HashiCorp

Terraform

Introduction à Terraform

Infrastructure as Code

Qu’est-ce que cela signifie ?

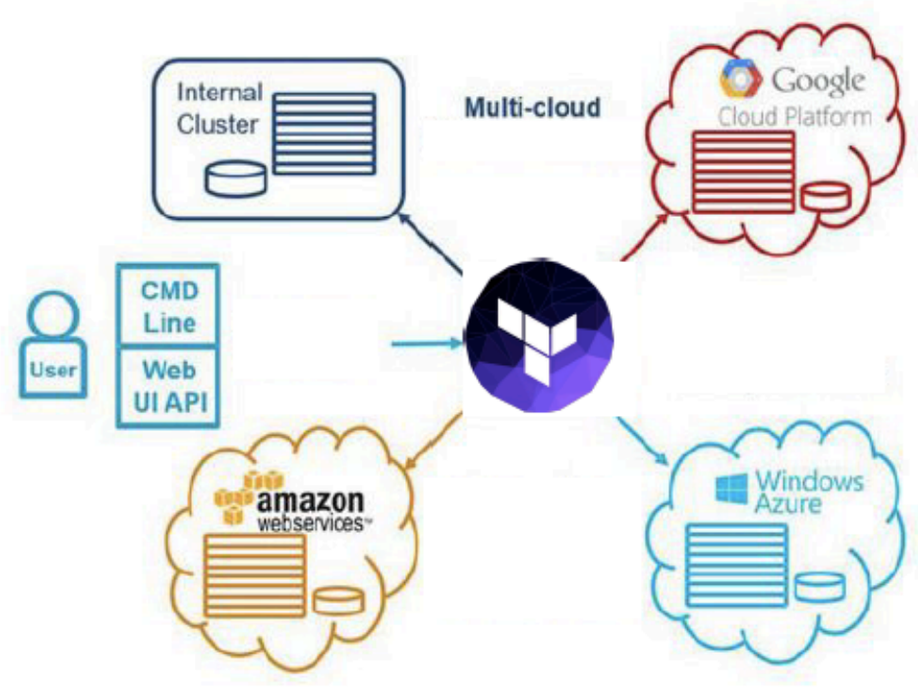
L’Infrastructure as Code désigne la pratique qui consiste à **décrire l’infrastructure** de vos systèmes **par le code.**
vous pouvez également **automatiser** la **création** de votre infrastructure.

Qu’est ce que Terraform ?

Un outil d’orchestration couplé avec un cloud provider peut provisionner des serveurs, des base de données , des loads-balancers et tous autres éléments d’infrastructure.

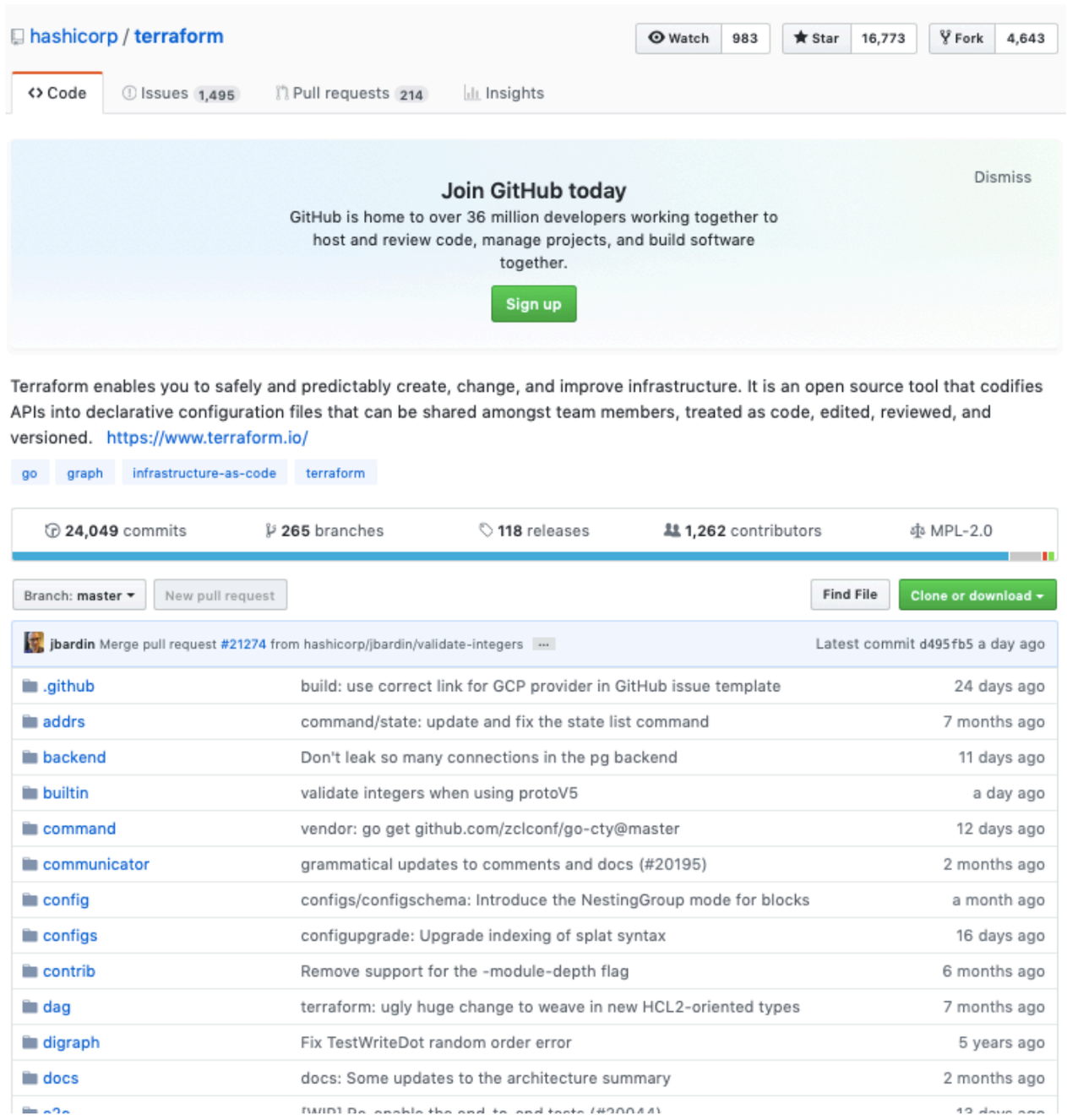
Par exemple dans le cas d’une application :

