename = "sample.txt"
3    content = "I love Terraform"
4  }
```

Les variables

- Les variables permettent de paramétrer les configurations Terraform et de les rendre réutilisables
- Exemple :

```
variable "image_id" { default = "abc123" }
```

Les outputs

- Les outputs permettent d'afficher des informations sur les ressources créées par Terraform
- Exemple :

```
output "instance_ip" { value = openstack_compute_instance_v2.example.access_ip_v4 }
```

Les modules

- Les modules permettent de regrouper et réutiliser des configurations Terraform
- Ils facilitent la gestion et l'organisation du code

Configuration du provider OpenStack

- Ajouter le provider OpenStack à votre configuration Terraform

Exemple :

```
provider "openstack" {
  user_name   = "myuser"
  tenant_name = "mytenant"
  password    = "mypassword"
  auth_url    = "https://myopenstack.example.com:5000/v3"
}
```

Planification avec Terraform

- Exécuter `terraform init` pour initialiser le projet
- Exécuter `terraform plan` pour générer un plan d'exécution
- Analyser le plan d'exécution pour vérifier les modifications apportées à l'infrastructure

Déploiement avec Terraform

- Exécuter `terraform apply` pour appliquer le plan d'exécution et déployer l'infrastructure
- Vérifier que les ressources ont été créées dans OpenStack

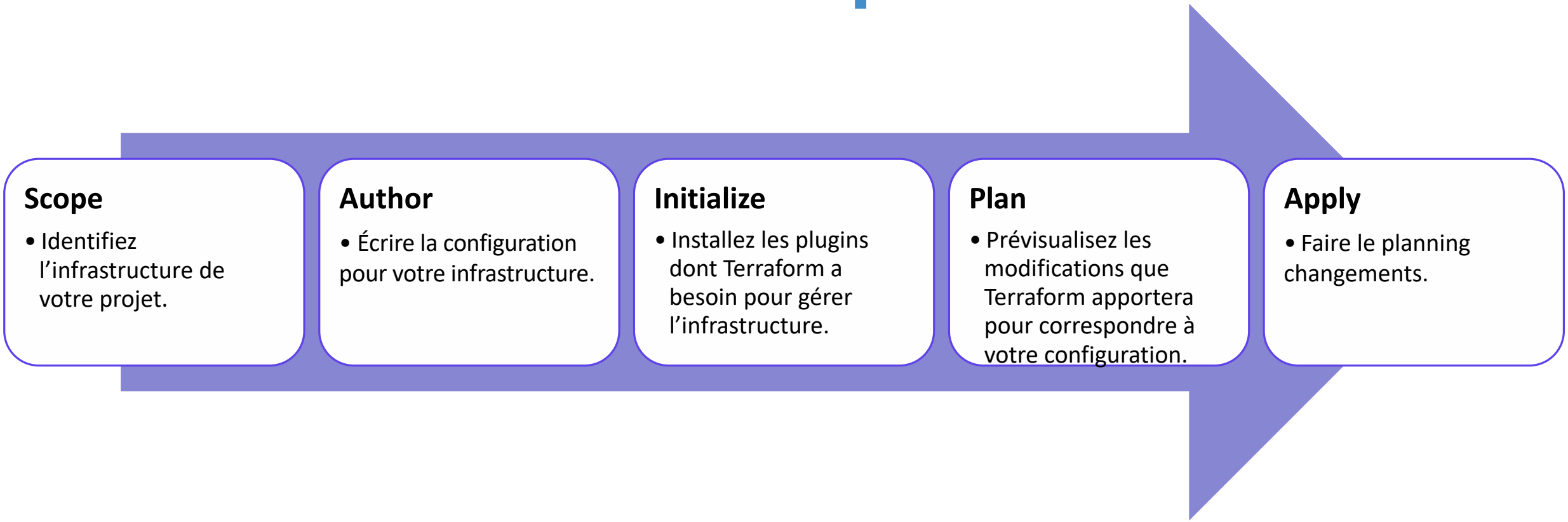
Mise à jour de l'infrastructure

- Modifier la configuration Terraform pour mettre à jour les ressources
- Exécuter `terraform plan` et `terraform apply` pour appliquer les modifications

Destruction de l'infrastructure

- Exécuter `terraform destroy` pour supprimer toutes les ressources créées par Terraform
- Vérifier que les ressources ont été supprimées dans OpenStack

Schéma simplifié



Création d'un réseau et d'un sous-réseau

- Créer un réseau avec openstack_networking_network_v2
- Créer un sous-réseau avec openstack_networking_subnet_v2
- Exemple de configuration :

