Introduction au DevOps

Terraform

Jolan PHILIPPE

September 11, 2025

Université d'Orléans

Les rôles des outils d'IaC

Management d'application

Customiser, configurer
tester l'application
et la conteneuriser

Provisionnement d'infrastructure

Demander ressources
physiques ou virtuelles;
configurer le réseau;
et règles sécurité

Installation et Configuration

Installer les services
(app + deps)
configurer les services;
et les intégrer

Orchestration de cycle de vie

Upgrades auto;
Backup et recovery;
Surveillance;
Passage à l'échelle

Une infrastructure, c'est quoi ?

Définitions

Infrastructure

L'infrastructure fait référence au logiciel, à la plate-forme ou au matériel qui fournit ou déploie des applications. *Source: Infrastructure-as-code, patterns and practices*

Infrastructure as a Service (IaaS)

L'IaaS (infrastructure as a service) est un service de cloud computing offrant des ressources informatiques matérielles (stockage, réseau, baies de serveurs). *Source: CNIL*

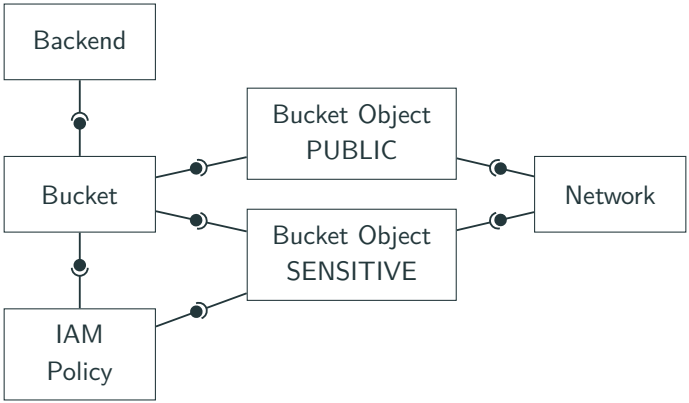
Dans le Cloud: Infrastructure ⇒ APIs

Dans un contexte de IaaS, le paradigme du Cloud computing a transformé l'infrastructure en APIs externes (e.g., REST) sur lesquelles on peut envoyer des requêtes.

Aujourd'hui, dans le Cloud, les infrastructures sont comme n'importe quel logiciel offrant des services.

- Une requête pour un bucket pour stocker du contenu (e.g., BDD)
- Une requête pour un cluster GKE (Google Kubernetes Engine) pour acquerir des applciations micro-services
- Une requête pour des machines virtuelles, sur lesquelles on installera une application ou des logiciels
- etc.

Exemple


Backend
Bucket
Bucket Object
PUBLIC
Bucket Object
SENSITIVE
IAM
Policy
Network

Provisionner une infrastructure

Définition

Le provisionnement est l'opération de création et de gestion d'un ensemble de ressources dans une infrastructure dans un Cloud public, un data center, ou solution de monitoring hébergée.

Concrètement, le provisionnement c'est demander des machines et des ressources.

Provisionner une infrastructure

Définition

Le provisionnement est l'opération de création et de gestion d'un ensemble de ressources dans une infrastructure dans un Cloud public, un data center, ou solution de monitoring hébergée.

Concrètement, le provisionnement c'est demander des machines et des ressources.

Ressources

Dans le provisionnement, TOUT est une ressource.

- Une machine virtuelle
- Un stockage (e.g., BDD, services de stockage, etc.)
- Un réseau
- Un secret manager
- etc.

Provisionner une infrastructure

Les outils de provisionnement ont été fait pour rendre plus réutilisable, reproductible, flexible et sûr la gestion d'infrastructure dans le Cloud.

