UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

# Terraform avec OpenStack

Utopios® Tous droits réservés

# Programme

1. Introduction
   1.1 Objectifs de la formation
   1.2 Présentation de Terraform
   1.3 Pourquoi utiliser Terraform avec OpenStack
2. Mise en place de l'environnement
   2.1 Installation de Terraform
   2.2 Configuration de Terraform
3. HCL
   3.1 Présentation de Terraform HCL
   3.2 Les différentes variables et commandes
   3.3 Le flux de travail de Terraform
   3.4 Les dépendances explicites et implicites entre les ressources
   3.5 Les cycles de vie des ressources
   3.6 Les expressions conditionnelles et itératives
   3.7 Les templates et fonctions intégrées
   3.8 Comprendre et gérer l'état de Terraform

Utopios® Tous droits réservés

# Programme

4. OpenStack et terraform
   - 4.1 Provider
   - 4.2 Gestion du réseau
   - 4.3 Gestion de l'Infrastructure
5. Terraform avancée
   - 5.1 Terraform Provisioner
   - 5.2 Automatiser son workflow avec Terraform
   - 5.3 Les modules Terraform
   - 5.4 Gestion des outputs Terraform
   - 5.5 Terraform et les environnements
6. Troubleshooting

Utopios® Tous droits réservés

# Objectifs de la formation

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

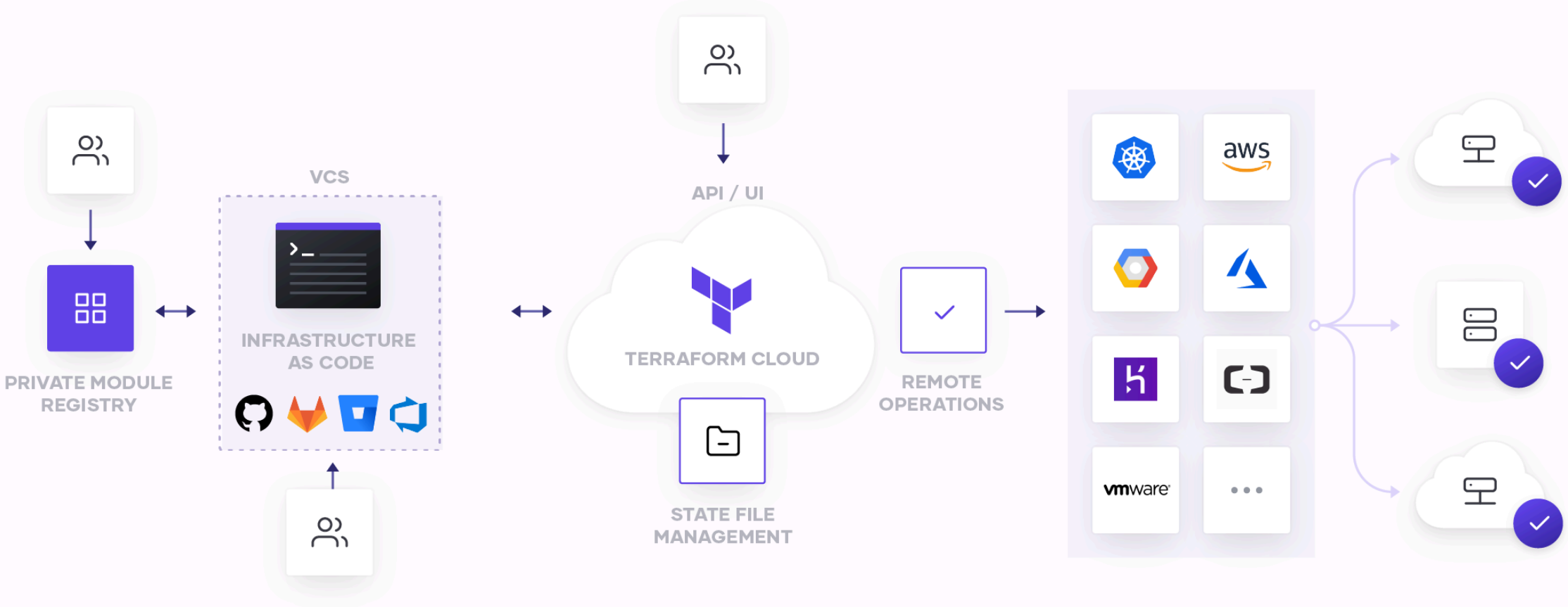
Utopios® Tous droits réservés

# 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.

Utopios® Tous droits réservés

# Présentation de Terraform




Utopios® Tous droits réservés

# Pourquoi utiliser Terraform avec OpenStack ?

**Gestion simplifiée de l'infrastructure**: Avec Terraform, vous pouvez définir toute votre infrastructure Terraform dans un fichier de configuration. Cela rend la gestion de l'infrastructure plus facile et plus transparente car tout est codé dans un fichier ou un ensemble de fichiers.
**Modularité et réutilisabilité**: Terraform vous permet de créer des modules pour les différents composants de votre infrastructure. Cela permet de réutiliser les configurations pour différents environnements ou projets.
**Gestion des dépendances**: Terraform gère les dépendances entre les différents composants de l'infrastructure. Par exemple, si une machine virtuelle dépend d'un réseau virtuel, Terraform s'assurera que le réseau virtuel est créé avant la machine virtuelle.
**Gestion des changements**: Terraform compare l'état actuel de l'infrastructure avec la configuration définie et ne fait que les modifications nécessaires pour aligner les deux. Cela permet une gestion des changements plus efficace et réduit le risque d'erreurs.
**Interopérabilité**: Terraform peut être utilisé avec de nombreux autres fournisseurs de services cloud et d'outils de gestion de l'infrastructure. Cela le rend idéal pour les entreprises qui ont une infrastructure hybride ou multi-cloud.
**Outil open-source**: Terraform est un outil open-source, ce qui signifie qu'il est gratuit à utiliser et a une grande communauté de développeurs qui contribuent à son développement et à son support.