```
resource "openstack_networking_network_v2" "example_network" {
  name = "example-network"
}

resource "openstack_networking_subnet_v2" "example_subnet" {
  name = "example-subnet"
  network_id = openstack_networking_network_v2.example_network.id
  cidr = "192.168.0.0/24"
}
```

Création d'une instance

- Créer une instance avec openstack_compute_instance_v2
- Spécifier l'image, le flavor, le réseau et le groupe de sécurité
- Exemple de configuration :

```
resource "openstack_compute_instance_v2" "example" {
  name = "example-instance"
  image_id = "image_id"
  flavor_id = "flavor_id"
  key_pair = "key_pair_name"
  security_groups = ["default", "example_security_group"]

  network {
    uuid = openstack_networking_network_v2.example_network.id
  }
}
```

Création d'un groupe de sécurité

- Créer un groupe de sécurité avec openstack_compute_secgroup_v2
- Ajouter des règles avec openstack_compute_secgroup_rule_v2
  Exemple de configuration :

```
resource "openstack_compute_secgroup_v2" "example" {
  name = "example_security_group"
  description = "Example security group"
}

resource "openstack_compute_secgroup_rule_v2" "example_rule" {
  direction = "ingress"
  ethertype = "IPv4"
  protocol = "tcp"
  port_range_min = 22
  port_range_max = 22
  remote_ip_prefix = "0.0.0.0/0"
  security_group_id = openstack_compute_secgroup_v2.example.id
}
```

Exercice 2 - Création d'une instance simple

- Créer un fichier de configuration Terraform pour déployer une instance OpenStack
- Utiliser le provider OpenStack et la ressource openstack_compute_instance_v2
- Appliquer la configuration

Exercice 3 - Configuration d'un réseau et d'un sous-réseau

- Modifier la configuration de l'exercice 2 Terraform pour inclure un réseau et un sous-réseau personnalisés
- Utiliser les ressources openstack_networking_network_v2 et openstack_networking_subnet_v2
- Appliquer les modifications

Exercice 4 - Ajout d'un groupe de sécurité

- Ajouter un groupe de sécurité à la configuration Terraform
- Utiliser les ressources openstack_compute_secgroup_v2 et openstack_compute_secgroup_rule_v2
- Appliquer les modifications

Création d'un volume

- Créer un volume avec openstack_blockstorage_volume_v2
- Attacher le volume à l'instance avec openstack_compute_volume_attach_v2
- Exemple de configuration :

```
resource "openstack_blockstorage_volume_v2" "example" {
  name = "example-volume"
  size = 10
}

resource "openstack_compute_volume_attach_v2" "example_attach" {
  instance_id = openstack_compute_instance_v2.example.id
  volume_id = openstack_blockstorage_volume_v2.example.id
}
```

Exercice 6 - Création de volumes

- Créez un fichier pour définir les variables nécessaires.
- Créez un fichier pour définir les ressources suivantes :
  - openstack_blockstorage_volume_v3 (volume)
  - openstack_compute_instance_v2 (instance)
  - openstack_compute_volume_attach_v2 (attachement du volume)
- Utilisez les variables pour paramétrer les ressources.
- Configurez la taille du volume, le type de volume et les autres paramètres nécessaires.
- Déployez l'infrastructure.
- Vérifiez que le volume a été créé et attaché à l'instance dans OpenStack.
- Supprimez l'infrastructure.

Création d'un routeur et connexion au réseau externe

- Créer un routeur avec openstack_networking_router_v2
- Connecter le routeur au réseau externe avec openstack_networking_router_interface_v2
- Exemple de configuration :

```
resource "openstack_networking_router_v2" "example" {
  name = "example-router"
  external_gateway = "external_network_id"
}

resource "openstack_networking_router_interface_v2" "example_interface" {
  router_id = openstack_networking_router_v2