- Création des VM
- Création des composants réseau
- Déploiements d’un load balancer



UN PROJET OPEN SOURCE :

- Projet Open Source sur Github
- Éditeur : HashiCorp
- Langage : Go
- Premier commit : 22/05/2014
- Contributeurs : 1262 (au 12/05/2019)



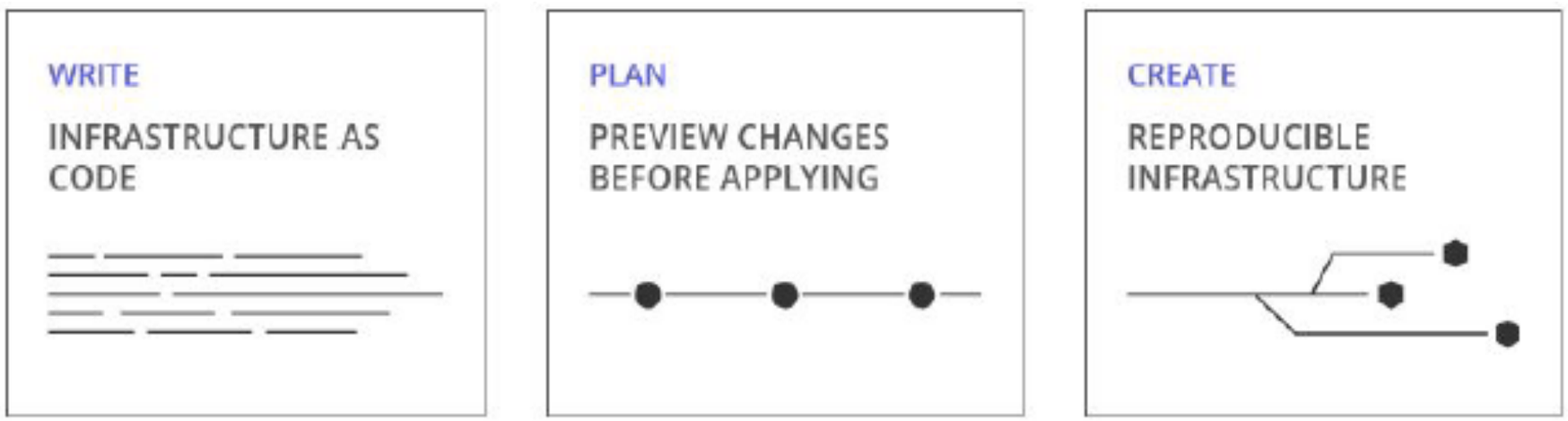
Gestion de l’infrastructure multi-provider

Terrafrom donne la capacité de gérer des resources sur différents providers:

...

ET DE MANIÈRE CONCRÈTE ?

Avant tout un outil en **ligne de commande**
Terraform s'installe sur un **poste de travail** pouvant accéder au service à interroger
Terraform pilote toutes ses ressources:
○ Via des appels d'**API** sur les différents providers
○ Chaque provider est indépendant des autres
Terraform lit les **fichiers de description** contenant les informations de configuration et ordonnance les appels



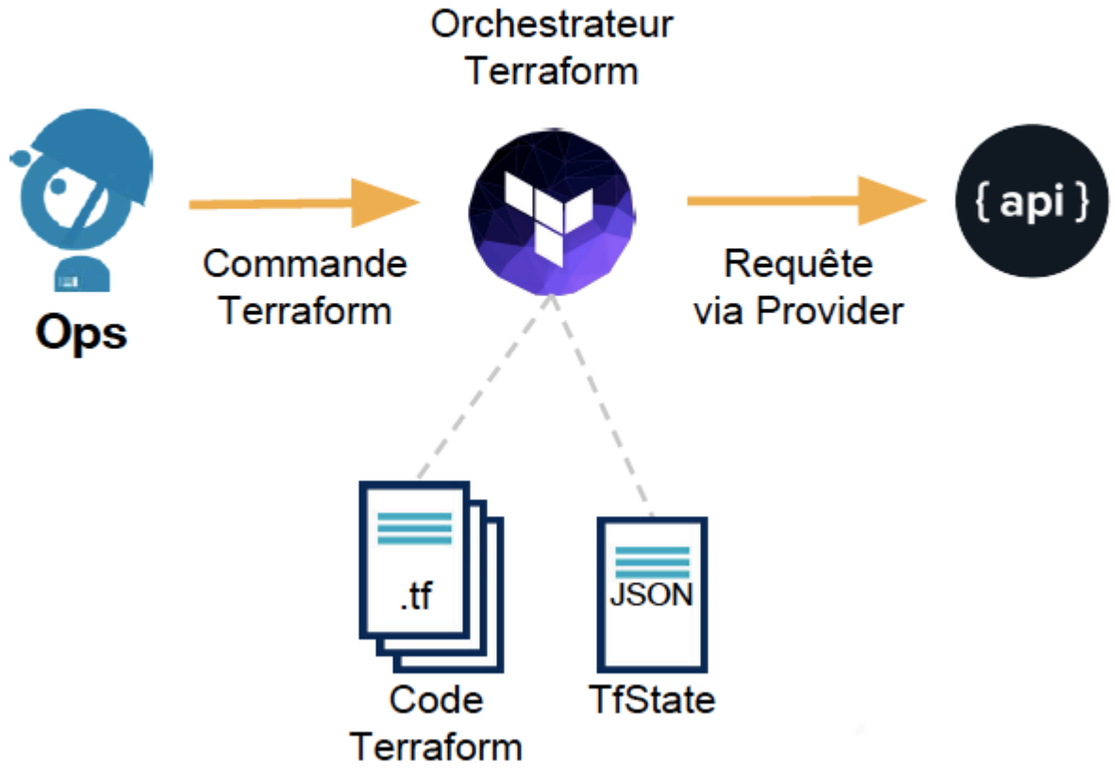
CINÉMATIQUE GÉNÉRALE DE TERRAFORM:

L’Orchestrateur dispose :

- Des flux réseau ouverts pour appeler les APIs
- Des credentials pour se connecter sur les APIs
- Le Binaire Terraform
- Le code Terraform

L’Orchestrateur produit:

- Un Fichier d’état: Le TfState



TERRAFORM, AU SERVICE DE DEVOPS

Un outil 100% en phase avec la démarche Devops

- **Automatisation**
- Creation d’environnements réalisés sans intervention
- Moins d’erreurs liées aux actions manuelles
- **Répétabilité (ou idempotence)**
- Lancer deux fois Terraform sur un même provider avec les mecs fichiers de configuration produit le même résultat
- **Audibilité**
- Capacité de connaitre l’état des resources sur un provider avec la sortie de l’exécution de Terraform
- **Infrastructure as code**
- Description de l’infrastructure sous forme de code.
- **Intégration et déploiement continue**
- Participation au déploiement des applications
- Intégration aux outils classiques d’intégration continue (Jenkins, …)

TERRAFORM, AU SERVICE DE DEVOPS

Pour être bien utilisé, il faut en plus

- Adopter des pratiques issues du développement

  Systématiquement gérer les fichiers de configurations de Terraform dans **un gestionnaire de sources** (Git, SVN…)

- Avoir recours à des pratiques de codage

  Revue de code
  Pair programming
  Écriture de test
  Documentation

- Respecter les principes d’écriture de code Terraform

DEVOPS: UNE DÉFINITION ET DES PRINCIPES

> *DevOps est un modèle de création de valeur. Ce modèle est axé sur le produit et englobe la totalité du cycle de vie logicielle. L'objectif est de minimiser le TTM (Time To Market) à savoir le temps entre l'expression d'une idée et la mise en production des fonctionnalités correspondantes.*

La mise en œuvre DevOps, c'est :

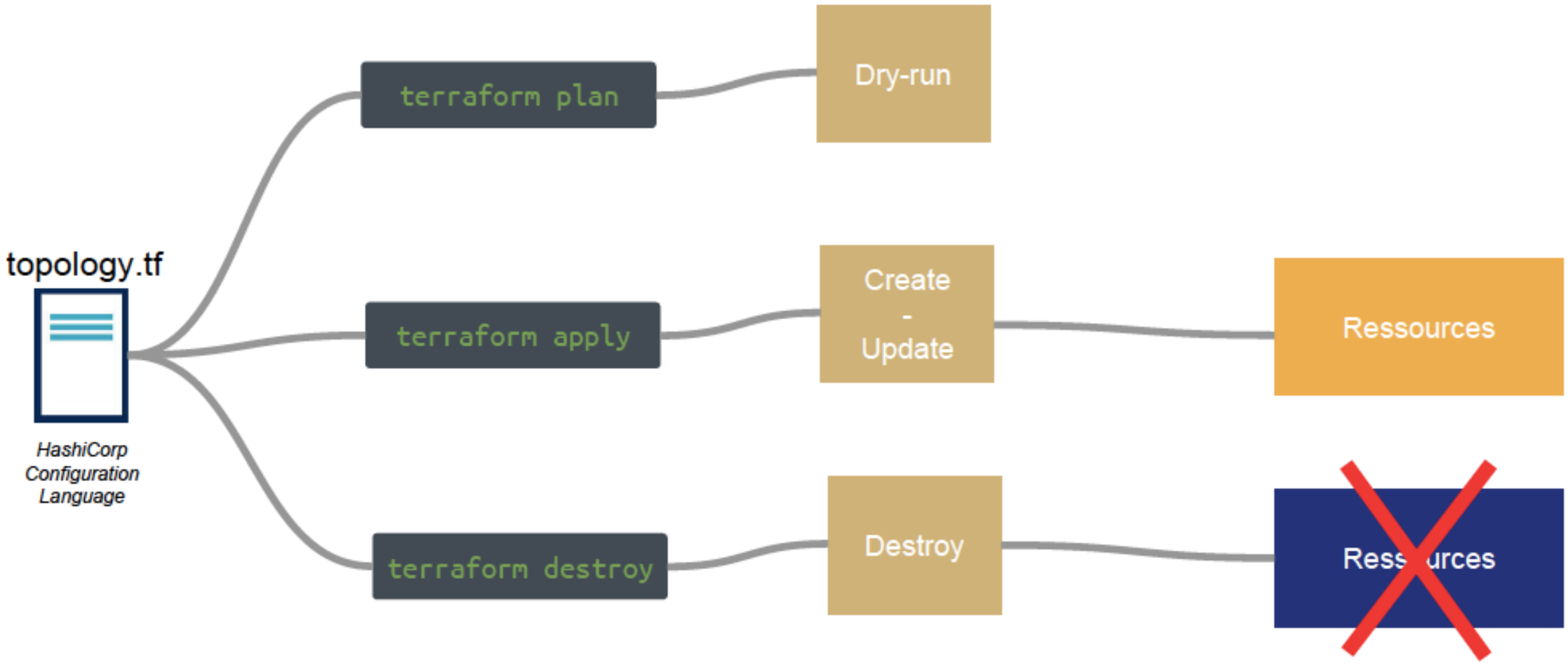
1. une **culture de rapprochement** des Dev et des Ops*
2. **de l'automatisation** : tout doit être automatisé de la création de ressources (serveurs, stockage) jusqu'au monitoring de production
3. **de la mesure** : on doit être en capacité d'avoir des métriques techniques et métiers avec un objectif d'amélioration continue
4. du **partage de connaissance**