Utopios® Tous droits réservés

# Installation de Terraform

1. **Téléchargez Terraform**: Allez sur le site officiel de Terraform et téléchargez la dernière version de Terraform pour votre système d'exploitation. Voici le lien pour télécharger Terraform : [Terraform Downloads].
2. **Décompressez le fichier**: Une fois le fichier téléchargé, décompressez le fichier dans un répertoire de votre choix.
3. **Ajoutez Terraform à votre PATH**:

- Sur Windows:
  - Copiez le fichier terraform.exe dans un répertoire qui est dans votre PATH, ou ajoutez le répertoire où se trouve terraform.exe à votre PATH.
  - Pour ajouter un répertoire à votre PATH, vous pouvez le faire via les paramètres système ou en utilisant la ligne de commande. Par exemple, si vous avez décompressé terraform.exe dans le répertoire C:\terraform, vous pouvez ajouter ce répertoire à votre PATH en exécutant la commande suivante dans l'invite de commande : setx PATH "%PATH%;C:\terraform".

Utopios® Tous droits réservés

# Installation de Terraform

- Sur macOS et Linux:
  - Déplacez le fichier terraform dans un répertoire qui est dans votre PATH. Par exemple, vous pouvez déplacer terraform dans le répertoire /usr/local/bin en exécutant la commande suivante : mv terraform /usr/local/bin/.

4. **Vérifiez l'installation**: Ouvrez une nouvelle invite de commande ou un terminal et exécutez la commande terraform -v. Si Terraform est correctement installé, cette commande affichera la version de Terraform que vous avez installée.

Utopios® Tous droits réservés

# Présentation de Terraform HCL

- **HCL (HashiCorp Configuration Language)** est un langage de configuration développé par HashiCorp, la société derrière Terraform. HCL est conçu pour être à la fois lisible par l'homme et lisible par la machine, ce qui le rend idéal pour la configuration de l'infrastructure.
- HCL est utilisé pour écrire les fichiers de configuration Terraform. Les fichiers de configuration Terraform décrivent les ressources et les infrastructures à provisionner.
- **Structure de base**: Les fichiers de configuration Terraform, écrits en HCL, sont structurés en trois blocs principaux: provider, resource et output.
    1. Le bloc provider configure le fournisseur de cloud à utiliser (par exemple, AWS, Azure, GCP, etc.).
    2. Le bloc resource définie une ressource à provisionner (par exemple, une machine virtuelle, un réseau, une base de données, etc.).
    3. Le bloc output définie les valeurs à afficher à l'utilisateur après le provisionnement de l'infrastructure.

Utopios® Tous droits réservés

# Les différentes variables et commandes

## Commandes terraform

- **terraform init**: Initialise le répertoire de travail de Terraform. Cette commande doit être exécutée avant d'autres commandes Terraform.
- **terraform plan**: Crée un plan d'exécution. Cette commande montre quelles actions Terraform effectuera pour aligner l'infrastructure sur la configuration.
- **terraform apply**: Applique les changements nécessaires pour aligner l'infrastructure sur la configuration.
- **terraform destroy**: Détruit les ressources créées par Terraform.

Utopios® Tous droits réservés

# Le flux de travail de Terraform

1. **Write** : La première étape du flux de travail de Terraform consiste à écrire un fichier de configuration. Ce fichier, souvent appelé main.tf, définira toutes les ressources que vous souhaitez créer ou gérer. Dans le contexte d'openstack, cela signifierait définir des ressources comme des machines virtuelles, reseaux,..., en utilisant le fournisseur openstack et le langage de configuration de Terraform (HCL).
2. **Plan** : Après avoir écrit votre configuration, l'étape suivante est de créer un plan d'exécution. Vous pouvez faire cela en exécutant la commande terraform plan. Cette commande permet à Terraform de simuler les actions qu'il effectuera en fonction de la configuration actuelle par rapport à l'état actuel de votre infrastructure. Cela permet de voir quelles actions Terraform effectuera sans effectuer réellement de modifications.
3. **Apply** : Une fois que vous avez vérifié que le plan d'exécution est correct, la dernière étape est d'appliquer les modifications en exécutant la commande terraform apply. Cela demandera à Terraform de créer, mettre à jour ou détruire les ressources nécessaires pour correspondre à la configuration spécifiée dans votre fichier main.tf.

Utopios® Tous droits réservés

# Les différentes variables et commandes

## variables terraform

- Les variables dans Terraform sont un moyen de définir des valeurs qui peuvent être réutilisées dans toute votre configuration. Vous pouvez définir des valeurs par défaut pour vos variables et les substituer au moment de l'exécution si nécessaire
- *__Déclaration__ : Vous devez d'abord déclarer vos variables avant de les utiliser. Habituellement, les variables sont déclarées dans un fichier séparé appelé variables.tf, mais elles peuvent être déclarées dans n'importe quel fichier .tf dans votre configuration.

