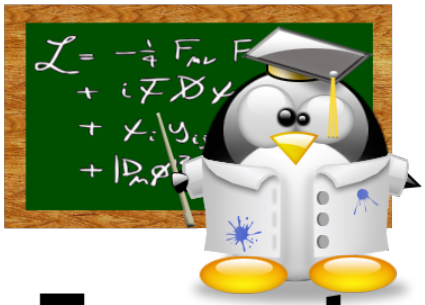

Formatux

Terraform

Infrastructure as Code

Table des matières

Chapitre 1. Introduction

Terraform est un produit de la société **HashiCorp** (*Terraform*, *Vault*, *Consul*, *Packer*, *Vagrant*), permettant de :

- **décrire** dans un langage humainement compréhensible l'infrastructure cible : la **HCL** (**H**ashiCorp **C**onfiguration **L**anguage),
- **versionner** les changements,
- **planifier** le déploiement,
- **créer** l'infrastructure.

Les changements apportés par une modification de code à l'infrastructure sont **prédictifs**.

> If it can be codified, it can be automated.

Le même outil permet de créer des ressources chez les plus grands fournisseurs d'infrastructure :

- AWS,
- GCP,
- Azure,
- OpenStack,
- VMware,
- etc.

L'infrastructure devient reproductible (intégration ⇒ preproduction ⇒ production) et facilement scalable. Les erreurs de manipulations humaines sont réduites.

Un exemple de création de ressouces :

```
# Une machine virtuelle
resource "digitalocean_droplet" "web" {
  name   = "tf-web"
  size   = "512mb"
  image  = "centos-7-5-x86"
  region = "sfo1"
}

/* Un enregistrement DNS
   en IPV4 type "A"
  */
resource "dnsimple_record" "hello" {
  domain = "example.com"
  name   = "test"
  value  = "${digitalocean_droplet.web.ipv4_address}"
  type   = "A"
}
```

Retrouvez plus d'informations sur le site https://www.terraform.io/, dans la documentation https://www.terraform.io/intro/index.html et dans l'espace de formation https://learn.hashicorp.com/terraform/getting-started/install.html.

Chapitre 2. La HCL

Le langage **H**ashiCorp **C**onfiguration **L**anguage est spécifique. Voici quelques points d'attention :

- Les commentaires sur une seule ligne commencent avec un `#`
- Les commentaires sur plusieurs lignes sont encadrés par `/*` et `*/`
- Les variables sont assignées avec la syntaxe `key = value` (aucune importance concernant les espaces). Les valeurs peuvent être des primitives `string`, `number` ou `boolean` ou encore une `list` ou une `map`.
- Les chaînes de caractères sont encadrées par des doubles-quotes.
- Les valeurs booléennes peuvent être `true` ou `false`.

Chapitre 3. Les providers

Terraform est utilisé pour gérer des ressources qui peuvent être des serveurs physiques, des VM, des équipements réseaux, des containeurs, mais aussi pourquoi pas des utilisateurs grafana, etc.

La liste des providers disponibles est consultable ici : https://www.terraform.io/docs/providers/index.html.

En voici quelques-uns :

- AWS,
- Azure,
- VMware vSphere
- OpenStack, CloudStack, OVH, DigitalOcean, etc.
- Gitlab,
- Datadog, PagerDuty,
- MySQL,
- Active Directory
- …

Chaque provider venant avec ses propres ressources, il faut lire la doc !

Chapitre 4. Les actions

- `terraform init`

La première commande à lancer pour une nouvelle configuration qui va initialiser la configuration locale (import de modules par exemple).

La commande `terraform init` va automatiquement télécharger et installer les binaires des providers nécessaires.

- `terraform plan`

La commande `terraform plan` permet d'afficher le plan d'exécution, qui décrit quelles actions Terraform va prendre pour effectuer les changements réels de l'infrastructure.

Si une valeur est affichée comme `<computed>`, cela veut dire que cette valeur ne sera connue qu'au moment de l'exécution du plan.

- `terraform apply`

La commande `terraform apply` va réellement appliquer les changements tels qu'ils ont été décrits par la commande `terraform plan`.

- `terraform show`

La commande `terraform show` permet d'afficher l'état courant de l'infrastructure.

Une fois que l'infrastructure est gérée via Terraform, il est préférable d'éviter de la modifier manuellement.

Terraform va inscrire des données importantes dans un fichier `terraform.tfstate`. Ce fichier va stocker les ID des ressources créées de façon à savoir quelles ressources sont gérées par Terraform, et lesquelles ne le sont pas. Ce fichier doit donc à son tour être conservé et partagé avec toutes les personnes devant intervenir sur la configuration.

Ce fichier `terraform.tfstate` contient des données sensibles. Il doit donc être partagé mais de manière sécurisée (des modules existent), éventuellement ailleurs que dans le repo du code de l'infrastructure.

Des ressources déjà existantes peuvent être importées avec la commande `terraform import ressourcetype.name id_existant`.

- `terraform destroy`

Avec l'avènement du cloud, le cycle de vie d'un serveur et notre façon de consommer les ressources ont considérablement changé. Une VM ou une infrastructure doit tout aussi facilement pouvoir être créée que supprimée.