* OPS : équipes opérationnelles / d'exploitation

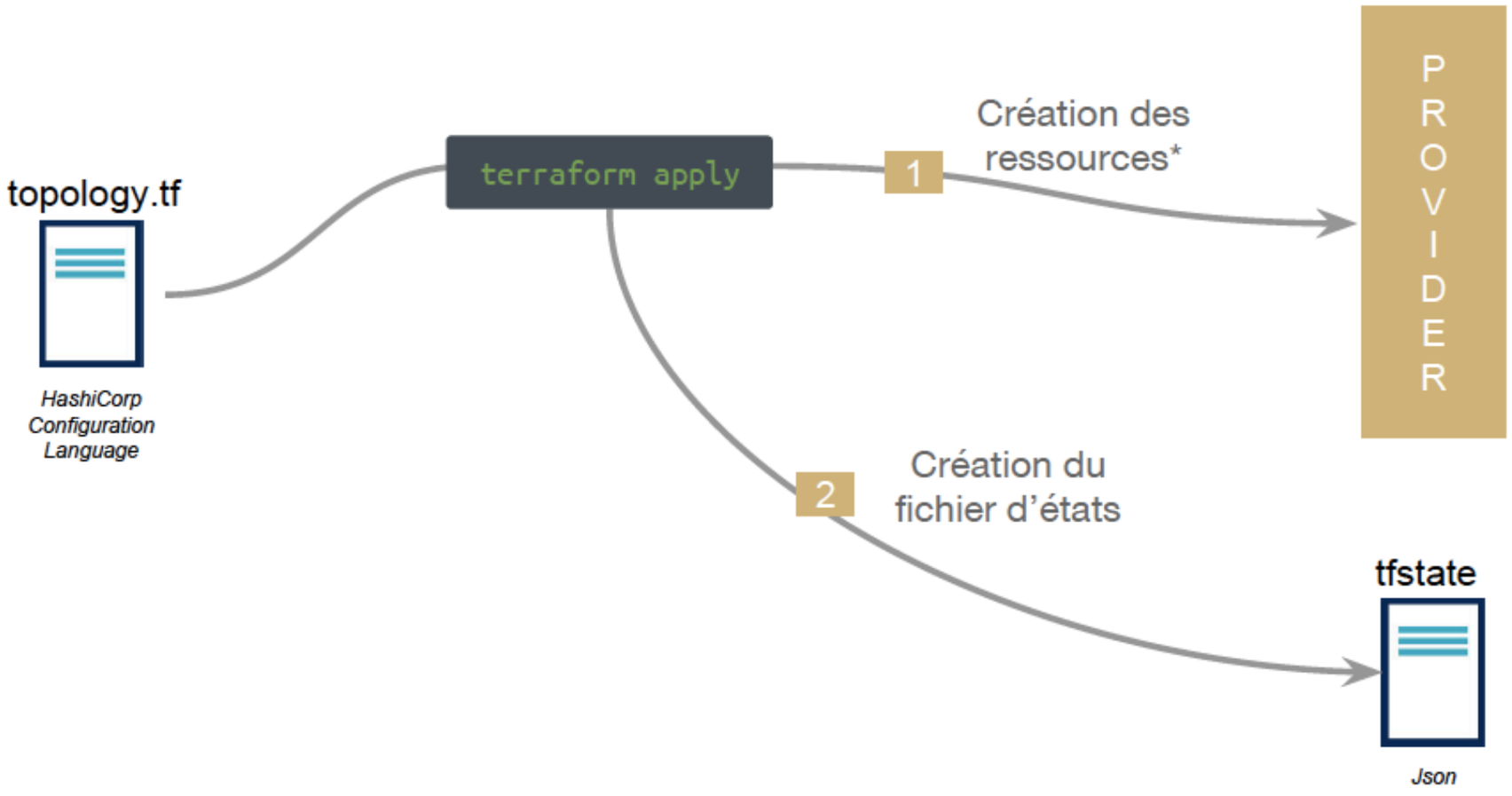
Comment ça marche ?

- Vous exécutez certaines commandes Terraform, telles que terraform apply, pour déployer une infrastructure. La version binaire de terraform analyse votre code, le traduit en une série d'appels d'API aux fournisseurs de cloud spécifiés dans le code et effectue ces appels d'API de la manière la plus efficace possible,comme illustré à la figure.