Utopios® Tous droits réservés

# Les différentes variables et commandes

## variables terraform

- variables.tf

```
variable "datacenter" {
  description = "The datacenter in vSphere to deploy to"
  type        = string
  default     = "dc-01"
}

variable "resource_pools" {
  description = "The resource pools to deploy VMs to"
  type        = list(string)
  default     = ["resource-pool-1", "resource-pool-2"]
}

variable "vm_tags" {
  description = "The tags to apply to virtual machines"
  type        = map(string)
  default     = { Owner = "DevOps", Environment = "Production" }
}
```

Utopios® Tous droits réservés

# Les différentes variables et commandes

## variables terraform

- Utilisation dans main.tf

```
output "selected_datacenter" {
  value = var.datacenter
}

output "selected_resource_pools" {
  value = var.resource_pools
}

output "vm_common_tags" {
  value = var.vm_tags
}
```

Utopios® Tous droits réservés

# Les différentes variables et commandes

## variables terraform

- Dans Terraform, les variables peuvent être de plusieurs types, dont voici quelques-uns des plus couramment utilisés

1. **String**: Il s'agit du type le plus basique. Une variable de type string est simplement une chaîne de caractères.

```
variable "example_string" {
  type    = string
  default = "Hello, World"
}
```

2. **Number**: Ce type est utilisé pour les variables qui sont des nombres. Ils peuvent être des entiers ou des nombres à virgule flottante.

```
variable "example_number" {
  type    = number
  default = 42
}
```

Utopios® Tous droits réservés

# Les différentes variables et commandes

## variables terraform

3. **Bool**: Ce type est utilisé pour les variables qui sont soit true, soit false.

```
variable "example_bool" {
  type    = bool
  default = true
}
```

4. **List (ou tuple)**: Une liste est une collection ordonnée d'éléments du même type.

```
variable "example_list" {
  type    = list(string)
  default = ["one", "two", "three"]
}
```

Utopios® Tous droits réservés

# Les différentes variables et commandes

## variables terraform

5. **Map**: Une carte est une collection non ordonnée d'éléments de type clé-valeur

```
variable "example_map" {
  type = map(string)
  default = {
    key1 = "value1"
    key2 = "value2"
  }
}
```

6. **Set**: Un ensemble est une collection non ordonnée d'éléments uniques du même type.

```
variable "example_set" {
  type    = set(string)
  default = ["one", "two", "three"]
}
```

Utopios® Tous droits réservés

# Les différentes variables et commandes

## variables terraform

7. **Object**: Un objet est une collection d'attributs avec des types spécifiés

```
variable "example_object" {
  type = object({
    attribute1 = string
    attribute2 = list(number)
  })
  default = {
    attribute1 = "value"
    attribute2 = [1, 2, 3]
  }
}
```

Utopios® Tous droits réservés

# Les différentes variables et commandes

## Exercice

- Configurer une configuration de base Terraform qui utilise différentes variables pour personnaliser les valeurs des sorties, y compris des variables qui dépendent d'autres variables.
- Dans le fichier variables.tf, déclarez les variables suivantes :
  - Une variable first_name de type string sans valeur par défaut.
  - Une variable last_name de type string sans valeur par défaut.
  - Une variable age de type number sans valeur par défaut.
  - Une variable is_student de type bool avec une valeur par défaut de false.
  - Une variable courses de type list(string) avec une valeur par défaut de ["Math", "Science"].
  - Une variable grades de type map(number) avec une valeur par défaut de { Math = 90, Science = 85 }.
  - Une variable student de type object qui a les attributs suivants : first_name de type string. last_name de type string. age de type number. is_student de type bool. courses de type list(string). grades de type map(number).

Utopios® Tous droits réservés

# Les dépendances explicites et implicites entre les ressources

- Les dépendances entre les ressources sont essentielles pour déterminer l'ordre dans lequel les ressources doivent être créées, mises à jour ou détruites

1. **Dépendances implicites** :

- Les dépendances implicites sont créées automatiquement par Terraform lorsque vous utilisez une référence à une autre ressource dans la configuration d'une ressource.

Utopios® Tous droits réservés

# Les dépendances explicites et implicites entre les ressources

2. **Dépendances explicites** :

- Vous pouvez également spécifier des dépendances explicites à l'aide de l'attribut depends_on. Cela peut être nécessaire dans certaines situations où Terraform ne peut pas déterminer les dépendances automatiquement.

Utopios® Tous droits réservés

# Les cycles de vie des ressources

- Les cycles de vie des ressources dans Terraform définissent l'ordre et le comportement des actions qui seront effectuées sur une ressource. Par exemple, lorsqu'une ressource doit être créée, mise à jour ou détruite. Le bloc lifecycle dans la configuration d'une ressource permet de contrôler ce comportement.
- **create_before_destroy**: Un booléen qui, lorsqu'il est défini sur true, indique à Terraform de créer la nouvelle ressource avant de détruire l'ancienne lors d'une mise à jour.
- **prevent_destroy**: Un booléen qui, lorsqu'il est défini sur true, empêchera la ressource d'être détruite. Cela peut être utile pour éviter la suppression accidentelle de ressources critiques.
- **ignore_changes**: Une liste d'attributs qui seront ignorés lors de la mise à jour d'une ressource. Cela peut être utile pour ignorer les modifications apportées par des sources externes à Terrafor
- **replace_triggered_by**: Il permet de spécifier une liste d'attributs de la ressource qui, lorsqu'ils sont modifiés, déclenchent la destruction et la recréation de la ressource au lieu de simplement la mettre à jour

