Orchestration et Infrastructure As Code avec Terraform

Thierry Boulay, Yanis Mammar

Liminaires

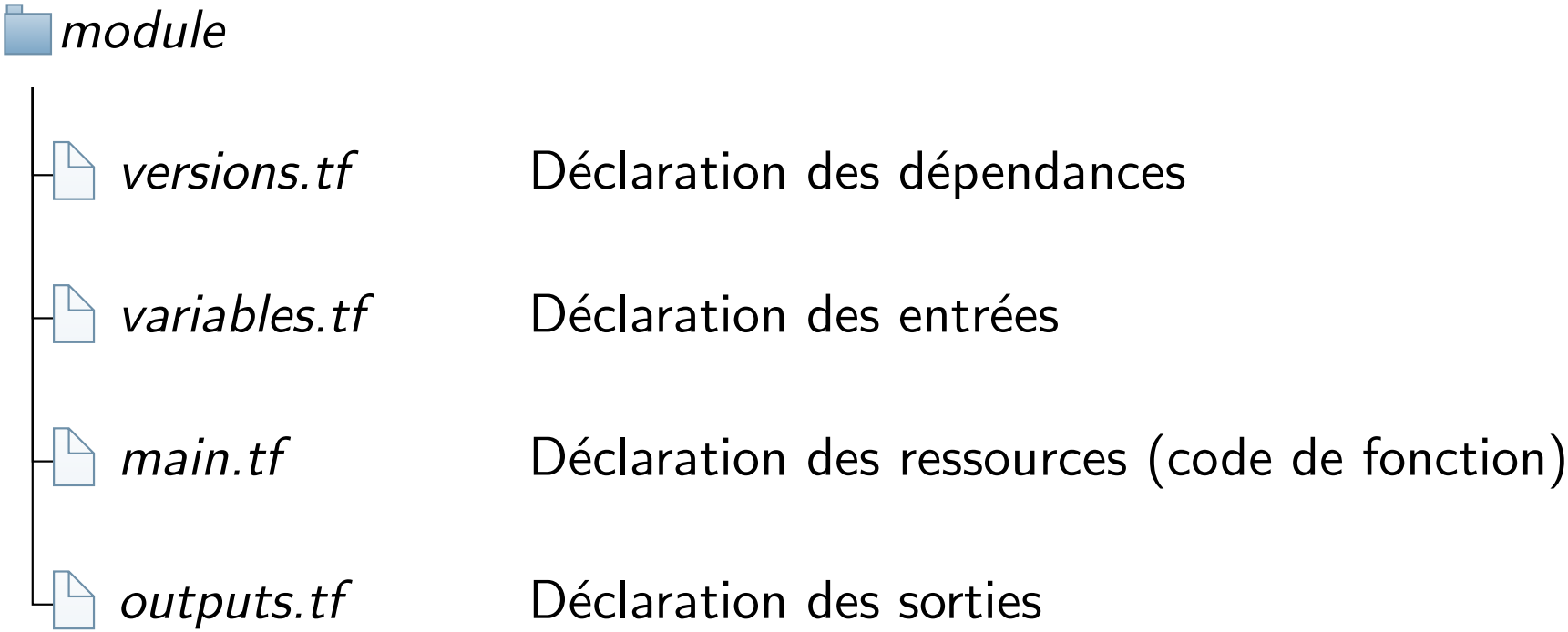
Avec la montée du modèle Cloud s'accompagne le développement de nouveaux outils d'orchestration des infrastructures.

Ces outils permettent d'automatiser la construction des ressources d'une infrastructure: les réseaux, les machines virtuelles, les systèmes de stockage, etc...

Terraform, développé par HashiCorp, supporte plusieurs fournisseurs via des plugins. C'est à dire qu'il permet de définir des topologies pour les principaux fournisseurs de cloud, tels qu'AWS, IBM Cloud, GCP, VMware vSphere ou encore OpenStack.

Structure

Un module Terraform est semblable à une fonction qui peut être paramétrée pour aboutir à des résultats différents.



versions.tf

Par analogie avec les libraires, ce fichier permet de déclarer les dépendances du modules.

```
terraform {
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.35.0"
    }
  }
}
```

Le bloc ci-dessus indique à Terraform que ce module dépend du plugin OpenStack.

variables.tf

Définition des entrées utilisateur acceptées par le module.

```
variable "name" {
  type = string
}

variable "count" {
  type = number
  default = 1
}
```

La variable `count` est optionnel et la variable `name` doit être fournit par l'utilisateur du module.

main.tf

Configuration du provider Openstack.

```
provider "openstack" {
  cloud = "openstack"
  use_octavia = true
}
```

Terraform récupère ici la configuration du provider depuis le fichier *clouds.yaml*.

main.tf

Définition de la topologie et des ressources créées par le module.

```
resource "openstack_compute_instance_v2" "vm" {
  count             = var.count
  name              = "${var.name}-${count.index}"
  image_id          = "ad091b52-742f-469e-8f3c"
  flavor_name       = "m1.small"
  security_groups = ["default"]
  network {
    name            = "private-net"
  }
}
```

Le nombre d'instances créées dépend ici de la variable `var.count`.

outputs.tf

Récupère, traite et expose des informations à l'utilisateur sur les ressources créées.

```
output "vm-id" {
  value = openstack_compute_instance_v2.vm.id
}

output "vm-access-ip" {
  value = openstack_compute_instance_v2.vm.access_ip_v4
}
```

Ces valeurs exportés (callee) peuvent ensuite être utilisées par le module parent (caller).

Exécution

Initialise le répertoire courant et les dépendances du module (plugins, backend, etc..).

```
$ terraform init
```

Génère le plan d'exécution et les actions effectuées par Terraform.

```
$ terraform plan
```

Applique les changements à l'infrastructure.

```
$ terraform apply
```

Affiche l'état des ressources créées par Terraform.

```
$ terraform show
```

Par défaut, Terraform conserve l'état des ressources dans un fichier "*tfstate*" au format JSON. Cet état est rafraîchi avant toute opération pour déterminer les changements à appliquer.

Protection des états

Un *"State Disaster"* survient lorsque le fichier d'état est perdu ou corrompu. Dans ce cas, Terraform ne peut plus déterminer les ressources à sa charge.

Pour prévenir ce problème, il faut privilégier une backend de stockage redondé qui supporte le versionnage (e.g S3 sur AWS, Swift sur OpenStack).

```
# Backend Swift avec un conteneur d'archivage
terraform {
  backend "swift" {
    container         = "terraform-state"
    archive_container = "terraform-state-archive"
  }
}
```

Exercice

Créer un module Terraform selon les descriptions suivantes :

```
variables:
  name: string
  cidr: string
ressources:
  - réseau:
      name: "private-net"
      cidr: var.cidr
  - instance:
      name: var.name
      réseau: "private-net"
sorties:
  - value: IP de l'instance
```