Comment ça marche ?



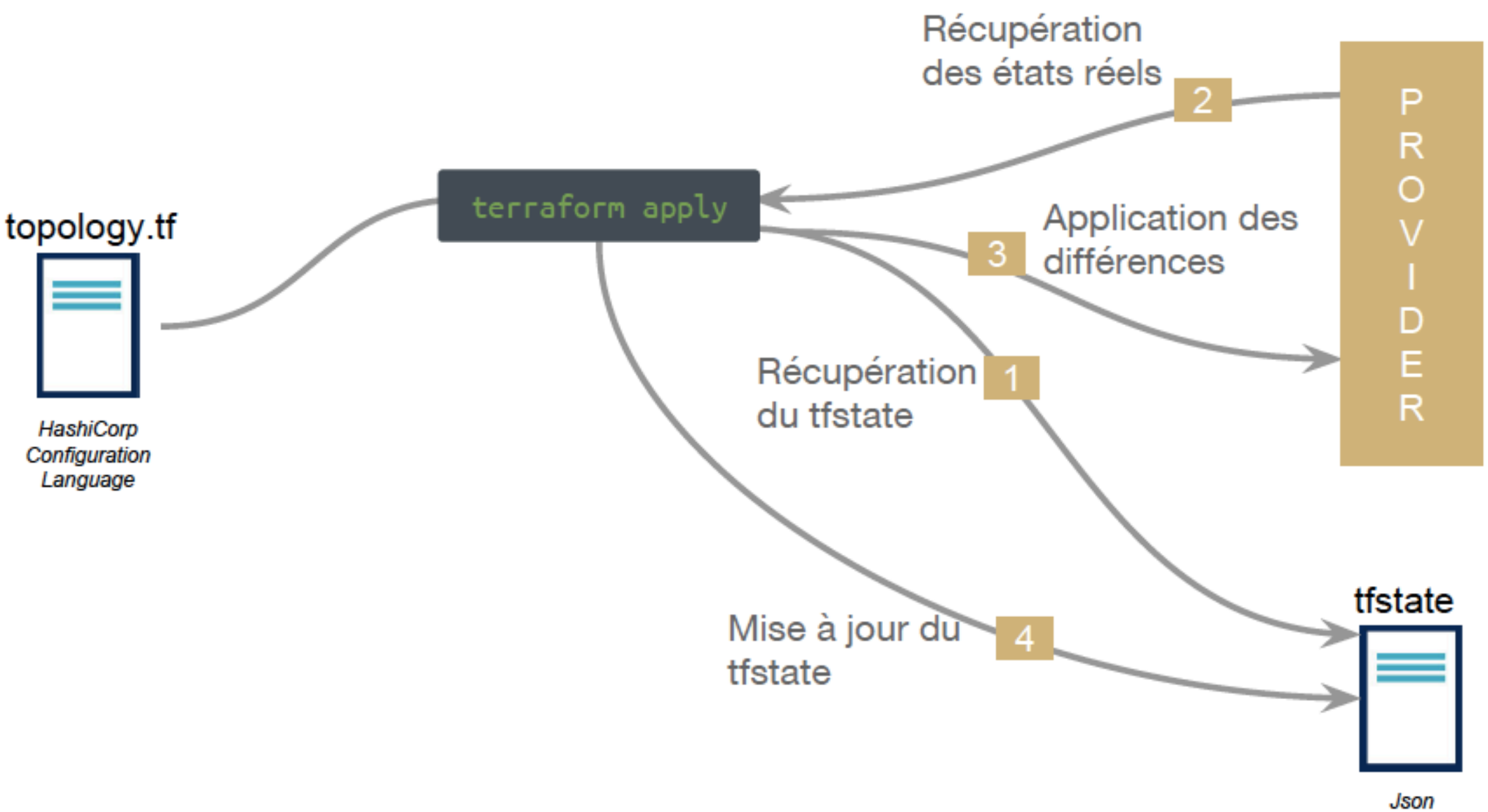
CRÉATION D'UNE INFRASTRUCTURE



Comment ça marche ?

Lorsqu'un membre de votre équipe doit apporter des modifications à l'infrastructure, au lieu de mettre à jour l'infrastructure manuellement et directement sur les serveurs, il modifie les fichiers de configuration Terraform, valide les modifications les modifications puis exécutez la commande terraform apply pour que Terraform effectue les appels API nécessaires au déploiement des modifications.

MISE À JOUR DE L'INFRASTRUCTURE



Les différents fichiers d'un projet Terraform

***.tf :** les fichiers d'extension tf définissent l'infrastructure à générer. La configuration dans son ensemble peut être répartie dans plusieurs fichiers.
Ces derniers seront alors lus par ordre alphabétique et analysés avant d'être appliqués, nous économisant ainsi la gestion des dépendances entre les éléments.

terraform.tfstate : fichier d'état de l'infrastructure. Il est généré lors de l'application des fichiers et est indispensable au bon fonctionnement de l'application. Des informations sensibles y sont présentes.

terraform.tfstate.backup : version n-1 du fichier tsfate terraform.tfvars : fichier de configuration. Si aucun fichier de configuration n'est spécifié, terraforn recherche alors **terraform.tfvars**. Ce fichier peut contenir les identifiants aux différents providers, ce fichier ne doit pas être enregistré dans le gestionnaire de source. Il est conseillé d'utiliser un autre fichier pour les valeurs non sensibles, **variables.tf** par exemple.