Utopios® Tous droits réservés

# Les cycles de vie des ressources

- **Exemple**

```
resource "example_resource" "example" {
  # ...

  lifecycle {
    create_before_destroy = true
    prevent_destroy       = true
    ignore_changes        = [example_attribute]
    replace_triggered_by = [example_attribute]
  }
}
```

Utopios® Tous droits réservés

# Les cycles de vie des ressources Exercice

1. Déclarez deux variables, example_attribute1 et example_attribute2, toutes deux de type string avec des valeurs par défaut.
2. Configurez quatre ressources fictives : example_resource_1, example_resource_2, example_resource_3, et example_resource_4.
    - Pour example_resource_1, utilisez la variable example_attribute1 et configurez son bloc lifecycle pour que la nouvelle ressource soit créée avant la destruction de l'ancienne, et pour ignorer les modifications de example_attribute1.
    - Pour example_resource_2, utilisez la variable example_attribute2, configurez une dépendance explicite sur example_resource_1 en utilisant depends_on, et configurez son bloc lifecycle pour empêcher la destruction de la ressource.
    - Pour example_resource_3, configurez une dépendance explicite sur example_resource_2 et example_resource_1 en utilisant depends_on.
    - Pour example_resource_4, configurez une dépendance explicite sur example_resource_3 en utilisant depends_on, et configurez son bloc lifecycle pour ignorer les modifications de toutes les propriétés, à l'exception de example_attribute2.

Utopios® Tous droits réservés

# Les expressions conditionnelles et itératives

- Les expressions conditionnelles dans Terraform permettent d'évaluer une condition et de retourner une valeur en fonction du résultat de cette condition. La syntaxe de base pour une expression conditionnelle est la suivante

```
condition ? true_value : false_value
```

- Exemple :
  - *variables.tf*

```
variable "environment" {
  description = "The environment"
  type        = string
}
```

- *main.tf*

```
resource "example_resource" "example" {
example_attribute = var.environment == "production" ? "production_value" : "non_production_value"
}
```

Utopios® Tous droits réservés

# Les expressions conditionnelles et itératives

- les expressions itératives, comme for et for_each, permettent de créer plusieurs ressources ou d'effectuer des opérations sur chaque élément d'une collection

1. **Expression for** :

- L'expression for peut être utilisée pour transformer une liste ou un ensemble de valeurs. Elle peut également être utilisée pour filtrer ou transformer les éléments d'une collection.

```
variable "list_of_strings" {
  description = "A list of strings"
  type        = list(string)
  default     = ["apple", "banana", "cherry"]
}

output "lengths_of_strings" {
  value = [for s in var.list_of_strings : length(s)]
}
```

Utopios® Tous droits réservés

# Les expressions conditionnelles et itératives

2. **L'attribut for_each** :

- L'attribut for_each peut être utilisé pour créer une instance de ressource pour chaque élément dans une collection.

```
variable "set_of_names" {
  description = "A set of names"
  type        = set(string)
  default     = ["Alice", "Bob", "Charlie"]
}

resource "example_resource" "example" {
  for_each = var.set_of_names

  name = each.value
}
```

Utopios® Tous droits réservés

# Les expressions conditionnelles et itératives

## Exercice

**Structures conditionnelles** :

- Créez un fichier main.tf et déclarez une variable environment qui prendra deux valeurs possibles : "production" ou "development".
- Déclarez une autre variable type qui prendra deux valeurs possibles : "small" ou "large".
- Utilisez une expression conditionnelle pour définir une troisième variable isize basée sur la valeur de type:
  - Si type est "small", instance_size devrait être "micro".
  - Si type est "large", instance_size devrait être "big".

**Structures itératives** :

- Déclarez une variable names qui est une liste de noms.
- Utilisez une expression for pour créer une liste de messages de bienvenue pour chaque nom dans la liste names. Chaque message de bienvenue doit être formaté comme suit : "Welcome, [name]!"
- Affichez la valeur de instance_size et la liste des messages de bienvenue.

Utopios® Tous droits réservés

# Les templates et fonctions intégrées

1. **Templates** :

Les templates dans Terraform sont utilisés pour insérer des valeurs dans des chaînes de caractères. Vous pouvez utiliser les expressions et les fonctions de Terraform dans un template. La syntaxe des templates est ${...}.

```
variable "name" {
  default = "world"
}

output "greeting" {
  value = "Hello, ${var.name}!"
}
```

Utopios® Tous droits réservés

# Les templates et fonctions intégrées

2. **Fonctions intégrées** :

Terraform a un grand nombre de fonctions intégrées qui peuvent être utilisées pour transformer et combiner des valeurs. Ces fonctions peuvent être utilisées dans n'importe quelle expression dans une configuration Terraform

```
variable "list" {
  default = [1, 2, 3]
}
output "sum" {
  value = sum(var.list)
}
```

- Quelques autres fonctions intégrées communes sont :
  - length(list) : Retourne le nombre d'éléments dans une liste.
  - join(separator, list) : Combine les éléments d'une liste en une seule chaîne, en insérant un séparateur entre chaque élément.
  - split(separator, string) : Divise une chaîne en une liste d'éléments, en utilisant un séparateur pour déterminer les limites de chaque élément.