Solution spécifique

- Heat pour OpenStack
- Cloud Formation (CFN) pour Amazon Web Service (AWS)
- Azure Resource Manager (ARM) pour Microsoft Azure
- GCP pour Google Cloud (déprécié, Google utilise maintenant Terraform)

Solution multi-Cloud

- Terraform
- Pulumi

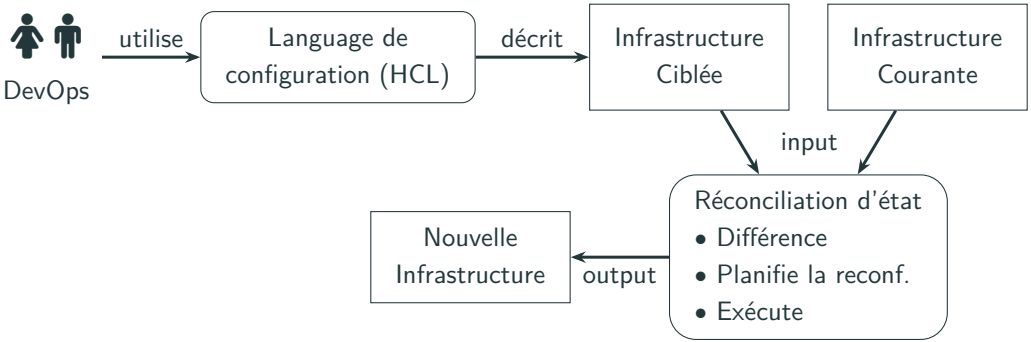
Terraform

Dans ce module, nous utiliserons un outil de provisionnement: Terraform

Terraform est un outil

- Déclaratif: l'état souhaité est décrit dans un fichier, en utilisant le langage HCL (HashiCorp Configuration Language).
- Stateful: l'état courant de l'infrastructure est conservé dans un fichier.
- De reconciliation: l'état souhaité est comparé avec l'état courant pour définir quelle opérations (CRUD) doivent être réalisées. Ces actions sont ensuite exécutées.
- Conccurent: les opérations de réconcialtion peuvent être exécutée en parallel, tant que les dépendences sont respectées.

Boucle de reconciliation



Terraform

Vue d'ensemble de Terraform

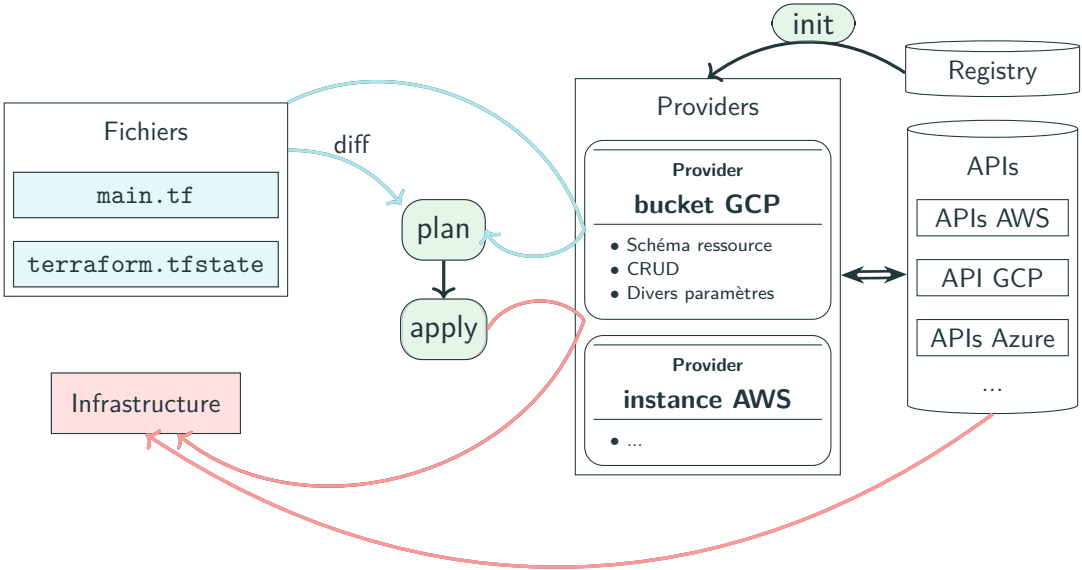
Les différents éléments de Terraform

- CLI (Command Line Interface)
- Le langage déclaratif HCL (HashiCorp Configuration Language)
- Les fichiers de configuration (`.tf`)
- Les providers
- Le state (fichier d'état)

Les rôles de Terraform

- Provisionner des ressources (VM, réseaux, bases, etc.)
- Gérer le cycle de vie de l'infrastructure (create, update, delete)
- Maintenir l'infrastructure en cohérence avec le code
- Fournir un plan d'exécution avant application
- Supporter de multiples fournisseurs cloud

Vue d'ensemble de Terraform



CLI

```
terraform init
```

Initialise le repertoire de travail, et télécharge les providers.