gtm_terraform 〉 exemple.tf

Project

gtm_terraform ~/gtm_terraform
- .idea
- .terraform
- exemple.tf
- providers.tf
- terraform.tfvars
- variables.tf

External Libraries

Scratches and Consoles

exemple.tf | terraform.tfvars | providers.tf | variables.tf

```
resource "azurerm_resource_group" "terraform1" {
  name                                = "${var.name_resource_groupe1}"
  location                            = "${var.location_group_resource1}"
}
# virtual network pour la vm jenkins public et le 3 vm avec ip privé dans le meme sous reseau
resource "azurerm_virtual_network" "terraform1" {
  name = "${var.name_virtual_network_terraform1}"
  address_space = ["10.0.0.0/16"]
  location = "${azurerm_resource_group.terraform1.location}"
  resource_group_name = "${azurerm_resource_group.terraform1.name}"
}
#vm au japon tomcat avec adresse public
resource "azurerm_virtual_network" "terraform2" {
  name = "${var.name_virtual_network_terraform2}"
  address_space = ["10.0.0.0/16"]
  location = "Japan East"
  resource_group_name = "${azurerm_resource_group.terraform1.name}"

}

resource "azurerm_subnet" "terraform1" {
  name = "${var.name_subnet}"
  resource_group_name = "${azurerm_resource_group.terraform1.name}"
  virtual_network_name = "${azurerm_virtual_network.terraform1.name}"
  address_prefix = "10.0.2.0/24"
}
#subnet de la vm au japon
resource "azurerm_subnet" "terraform2" {
  name = "${var.name_subnet}"
  resource_group_name = "${azurerm_resource_group.terraform1.name}"
  virtual_network_name = "${azurerm_virtual_network.terraform2.name}"
  address_prefix = "10.0.0.0/24"
}
resource "azurerm_public_ip" "terraform1" {
  name = "${var.name_public_ip}"
  location = "${azurerm_resource_group.terraform1.location}"
  resource_group_name = "${azurerm_resource_group.terraform1.name}"
  public_ip_address_allocation = "dynamic"
  domain_name_label = "vmjenkinspublic"
```

Les variables

• Les variables sont déclarées par l'instruction variable, et peuvent prendre une valeur par défaut et une description.

```
#Common
variable "app_name" {}
variable "env_name" {}
variable "rg_apps_name" {}
variable "rg_infr_name" {}
variable "sa_infr_name" {}
variable "api_url" {
  default = "https://vpodapi.vpod1.carrefour.com"
}
variable "create_dns_record" {
  default = "yes"
}
variable "api_token" {}
variable "subscription_id" {}
variable "client_id" {}
variable "client_secret" {}
variable "tenant_id" {}

#DNS
variable "dns_fqdn_api" {
  default = ""
}

variable "dns_secret" {
  default = ""
}
variable "dns_application_name" {
  default = ""
}
variable "xpod_dns_zone_name" {
  default = ""
}
variable "vpod_dns_zone_name" {
  default = ""
}
variable "linux_host_names" {
  description = "List of the Windows vm names"
  type        = "list"
  default     = []
}
variable "linux_ips" {
  description = "List of the Linux ips"
  type        = "list"
  default     = []
}
variable "windows_host_names" {
  description = "List of the Windows vm names"
  type        = "list"
  default     = []
}
```

Comment fonctionne Terraform?

- Terraform est un outil open source créé par HashiCorp et écrit dans le langage de programmation Go. Le code Go se compile en un seul fichier binaire (ou plutôt un fichier binaire pour chacun des systèmes d'exploitation pris en charge) appelé terraform.

- Vous pouvez utiliser ce fichier binaire pour déployer l'infrastructure à partir de votre ordinateur portable, d'un serveur de génération ou de n'importe quel autre ordinateur.

- Il n'est donc pas nécessaire d'exécuter d'autres infrastructures pour y parvenir. En effet, sous le capot, le binaire terraform effectue des appels d'API en votre nom à un ou plusieurs fournisseurs, tels qu'Amazon Web Services (AWS), Azure, Google Cloud, DigitalOcean, OpenStack, etc. les providers fonctionnent déjà pour leurs serveurs API, ainsi que les mécanismes d'authentification que vous utilisez déjà avec ces providers

```
resource "azurerm_virtual_network" "terraform1" {
  name                                = "${var.name_virtual_network_terraform1}"
  address_space                       = ["10.0.0.0/16"]
  location                            = "${azurerm_resource_group.terraform1.location}"
  resource_group_name                 = "${azurerm_resource_group.terraform1.name}"
}


 resource "azurerm_subnet" "terraform1" {
   name                                = "${var.name_subnet}"
   resource_group_name                 = "${azurerm_resource_group.terraform1.name}"
   virtual_network_name                = "${azurerm_virtual_network.terraform1.name}"
   address_prefix                      = "10.0.2.0/24"
 }


 resource "azurerm_network_security_group" "terraform1" {
   name                                = "${var.name_security_group}"
   location                            = "${azurerm_resource_group.terraform1.location}"
   resource_group_name                 = "${azurerm_resource_group.terraform1.name}"
   security_rule {
     name                              = "ssh"
     priority                          = 1001
     direction                         = "Inbound"
     access                            = "Allow"
     protocol                          = "Tcp"
     source_port_range                 = "*"
     destination_port_range            = "22"
     source_address_prefix             = "*"
     destination_address_prefix        = "*"
   }
   security_rule {
     name                              = "http"
     priority                          = 1002
     direction                         = "Inbound"
     access                            = "Allow"
     protocol                          = "Tcp"
     source_port_range                 = "*"
     destination_port_range            = "80"
     source_address_prefix             = "*"
     destination_address_prefix        = "*"
   }
}
```

Cet extrait de code indique à Terraform de faire des appels d'API à AZURE pour créer un réseau, un sous réseau et un NSG.
En une seule syntaxe simple, Terraform vous permet de déployer des ressources interconnectées entre plusieurs fournisseurs de cloud. Vous pouvez définir l'intégralité de votre infrastructure (serveurs, bases de données, LB, topologie de réseau, etc.) dans les fichiers de configuration Terraform et valider ces fichiers dans le contrôle de version.