Utopios® Tous droits réservés

# Les templates et fonctions intégrées

## Exercice

- Créez un fichier main.tf et déclarez une variable names qui est une liste de noms.
- Combinez les noms de la liste names en une seule chaîne séparée par des virgules.
- Comptez le nombre de noms dans la liste names.
- Créez un fichier example.tpl avec le contenu suivant :

```
Hello, ${name}!
```

- Utilisez le fichier example.tpl et remplacer la variable ${name} par chaque nom de la liste names.

Utopios® Tous droits réservés

# Comprendre et gérer l'état de Terraform

- L'état de Terraform est un aspect crucial de Terraform.
- Il garde une trace de l'infrastructure que Terraform a provisionnée et est utilisé pour planifier et appliquer des modifications à cette infrastructure
- L'état de Terraform est un fichier JSON qui contient les propriétés actuelles de l'infrastructure provisionnée. Il est utilisé par Terraform pour mapper les ressources réelles aux ressources dans votre configuration, pour stocker les attributs des ressources managées, et pour optimiser les performances pour les grandes infrastructures.
- Par défaut, Terraform stocke l'état localement dans un fichier appelé terraform.tfstate. Cependant, il est recommandé de stocker l'état dans un emplacement distant, tel qu'un bucket S3 ou Azure Blob Storage, pour des raisons de sécurité, de performance et de collaboration

Utopios® Tous droits réservés

# Comprendre et gérer l'état de Terraform

## Gestion de l'état de Terraform

- **Stockage distant** : Vous pouvez configurer Terraform pour stocker l'état dans un emplacement distant. Cela est particulièrement important lorsque vous travaillez en équipe, car il permet de partager l'état de Terraform entre les membres de l'équipe.
- **Verrouillage d'état** : Le verrouillage d'état empêche d'autres utilisateurs de modifier l'état de Terraform en même temps. Certains backends d'état, comme S3 avec DynamoDB, prennent en charge le verrouillage d'état.
- **Séparation de l'état** : Pour les grandes infrastructures, il peut être utile de séparer l'état de Terraform en plusieurs fichiers d'état. Cela peut être accompli en utilisant les workspaces de Terraform ou en définissant des configurations de Terraform distinctes pour différentes parties de votre infrastructure.
- Vous pouvez inspecter l'état actuel de Terraform en utilisant la commande ```terraform show``````.
- `terraform state list` : Liste toutes les ressources dans l'état de Terraform.
- `terraform state show` : Affiche les détails d'une ressource spécifique dans l'état de Terraform.
- `terraform state mv` : Déplace une ressource d'un état à un autre ou à un autre emplacement dans le même état.
- `terraform state rm` : Supprime une ressource de l'état de Terraform

Utopios® Tous droits réservés

# Comprendre et gérer l'état de Terraform

## Sécurisation l'état de Terraform

- L'état de Terraform peut contenir des informations sensibles, comme des mots de passe ou des clés d'accès. Il est donc important de sécuriser l'état de Terraform.
- Stockez l'état dans un emplacement sécurisé et contrôlez l'accès à cet emplacement.
- Utilisez le chiffrement pour stocker l'état.
- Ne stockez jamais l'état de Terraform dans un dépôt de code public ou non sécurisé

Utopios® Tous droits réservés

# Comprendre et gérer l'état de Terraform

## Exercice

- Créez un fichier main.tf avec une ressource fictive

```
resource "null_resource" "example" {
  triggers = {
    always_run = "${timestamp()}"
  }
}
```

- Listez toutes les ressources dans votre état de Terraform.
- Affichez les détails de la ressource null_resource.example.
- Supprimez la ressource null_resource.

Utopios® Tous droits réservés

# Gestion des ressources openstack avec Terraform - Provider

- Un **provider** dans Terraform agit comme un intermédiaire entre Terraform et une plateforme spécifique. Pour Openstack, le provider est openstack. Ce provider définit les ressources spécifiques à openstack.
- Pour interagir avec les ressources de Openstack, vous utiliserez le provider openstack de Terraform.

```
provider "openstack" {
  user_name   = ""
  tenant_name = ""
  password    = ""
  auth_url    = ""
  region      = "microstack"
}
```

Utopios® Tous droits réservés

# 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"
}
```

Utopios® Tous droits réservés

# 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
  }
}
```

Utopios® Tous droits réservés

# 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
}
```

Utopios® Tous droits réservés

# 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

Utopios® Tous droits réservés

# 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

Utopios® Tous droits réservés

# 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

Utopios® Tous droits réservés

# 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
}
```

Utopios® Tous droits réservés

# 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.

Utopios® Tous droits réservés

# 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

}
```

Utopios® Tous droits réservés

# 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 :

Utopios® Tous droits réservés

# 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
    }
```

Utopios® Tous droits réservés

# 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.

Utopios® Tous droits réservés

# 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

Utopios® Tous droits réservés

# Terraform Provisioner pour Openstack

- **Description** : Les provisionneurs Terraform dans openstack sont utilisés pour effectuer des tâches post-déploiement sur des machines virtuelles. Cela est souvent nécessaire pour l'initialisation de systèmes ou l'exécution de scripts de configuration.
- **Exemple d'utilisation** : Pour provisionner une machine virtuelle openstack, on peut utiliser le provisionneur `remote-exec` pour exécuter des commandes ou des scripts sur la VM une fois qu'elle est déployée.