}
```

Création d'un équilibreur de charge

- Créer un équilibreur de charge avec openstack_lb_loadbalancer_v2
- Ajouter des auditeurs avec openstack_lb_listener_v2
- Ajouter des pools avec openstack_lb_pool_v2
- Ajouter des membres au pool avec openstack_lb_member_v2
- Exemple de configuration :

Création d'un équilibreur de charge

```
resource "openstack_lb_loadbalancer_v2" "example" {
name = "example-loadbalancer"
vip_subnet_id = openstack_networking_subnet_v2.example_subnet.id
}
    resource "openstack_lb_listener_v2" "example" {
      name = "example-listener"
      protocol = "HTTP"
      protocol_port = 80
      loadbalancer_id = openstack_lb_loadbalancer_v2.example.id
    }

    resource "openstack_lb_pool_v2" "example" {
      name = "example-pool"
      protocol = "HTTP"
      lb_method = "ROUND_ROBIN"
      listener_id = openstack_lb_listener_v2.example.id
    }

    resource "openstack_lb_member_v2" "example" {
      address = "192.168.0.10"
      protocol_port = 80
      subnet_id = openstack_networking_subnet_v2.example_subnet.id
      pool_id = openstack_lb_pool_v2.example.id
    }
```

Exercice 7 - Création d'un équilibreur de charge

- Créez un fichier pour définir les variables nécessaires.
- Créez un fichier pour définir les ressources suivantes :
  - openstack_networking_network_v2 (réseau)
  - openstack_networking_subnet_v2 (sous-réseau)
  - openstack_compute_instance_v2 (2 instances)
  - openstack_lb_loadbalancer_v2 (équilibreur de charge)
  - openstack_lb_listener_v2 (auditeur)
  - openstack_lb_pool_v2 (pool)
  - openstack_lb_member_v2 (membres du pool)
- Utilisez les variables pour paramétrer les ressources.
- Déployez l'infrastructure.
- Supprimez l'infrastructure.

Bonnes pratiques - Utilisation de modules

- Utiliser des modules pour organiser et réutiliser des configurations Terraform
- Créer des modules personnalisés ou utiliser des modules existants dans le registre Terraform

Exercice 8 - Création d'un module

- Créer un module Terraform pour encapsuler la configuration du réseau, du sous-réseau et du groupe de sécurité
- Utiliser le module dans la configuration principale

Bonnes pratiques - Gestion des états

- Comprendre l'importance de la gestion des états Terraform
- Utiliser des backends distants pour stocker et partager les états entre les membres de l'équipe

Bonnes pratiques - Sécurité et confidentialité

- Ne pas inclure les informations sensibles, telles que les mots de passe, directement dans les fichiers de configuration
- Utiliser des variables d'environnement ou des fichiers d'entrée pour gérer les informations sensible

Collaboration et versionnage

- Utiliser des systèmes de gestion de version, tels que Git, pour suivre les modifications apportées aux fichiers de configuration Terraform
- Collaborer avec les membres de l'équipe en utilisant des outils comme GitHub ou GitLab pour les demandes de fusion (merge requests) et les revues de code

Test et validation

- Utiliser des outils de validation et de vérification de la syntaxe pour les fichiers de configuration Terraform, tels que `terraform validate` et `terraform fmt`
- Mettre en place des tests d'intégration pour valider que les ressources créées fonctionnent comme prévu

Test et validation

- Comprendre les erreurs courantes et comment les résoudre
- Utiliser des commandes comme `terraform refresh` et `terraform import` pour résoudre les problèmes liés à l'état de l'infrastructure

Approfondissements avec Terraform :

- Segmentation d’infrastructure pour limiter les périmètres opérationnels.
- Organisation des équipes et périmètres de responsabilité.

Segmentation d'infrastructure pour limiter les périmètres opérationnels.

- La segmentation d'infrastructure pour limiter les périmètres opérationnels est une pratique qui consiste à diviser l'infrastructure en segments distincts afin de réduire les risques et les impacts potentiels liés aux opérations.

- Cette approche vise à isoler les différents environnements, applications, services ou équipes au sein de l'infrastructure, afin de minimiser les effets de tout incident ou problème sur l'ensemble du système.

Segmentation d'infrastructure pour limiter les périmètres opérationnels.

- L'objectif principal de la segmentation d'infrastructure est de limiter la propagation des erreurs ou des défaillances en confinant leur impact à un segment spécifique plutôt que de les laisser se propager à travers toute l'infrastructure.