Portabilité transparente entre les fournisseurs de services cloud

Comme Terraform prend en charge de nombreux fournisseurs de services cloud, une question fréquemment posée est de savoir s'il prend en charge une portabilité transparente entre eux. Par exemple, si vous utilisiez Terraform pour définir un ensemble de serveurs, bases de données, équilibreurs de charge et autres infrastructures dans AWS, pourriez-vous demander à Terraform de déployer exactement la même infrastructure dans un autre fournisseur de cloud, tel qu'Azure ou Google Cloud?

Portabilité transparente entre les fournisseurs de services cloud

Cette question s'avère être un peu une ligne rouge. En réalité, vous ne pouvez pas déployer «exactement la même infrastructure» dans un autre fournisseur de cloud, car les fournisseurs de cloud n'offrent pas les mêmes types d'infrastructure! Les serveurs, les LB et les bases de données proposés par AWS sont très différents de ceux d’Azure et de Google Cloud en termes de fonctionnalités, de configuration, de gestion, de sécurité, d’évolutivité, de disponibilité, d’observabilité, etc. Il n’existe aucun moyen de traiter ces différences de manière«transparente».

ANSIBLE VS TERRAFORM

- Ansible est un outil de gestion de la configuration
- Utilise un style procédural dans lequel vous écrivez un code qui spécifie, étape par étape, comment atteindre l'état final souhaité
- Peut être utiliser pour créer de l’infrastructure
- Pas d’inventaire et ne tient pas compte de l’infrastructure présente
- Terraform, utilise un style plus déclaratif où vous écrivez du code qui spécifie l'état final souhaité
- Utiliser l’outil le mieux adapter à votre cas d’utilisation

Installation et Configuration

INSTALLATION DE TERRAFORM

- Binaire de la dernière version stable à télécharger ici :
  https://www.terraform.io/downloads.html

- Il s'agit d'un simple binaire à mettre dans son path

- Compatible windows

Azure CLI 2.0

- Azure CLI 2.0 est un outil de ligne de commande offrant une excellente expérience de gestion des ressources Azure. La CLI est conçue pour faciliter la création de scripts, interroger les données, prendre en charge les opérations de longue durée, etc.

Install Azure CLI 2.0 avec Apt

AZ_REPO=$(lsb_release -cs)

- echo "deb [arch=amd64] https://packages.microsoft.com/repos/azurecli/ $AZ_REPO main" l \
- sudo tee /etc/apt/sources.list.d/azure-cli.list
- curl -L https://packages.microsoft.com/keys/microsoft.asc l sudo aptkey add –
- sudo apt-get update sudo apt-get install apt-transport-https azure-cli

az login

Install Azure CLI 2.0 with yum

- sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
- sudo sh -c 'echo -e "[azure-cli]\nname=Azure CLI\nbaseurl=https://packages.microsoft.com/yumrepos/azure-cli\nenabled=1\ngpgcheck=1\ngpgkey=https://packages.microsoft.com/keys/microsoft.asc" > /etc/yum.repos.d/azure-cli.repo’
- sudo yum install azure-cli
- az login

Login Azure CLI

Configurer l'accès Terraform à Azure

- az account show --query “{subscriptionId:id, tenantId:tenantId}“

- Vous pouvez maintenant créer un principal de service à utiliser avec Terraform.
  Utilisez [az ad sp créer-pour-rbac] / cli / azure / ad / sp # azad-sp-créer-pour-rbac) et définir la portée de votre abonnement comme suit:

az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/${SUBSCRIPTION_ID} “

Exécuter un exemple de script

- Créez un fichier test.tf dans un répertoire vide et collez le script suivant:

```
provider "azurerm" {
}
resource "azurerm_resource_group" "rg" {
name = "testResourceGroup"
location = "westus"
}
```

Exécuter un exemple de script

- Enregistrez le fichier, puis initialisez le déploiement Terraform. Cette étape télécharge les modules Azure requis pour créer un groupe de ressources Azure.

terraform init

```
Initializing provider plugins...
- Checking for available provider plugins on https://releases.hashicorp.com.
- Downloading plugin for provider "azurerm" (1.15.0)...

The following providers do not have any version constraints in configuration
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breakin
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.azurerm: version = "~> 1.15"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to se
any changes that are required for your infrastructure. All Terraform command
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, ot
commands will detect it and remind you to do so if necessary.
```

Exécuter un exemple de script

Vous pouvez prévisualiser les actions à exécuter par le script Terraform avec le plan terraform. Lorsque vous êtes prêt à créer le groupe de ressources, appliquez votre plan Terraform comme suit:

```
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.


------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + azurerm_resource_group.rg
      id:       <computed>
      location: "westus"
      name:     "testResourceGroup"
      tags.%:   <computed>


Plan: 1 to add, 0 to change, 0 to destroy.

------------------------------------------------------------------------

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

Exécuter un exemple de script

Terraform apply

```
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + azurerm_resource_group.rg
      id:       <computed>
      location: "westus"
      name:     "testResourceGroup"
      tags.%:   <computed>


Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.
```

Créer une infrastructure de machine virtuelle Linux complète dans Azure avec Terraform

- La section provider indique à Terraform d'utiliser un provider Azure.