Utopios® Tous droits réservés

# Options de Provisionnement dans openstack

- **Stratégies de provisionnement** : Dans openstack, plusieurs stratégies peuvent être utilisées pour le provisionnement des VMs, comme le démarrage à partir de templates, la personnalisation du système d'exploitation, ou l'exécution de scripts post-déploiement.
- **Personnalisation** : Terraform permet de personnaliser les VMs openstack lors de leur création, notamment en termes de configuration réseau, d'options de sécurité, et de paramètres de système d'exploitation.

---

# Terraform Provisioner

- **local-exec** : Ce provisionneur exécute des commandes sur la machine locale, c'est-à-dire la machine qui exécute Terraform.
- **remote-exec** : Ce provisionneur exécute des commandes sur une machine distante.
- **file** : Ce provisionneur permet de copier des fichiers ou des répertoires d'une machine locale vers une machine distante.

*L'utilisation des provisionneurs est souvent déconseillée, car elle peut entraîner des configurations difficilement reproductibles. Il est généralement préférable d'utiliser des outils de configuration tels que Ansible, Chef ou Puppet pour configurer les machines virtuelles après leur création*

Utopios® Tous droits réservés

# Terraform Provisioner - Scénario Pratique

**Contexte** : Vous êtes un ingénieur DevOps travaillant sur un projet d'infrastructure openstack. Votre tâche consiste à automatiser la configuration des VMs après leur déploiement.

- **Exercice** :
  - Créez une machine virtuelle avec Terraform dans un environnement openstack.
  - Utilisez un provisionneur `remote-exec` pour installer et configurer un serveur web Apache sur la VM.
  - Assurez-vous que la configuration réseau permet une connexion SSH sécurisée à la VM pour l'exécution des scripts.
  - Documentez les étapes et les commandes utilisées dans votre script Terraform pour référence future.

Utopios® Tous droits réservés

# Automatiser son workflow avec Terraform

- Automatiser son workflow avec Terraform signifie utiliser Terraform pour codifier et gérer toutes les étapes de votre processus de déploiement d'infrastructure, afin de minimiser l'intervention manuelle, les erreurs et le temps nécessaire pour déployer et gérer votre infrastructure.

1. **Codifier l'Infrastructure**: Écrivez le code qui décrit l'infrastructure que vous souhaitez créer ou modifier. Cela inclut la définition de toutes les ressources dont vous avez besoin (comme les machines virtuelles, les réseaux, etc.) en utilisant le langage de configuration de Terraform (HCL).
2. **Gestion du Code Source**: Stockez votre code Terraform dans un système de gestion de version de code source (comme Git) pour suivre les modifications, collaborer avec d'autres membres de l'équipe et gérer les versions de votre code
3. **Automatisation des Tests**: Écrivez des tests pour votre code Terraform pour vous assurer qu'il fonctionne comme prévu. Cela peut inclure des tests unitaires pour des morceaux de code individuels, des tests d'intégration pour s'assurer que différentes parties de votre système fonctionnent ensemble, et des tests fonctionnels pour s'assurer que tout le système fonctionne comme prévu.

Utopios® Tous droits réservés

# Automatiser son workflow avec Terraform

4. **Intégration Continue (CI)**: Utilisez un système d'intégration continue pour automatiser l'exécution de vos tests chaque fois que vous apportez des modifications à votre code. Cela vous aidera à identifier et à corriger rapidement les éventuels problèmes
5. **Déploiement Continu (CD)**: Utilisez un système de déploiement continu pour automatiser le déploiement de votre code en production. Cela peut inclure l'automatisation de l'exécution de terraform plan pour afficher les modifications qui seront apportées à votre infrastructure, et terraform apply pour appliquer ces modifications
6. **Gestion de l'État**: Utilisez un backend distant pour stocker l'état de votre infrastructure. Cela permet à plusieurs membres de l'équipe de travailler sur le même projet en même temps et de gérer l'état de manière sécurisée.
7. **Gestion des Variables**: Utilisez des fichiers de variables et des variables d'environnement pour gérer les configurations spécifiques à chaque environnement. Cela permet de réutiliser le même code pour différents environnements (comme le développement, le test et la production) en changeant simplement les valeurs des variables
8. **Modules**: Réutilisez le code en créant des modules pour des morceaux de votre infrastructure que vous utilisez fréquemment. Cela vous permet de ne pas réécrire le même code encore et encore.

Utopios® Tous droits réservés

# Les modules Terraform

- Les modules Terraform sont utilisés pour encapsuler des groupes de ressources dans des blocs réutilisables. Cela permet de créer des morceaux d'infrastructure réutilisables et partageables, d'organiser l'infrastructure en petits morceaux maniables et de mieux gérer l'infrastructure complexe.

1. **Création de Modules**:

- Pour créer un module, vous créez un nouveau répertoire et y placez un fichier de configuration Terraform. Par exemple, si vous avez un ensemble de configurations que vous utilisez fréquemment pour créer des machines virtuelles dans openstack, vous pouvez créer un module pour cela.
- Supposons que vous créiez un module pour une machine virtuelle openstack. Vous aurez un fichier principal (par exemple, main.tf) dans lequel vous définirez les ressources nécessaires pour créer une machine virtuelle.
- Vous définirez également les variables dans un fichier (par exemple, variables.tf) et les valeurs par défaut ou les sorties dans un autre fichier (par exemple, outputs.tf).