CLI

`terraform init`

Initialise le repertoire de travail, et télécharge les providers.

`terraform plan`

Produit un plan d'exécution pour reconcilier l'infrastructure avec le fichier de configuration (**.tf**). Le plan consiste en un ensemble de create/delete/update/replace.

CLI

`terraform init`

Initialise le repertoire de travail, et télécharge les providers.

`terraform plan`

Produit un plan d'exécution pour reconcilier l'infrastructure avec le fichier de configuration (**.tf**). Le plan consiste en un ensemble de create/delete/update/replace.

`terraform apply`

Produit un plan et l'exécute.

CLI

`terraform init`

Initialise le repertoire de travail, et télécharge les providers.

`terraform plan`

Produit un plan d'exécution pour reconcilier l'infrastructure avec le fichier de configuration (**.tf**). Le plan consiste en un ensemble de create/delete/update/replace.

`terraform apply`

Produit un plan et l'exécute.

`terraform destroy`

Supprime toutes les ressources.

Ressources et Providers

Ressource

- Une “**Resource**” est l’élément de base dans Terraform.
- Elle représente un objet de l’infrastructure (ex: VM, base de données, VPC).
- Elle est définie par :
  - un **nom** (identifiant local dans Terraform),
  - un **type** (ex: `aws_instance`, `google_storage_bucket`),
  - des **attributs/valeurs** (paramètres de configuration).

Ressources et Providers

Provider

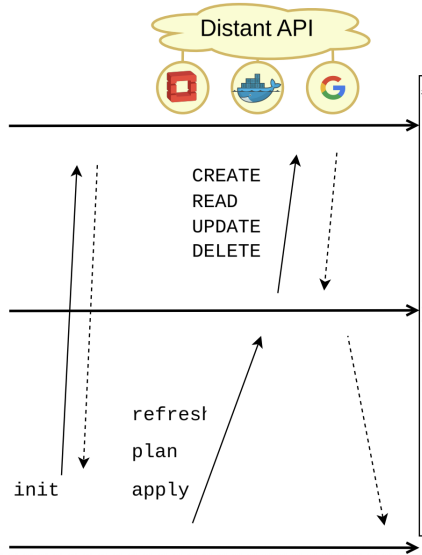
- Chaque resource est implémentée dans un **Provider** (fichier binaire écrit en Go).
- Le provider contient :
  - le **schéma** des ressources (attributs requis et optionnels; ainsi que leurs types),
  - les opérations **CRUD** (*Create, Read, Update, Delete*),
  - divers paramètres et règles de validation.
- Exemple : la resource `aws_instance` est définie dans le provider AWS.

Terraform

User Configuration

```
# vm.tf (HCL format)

terraform {
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "5.6.0"
    }
  }
}
provider "google" {
  project = var.gcp_project_id
  region  = var.gcp_region
  credentials = file(var.gcp_key)
}
resource "google_compute_instance" "vm"
{
  name         = "redis"
  machine_type = "e2-medium"
  network_interface {
    network = "default"
  }
}
```



Terraform Managed State

```
# terraform.tfstate (JSON format)

resource "google_compute_instance"
"vm" {
  name         = "redis"
  id           = "projects/tf-80/..."
  instance_id = "3012364625718931000"
  network_interface {
      name        = "nic0"
      network     =
"https://www.googleapis.com/compute"
      network_ip = "10.162.0.20"
}
```

Ecrire une configuration en HCL

HCL: un langage de configuration déclaratif

Définition en blocs

En HCL, on définie des blocs, contenants des attributs sous la forme de clé-valeur. Un bloc peut lui même contenir un bloc. On parle alors de blocs imbriqués.