Avec Terraform, une infrastructure complète peut être déployée juste à l'occasion des tests de non régréssion lors de la création d'une nouvelle version logicielle par exemple et être totalement détruite à la fin de ceux-ci pour réduire les coûts d'infrastructure au plus juste.

La commande `terraform destroy` est similaire à la commande `terraform apply`.

Chapitre 5. Dépendances des ressources

Lorsqu'une ressource dépend d'une autre ressource, la ressource parent peut être appelée comme ceci :

```
# Le VPC de la plateforme
resource "cloudstack_vpc" "default" {
  name         = "our-vpc"
  cidr         = "10.1.0.0/16"
  vpc_offering = "Default VPC offering"
  zone         = "EU-FR-IKDC2-Z4-ADV"
}

# Un réseau en 192.168.1.0/24 attaché à notre VPC
resource "cloudstack_network" "default" {
  name              = "our-network"
  cidr              = "10.1.1.0/24"
  network_offering  = "Isolated VPC tier (100MBps) with SourceNAT and StaticNAT"
  zone              = "EU-FR-IKDC2-Z4-ADV"
  vpc_id            = "${cloudstack_vpc.default.id}"
}
```

Dans l'exemple ci-dessus, un VPC (Virtual Private Cloud) est créé ainsi qu'un réseau privé. Ce réseau privé est rattaché à son VPC en lui fournissant son `id` connu dans terraform en tant que `${cloudstack_vpc.default.id}` où :

- `cloudstack_vpc` est le type de ressource,
- `default` est le nom de la ressource,
- `id` est l'attribut exporté de la ressource.

Les informations de dépendances déterminent l'ordre dans lequel Terraform va créer les ressources.

Chapitre 6. Les provisionners

Les provisionners permettent d'initialiser les instances une fois qu'elles ont été créées (lancer un playbook ansible par exemple).

Afin de mieux comprendre l'utilité d'un provisionner, étudiez l'exemple ci-dessous :

```
resource "aws_instance" "example" {
  ami           = "ami-b374d5a5"
  instance_type = "t2.micro"

  provisioner "local-exec" {
    command = "echo ${aws_instance.example.public_ip} > ip_address.txt"
  }
}
```

Lors de la création de la nouvelle VM, son adresse IP publique est stockée dans un fichier `ip_address.txt`, mais elle aurait très bien pu être envoyée à la CMDB par exemple.

Le provisioner `local-exec` exécute une commande localement, mais il existe de nombreux autres provisionners : https://www.terraform.io/docs/provisioners/index.html.

Chapitre 7. Les variables

Une variable est définie comme suit :

```
# variable de type string
variable "region" {
  default = "us-east-1"
}

# variable de type list
# definition implicite
variable "cidrs" { default = [] }

# definition explicite
variable "cidrs" { type = "list" }
```

Pour ensuite être utilisée comme suit :

```
provider "aws" {
  access_key = "${var.access_key}"
  secret_key = "${var.secret_key}"
  region     = "${var.region}"
}
```

Vous pouvez stocker vos variables dans un fichier externe (par exemple `variables.tf`) sachant que tous fichiers ayant pour extension `.tf` du répertoire courant seront chargés.

L'exemple du chapitre précédent peut être repris pour variabiliser la zone de déploiement de nos ressources :

```
# La zone de deploiement cible
variable "zone" {
  default = "EU-FR-IKDC2-Z4-ADV"
}

# Le VPC de la plateforme
resource "cloudstack_vpc" "default" {
  name         = "our-vpc"
  cidr         = "10.1.0.0/16"
  vpc_offering = "Default VPC offering"
  zone         = "${var.zone}"
}

# Un réseau en 192.168.1.0/24 attaché à notre VPC
resource "cloudstack_network" "default" {
  name              = "our-network"
  cidr              = "10.1.1.0/24"
  network_offering  = "Isolated VPC tier (100MBps) with SourceNAT and StaticNAT"
  zone              = "${var.zone}"
  vpc_id            = "${cloudstack_vpc.default.id}"
}
```

Les variables peuvent être également définies depuis la ligne de commande ou un fichier externe `terraform.tfvars` qui sera chargé automatiquement à l'exécution.

Il est d'usage de stocker les variables critiques (api key, mots de passe, etc.) dans un fichier terraform.tfvars et d'exclure ce fichier de votre configuration git.

Voir la documentation pour plus d'informations sur les variables (Maps, etc.) : https://learn.hashicorp.com/terraform/getting-started/variables

Chapitre 8. TD

Plusieurs possibilités :

- Créer un compte gratuit chez AWS puis :
  - Suivre le module `getting-started` d'hashicorp : https://learn.hashicorp.com/terraform/getting-started/install
  - Créer votre propre infrastructure sur AWS.
- Créer un compte chez Ikoula et gérer une infrastructure CloudStack (https://cloudstack.apache.org/).
- Piloter votre infrastructure VMWare.
- Installer grafana ou un autre logiciel disposant d'un provider sur une de vos VM et utiliser les ressources de ce provider.