Utopios® Tous droits réservés

# Les modules Terraform

2. **Partage de Modules**:

- Une fois que vous avez créé un module, vous pouvez le partager avec d'autres en le mettant dans un registre de modules comme le Terraform Registry ou un dépôt Git.
- Si vous travaillez au sein d'une organisation, vous pouvez également utiliser un registre privé.
- Assurez-vous que le module est bien documenté, de sorte que d'autres sachent comment l'utiliser, quelles sont les variables requises, etc.

3. **Utilisation de Modules**:

- Pour utiliser un module, vous l'invoquez dans votre configuration Terraform avec le bloc module.
- Vous spécifiez la source du module (c'est-à-dire l'emplacement où il est stocké, par exemple, le Terraform Registry, un dépôt Git, etc.) et les valeurs pour les variables que le module requiert.

Utopios® Tous droits réservés

# Les modules Terraform

3. **Utilisation de Modules**:

- Tarraform registry

```
module "virtual_machine" {
  source  = "OpenStack/virtual-machine/openstack"
  version = "2.0.0"
  #...
}
```

- Github

```
...

module "vm" {
  source = "git::https://github.com/your-org/vm-module.git"

  resource_group_name = "my-rg"

}
```

Utopios® Tous droits réservés

# Les modules Terraform - Lab

- **Objectif** :L'objectif de ce TP est de vous apprendre à créer un module Terraform, à le partager sur GitHub et à l'utiliser dans une configuration Terraform pour déployer des ressources dans openstack.
- **Contexte** : Vous travaillez en tant qu'ingénieur DevOps pour une entreprise qui utilise intensivement openstack. On vous a demandé de créer un module Terraform réutilisable pour déployer des machines virtuelles dans openstack. Vous devez ensuite utiliser ce module pour déployer une machine virtuelle dans openstack.
- **Exigences :**

1. Créez un nouveau répertoire pour votre module, par exemple vm-module.
   Le module devra créer une machine virtuelle dans OpenStack.
2. Créez un nouveau dépôt sur GitHub et publiez-y votre module. Assurez-vous d'inclure un fichier README.md qui explique comment utiliser le module, les variables requises, etc.
3. Créez une nouvelle configuration Terraform dans un répertoire séparé.
   Dans cette configuration, utilisez le module que vous avez créé pour déployer une machine virtuelle dans openstack. Configurez les variables nécessaires pour le module.

Utopios® Tous droits réservés

# Gestion des Outputs Terraform pour openstack

- **Introduction** : La gestion des outputs dans Terraform est cruciale pour récupérer des informations essentielles sur les ressources dans des environnements complexes, comme ceux basés sur openstack.

1. **Récupération des Outputs**:
    - Les outputs Terraform permettent de récupérer des valeurs essentielles après l'exécution de la configuration, utiles à la fois dans d'autres parties de votre configuration Terraform et en dehors.
    - Utilisez le bloc `output` pour définir ces valeurs dans votre configuration.

Utopios® Tous droits réservés

# Gestion des Outputs dans Terraform pour Openstack

2. **Utilisation des Outputs dans d'autres modules** :
    - Lors de l'utilisation de modules Terraform, vous pouvez récupérer et exploiter les outputs de ces modules. Par exemple, utiliser l'adresse IP d'une VM openstack dans un autre module.

Utopios® Tous droits réservés

# Gestion des Outputs dans Terraform pour Openstack

3. **Utilisation des Outputs en dehors de Terraform** :
   - Vous pouvez récupérer les outputs hors de Terraform avec la commande `terraform output`.
   - Les outputs sensibles, comme les mots de passe, peuvent être marqués comme tels pour empêcher leur affichage.

Utopios® Tous droits réservés

# Gestion des Outputs Terraform - Exercice Openstack

- **Objectif** : Utiliser Terraform pour déployer une machine virtuelle et un serveur de base de données dans un environnement opensack.
- **Exercice** :
  - Déployez une machine virtuelle opensack avec Terraform. L'output doit être l'adresse IP de la VM.
  - Créez un second module Terraform pour déployer un serveur de base de données sur une VM opensack.
    - Le module doit accepter des variables telles que le nom du serveur, l'emplacement, et les identifiants de l'administrateur.
    - Le module crée la VM et installe/configure le serveur de base de données.
  - Utilisez l'adresse IP de la VM comme input dans le second module pour configurer des éléments tels que les règles de pare-feu du serveur de base de données.

Utopios® Tous droits réservés

# Terraform et les Environnements dans Openstack

- Gérer plusieurs environnements dans Openstack avec Terraform nécessite l'utilisation de configurations, workspaces, variables, et modules pour gérer et déployer des ressources dans des environnements variés tels que développement, production, etc.

1. **Utiliser des Workspaces** :
    - Terraform Workspaces permettent de gérer différents environnements en isolant leurs états. Par exemple, un workspace pour 'dev' et un pour 'prod', chacun avec son propre état.
2. **Utiliser des Variables** :
    - Les variables servent à définir des valeurs spécifiques à chaque environnement. Par exemple, les configurations de ressources Openstack peuvent varier entre les environnements de développement et de production.
3. **Utiliser des Modules** :
    - Les modules Terraform regroupent des configurations réutilisables. Un module commun peut être utilisé dans divers environnements avec des variables spécifiques.