```
provider "azurerm" {
subscription_id = "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
client_id = "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
client_secret = "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
tenant_id = "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
}
```

crée un groupe de ressources nommé myResourceGroup dans l'emplacement Eastus:

- La section suivante crée un groupe de ressources nommé myResourceGroup dans l'emplacement Eastus:

```
resource "azurerm_resource_group" "myterraformgroup" {
name                = "myResourceGroup"
location            = "eastus"
tags {
environment         = "Terraform Demo"
}
}
```

Créer un réseau virtuel

```
resource "azurerm_virtual_network" "myterraformnetwork" {
name                                = "myVnet"
address_space                       = ["10.0.0.0/16"]
location                            = "eastus"
resource_group_name          = “${azurerm_resource_group.myterraformgroup.name}“
tags {
environment                         = "Terraform Demo"
}
}
```

crée un sous-réseau nommé mySubnet dans le réseau virtuel myVnet:

```
resource "azurerm_subnet" "myterraformsubnet" {
name                          = "mySubnet"
resource_group_name    ="${azurerm_resource_group.myterraformgroup.name}"
virtual_network_name    ="${azurerm_virtual_network.myterraformnetwork.name}"
address_prefix             = "10.0.2.0/24"
}
```

Créer une adresse IP publique

```
resource "azurerm_public_ip" "myterraformpublicip" {
name                         = "myPublicIP"
location                     = "eastus"
resource_group_name          ="${azurerm_resource_group.myterraformgroup.name}"
public_ip_address_allocation = "dynamic"
tags {
environment = "Terraform Demo"
}
}
```

Créer un groupe de sécurité réseau

```
resource "azurerm_network_security_group" "myterraformnsg" {
name                = "myNetworkSecurityGroup"
location            = "eastus"
resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"
tags {
environment = "Terraform Demo"
}
}
```

```
security_rule {
name                       = "SSH"
priority                   = 1001
direction                  = "Inbound"
access                     = "Allow"
protocol                   = "Tcp"
source_port_range          = "*"
destination_port_range     = "22"
source_address_prefix      = "*"
destination_address_prefix = "*"
}
```

Créer une carte d'interface réseau virtuelle

```
resource "azurerm_network_interface" "myterraformnic" {
name                                = "myNIC"
location                            = "eastus"
resource_group_name                 = “${azurerm_resource_group.myterraformgroup.name} »
ip_configuration {
name                                = "myNicConfiguration"
subnet_id                           = "${azurerm_subnet.myterraformsubnet.id}"
private_ip_address_allocation       = « dynamic"
public_ip_address_id                = "${azurerm_public_ip.myterraformpublicip.id}"
}
tags {
environment                         = "Terraform Demo"
}
}
```

Créer un compte de stockage pour les diagnostics

```
resource "random_id" "randomId" {
keepers = {
# Generate a new ID only when a new resource group is defined
resource_group      = "${azurerm_resource_group.myterraformgroup.name}"
}
byte_length         = 8
}
```

crée un compte de stockage, avec le nom basé sur le texte aléatoire généré à l'étape précédente

```
resource "azurerm_storage_account" "mystorageaccount" {
name                                = "diag${random_id.randomId.hex}"
resource_group_name                 = "${azurerm_resource_group.myterraformgroup.name}"
location                            = "eastus"
account_replication_type            = "LRS"
account_tier                        = "Standard"
tags {
environment                         = "Terraform Demo"
}
}
```

Créer une machine virtuelle

```
resource "azurerm_virtual_machine" "myterraformvm" {
name                          = "myVM"
location                      = "eastus"
resource_group_name           = "${azurerm_resource_group.myterraformgroup.name}"
network_interface_ids         = ["${azurerm_network_interface.myterraformnic.id}"]
vm_size                       = "Standard_DS1_v2"
storage_os_disk {
name                          = "myOsDisk"
caching                       = « ReadWrite"
create_option                 = "FromImage"
managed_disk_type             = "Premium_LRS"
}
```

Créer une machine virtuelle

```
storage_image_reference {
publisher                 = "Canonical"
offer                     = "UbuntuServer"
sku                       = "16.04.0-LTS"
version                   = "latest"
}
os_profile {
computer_nam          = "myvm"
admin_username        = "azureuser"
}
```

Créer une machine virtuelle

```
os_profile_linux_config {
disable_password_authentication      = true
ssh_keys {
path                                 = "/home/azureuser/.ssh/authorized_keys"
key_data                             = "ssh-rsa AAAAB3Nz{snip}hwhqT9h"
}
}
boot_diagnostics {
enabled                              = "true"
storage_uri                          = "${azurerm_storage_account.mystorageaccount.primary_blob_endpoint}"
}
tags {
environment                          = "Terraform Demo"
}
}
```

Build and deploy the infrastructure

terraform plan

```
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

azurerm_resource_group.rg: Refreshing state... (ID: /subscriptions/afc1f4be-a

------------------------------------------------------------------------

No changes. Infrastructure is up-to-date.

This means that Terraform did not detect any differences between your
configuration and real physical resources that exist. As a result, no
actions need to be performed.
```

Appliquez le modèle dans Terraform

Si tout semble correct et que vous êtes prêt à créer l'infrastructure dans Azure, appliquez le modèle dans Terraform:

terraform apply