- Cela permet de maintenir la stabilité et la disponibilité des systèmes en réduisant les risques opérationnels.

Quelques points clés de la segmentation d'infrastructure pour limiter les périmètres opérationnels

1. **Isolation des environnements** : La segmentation par environnement, tels que développement, staging et production, permet de séparer les différentes phases du cycle de vie des logiciels. Cela permet aux équipes de travailler de manière indépendante sur chaque environnement, réduisant ainsi les risques de conflits ou d'interférences.
2. **Segmentation par application ou service** : La segmentation par application ou service implique de regrouper les ressources qui appartiennent à une même application ou à un même service. Cela facilite la gestion et la maintenance de ces ressources, ainsi que le déploiement et la mise à l'échelle spécifiques à chaque application ou service.

Quelques points clés de la segmentation d'infrastructure pour limiter les périmètres opérationnels

3. **Segmentation par équipe ou projet** : La segmentation par équipe ou projet permet de définir des périmètres de responsabilité clairs au sein de l'infrastructure. Chaque équipe est responsable de ses propres ressources et de leur configuration, ce qui facilite la collaboration et la gestion des accès.
4. **Contrôle des accès et des autorisations** : La segmentation d'infrastructure implique également la mise en place de politiques de contrôle des accès et des autorisations spécifiques à chaque segment. Cela garantit que seules les personnes autorisées peuvent accéder et effectuer des modifications dans un segment donné, renforçant ainsi la sécurité globale de l'infrastructure.

L'organisation des équipes et des périmètres de responsabilité avec Terraform

Quelques conseils sur la façon d'organiser les équipes et de définir les périmètres de responsabilité :

1. Création de workspaces
2. Structure du code
3. Politiques de gouvernance
4. Collaboration et communication
5. Bonnes pratiques de sécurité
6. Documentation et formation

L'organisation des équipes et des périmètres de responsabilité avec Terraform

1.Création de workspaces :

1. Utilisez les workspaces de Terraform pour isoler les environnements et les segments de l'infrastructure. Chaque équipe peut travailler dans son propre workspace, ce qui permet une séparation claire des responsabilités.

2.Structure du code :

1. Organisez votre code Terraform de manière à refléter la structure organisationnelle et les responsabilités des équipes.
2. Utilisez des modules Terraform pour encapsuler la logique spécifique à chaque équipe, application ou service. Cela facilite la réutilisation du code et maintient la cohérence dans l'ensemble de l'organisation.

L'organisation des équipes et des périmètres de responsabilité avec Terraform

3. **Politiques de gouvernance :**
    1. Définissez des politiques de gouvernance claires pour l'accès et les modifications des ressources Terraform.
    2. Utilisez les fonctionnalités de contrôle d'accès et d'autorisation de votre infrastructure pour restreindre l'accès aux ressources spécifiques à chaque équipe.
4. **Collaboration et communication :**
    1. Favorisez la collaboration entre les équipes en utilisant des outils de gestion de code source tels que Git et des plateformes de collaboration pour partager le code Terraform et les configurations.
    2. Établissez des canaux de communication clairs pour faciliter la coordination et la résolution des problèmes entre les équipes.

L'organisation des équipes et des périmètres de responsabilité avec Terraform

5. **Bonnes pratiques de sécurité :**
    1. Mettez en place des mesures de sécurité appropriées pour protéger les ressources Terraform.
    2. Utilisez des rôles IAM ou des groupes de sécurité pour contrôler l'accès aux ressources et implémentez des politiques de sécurité telles que le chiffrement des données sensibles.
6. **Documentation et formation :**
    1. Documentez les bonnes pratiques, les conventions de nommage et les procédures spécifiques à l'organisation.
    2. Fournissez une formation et un support adéquats pour que les équipes puissent utiliser Terraform de manière efficace et conforme aux politiques de l'organisation.