Utopios® Tous droits réservés

# Terraform et les Environnements Openstack

**4. Gestion des États** :

- L'état de Terraform doit être stocké de manière sécurisée, accessible depuis différents environnements. Pour Openstack, vous pouvez utiliser des solutions comme Consul ou des systèmes de fichiers partagés pour stocker l'état Terraform.

```
terraform {
  backend "consul" {
    address = "consul.example.com"
    path    = "terraform/state"
  }
}
```

5. **Contrôle d'Accès** :
    - Le contrôle d'accès est crucial pour gérer qui peut déployer ou modifier les ressources Openstack. Utilisez les contrôles d'accès intégrés de Openstack ou d'autres systèmes de gestion d'identité.
6. **Automatisation** :
    - Automatisez le déploiement de vos configurations Terraform avec des outils de CI/CD comme Jenkins, GitLab CI, ou d'autres pipelines d'intégration continue.

Utopios® Tous droits réservés

# Terraform et les Environnements - Workspaces Openstack

- Terraform offre la possibilité de gérer différents environnements avec des "workspaces".

1. **Créer un Workspace** :
    - Créez un nouveau "workspace" pour chaque environnement (par exemple, développement, test, production).

```
terraform workspace new development
```

2. **Sélectionner un Workspace** :
    - Sélectionnez le "workspace" approprié pour déployer ou modifier les ressources Openstack.

```
terraform workspace select development
```

Utopios® Tous droits réservés

# Terraform et les Environnements - Workspaces Openstack

3. **Lister les Workspaces** :
    - Visualisez tous les "workspaces" existants pour gérer les différents environnements.

```
terraform workspace list
```

4. **Supprimer un Workspace** :
    - Supprimez un "workspace" lorsque son utilisation n'est plus nécessaire.

```
terraform workspace delete development
```

Utopios® Tous droits réservés

# Terraform et les Environnements - Workspaces - Lab Openstack

- **Objectif** : Apprenez à gérer différents environnements comme le développement, le test et la production dans Openstack en utilisant les workspaces de Terraform.
- **Exercice** :
  - Configurez les crédentials Openstack pour Terraform.
  - Configurez le backend de Terraform pour stocker l'état dans une solution compatible avec Openstack.
  - Créez un fichier de variables pour vos configurations.
  - Définissez les ressources Openstack dans un fichier de configuration Terraform.
  - Utilisez les workspaces de Terraform pour créer et gérer différents environnements.

Utopios® Tous droits réservés

# Troubleshooting

- Le dépannage dans Terraform peut impliquer plusieurs étapes et techniques différentes en fonction de la nature du problème.
- **Consultez les journaux et les messages d'erreur** : Les messages d'erreur affichés par Terraform peuvent souvent fournir des informations précieuses sur ce qui ne va pas. Assurez-vous de lire et de comprendre les messages d'erreur.
- **Utilisez terraform plan** : Avant d'appliquer les changements, utilisez toujours la commande terraform plan pour voir quels seront les changements apportés à votre infrastructure.
- **Utilisez la commande terraform show**: Cette commande vous permet de visualiser l'état actuel de votre infrastructure et peut être utile pour identifier les problèmes.
- **Vérifiez l'état de Terraform** : Utilisez la commande terraform state list pour voir toutes les ressources dans l'état actuel de Terraform. Cela peut être utile pour vérifier si une ressource a été créée, modifiée ou détruite.

Utopios® Tous droits réservés

# Troubleshooting

- *__Vérifiez les versions de Terraform et des providers__: Assurez-vous d'utiliser des versions de Terraform et des providers compatibles. Vous pouvez spécifier les versions dans votre fichier de configuration Terraform.
- **Vérifiez les dépendances des ressources** : Assurez-vous que les ressources sont créées dans le bon ordre et que les dépendances sont correctement configurées. Terraform gère généralement les dépendances automatiquement, mais dans certains cas, vous devrez peut-être les spécifier explicitement à l'aide de l'argument depends_on.
- **Vérifiez les permissions et les politiques** : Assurez-vous que le compte ou le service principal utilisé par Terraform a les permissions nécessaires pour créer, modifier ou détruire les ressources.
- **Vérifiez les limites de votre compte** : Certains services cloud ont des limites sur le nombre de ressources que vous pouvez créer. Assurez-vous que vous n'avez pas atteint ces limites.
- **Consultez la documentation** : La documentation de Terraform et des providers peut souvent fournir des informations utiles pour résoudre les problèmes.

Utopios® Tous droits réservés

# Troubleshooting

- **Vérifiez les valeurs des variables** : Assurez-vous que les valeurs des variables que vous avez spécifiées sont correctes. Vous pouvez utiliser la commande terraform console pour vérifier les valeurs des variables et tester les expressions Terraform.
- **Utilisez terraform validate** : Cette commande vérifie que votre fichier de configuration est syntaxiquement correct et valide.
- **Vérifiez les ressources créées manuellement** : Assurez-vous que les ressources que vous gérez avec Terraform n'ont pas été modifiées manuellement ou par un autre outil. Terraform peut avoir du mal à gérer les ressources qui ne sont pas entièrement gérées par lui-même.
- **Utilisez des sorties de débogage** : Vous pouvez obtenir des informations supplémentaires sur l'exécution de Terraform en définissant la variable d'environnement TF_LOG à DEBUG ou TRACE.
- **Utilisez le terraform graph** : Cette commande génère un graphique visuel de vos ressources Terraform et de leurs dépendances. Vous pouvez utiliser un outil comme Graphviz pour visualiser ce graphique.

Utopios® Tous droits réservés