HCL: un langage de configuration déclaratif

Définition en blocs

En HCL, on définie des blocs, contenants des attributs sous la forme de clé-valeur. Un bloc peut lui même contenir un bloc. On parle alors de blocs imbriqués.

Le bloc `terraform`

Définit les paramètres globaux du projet Terraform : version requise, configuration du backend pour le *state*, et gestion des providers.

HCL: un langage de configuration déclaratif

Définition en blocs

En HCL, on définie des blocs, contenants des attributs sous la forme de clé-valeur. Un bloc peut lui même contenir un bloc. On parle alors de blocs imbriqués.

Le bloc `terraform`

Définit les paramètres globaux du projet Terraform : version requise, configuration du backend pour le *state*, et gestion des providers.

Le bloc `provider`

Configure l'accès à un service cloud ou une API : informations de connexion, région, options spécifiques au fournisseur. Ce bloc est nommé.

HCL: un langage de configuration déclaratif

Définition en blocs

En HCL, on définie des blocs, contenants des attributs sous la forme de clé-valeur. Un bloc peut lui même contenir un bloc. On parle alors de blocs imbriqués.

Le bloc `terraform`

Définit les paramètres globaux du projet Terraform : version requise, configuration du backend pour le *state*, et gestion des providers.

Le bloc `provider`

Configure l'accès à un service cloud ou une API : informations de connexion, région, options spécifiques au fournisseur. Ce bloc est nommé.

Le bloc `resource`

Décrit une ressource de l'infrastructure (VM, réseau, bucket...). Contient un type, un nom et des attributs configurables.

HCL: un langage de configuration déclaratif

Attributs / Arguments / Champs

Les attributs en HCL sont définis avec le signe égal (=).

La valeur des attributs peut être n'importe quelle expression: constante, appel de fonction, une liste, un objet, un référence, etc.

- `name = "mon serveur"`
- `credentials = file("./creds.json")`
- `labels = {app = "redis"}`
- `image = docker_image.redis.name`
- etc

Les définitions multiples d'un attribut sont interdites. On ne fait qu'une seule attribution.

HCL: un langage de configuration déclaratif

```
terraform {
    required_providers {
        google = {
            source  = "hashicorp/google"
            version = "5.6.0"
        }
    }
}
provider "google" {
    project = "mon_projet"
    region = "europe-west1"
    credentials = file(var.gcp_key)
}
resource "google_compute_instance" "vm" {
    name = "redis-${provider::google.project}"
    machine_type = "e2-medium"
    network_interface {
        network = "default"
    }
}
```

Bloc `terraform`

- Le bloc **`terraform`** définit des paramètres globaux.
- Ici, on indique que le provider utilisé est `google`.
- Le provider est téléchargé depuis le registry `hashicorp`.
- La version requise est fixée à `5.6.0`.

HCL: un langage de configuration déclaratif

```
terraform {
    required_providers {
        google = {
            source  = "hashicorp/google"
            version = "5.6.0"
        }
    }
}
provider "google" {
    project = "mon_projet"
    region = "europe-west1"
    credentials = file(var.gcp_key)
}
resource "google_compute_instance" "vm" {
    name = "redis-${provider::google.project}"
    machine_type = "e2-medium"
    network_interface {
        network = "default"
    }
}
```

Bloc `provider`

- `project = "mon_projet"` : identifiant du projet
- `region = "europe-west1"` : région utilisée pour déployer les ressources.
- `credentials = file(gcp_key.json)` : chemin vers la clé de service GCP

HCL: un langage de configuration déclaratif

```
terraform {
    required_providers {
        google = {
            source  = "hashicorp/google"
            version = "5.6.0"
        }
    }
}
provider "google" {
    project = "mon_projet"
    region = "europe-west1"
    credentials = file(var.gcp_key)
}
resource "google_compute_instance" "vm" {
    name = "redis-${provider::google.project}"
    machine_type = "e2-medium"
    network_interface {
        network = "default"
    }
}
```

Bloc `resource`

- `resource "..."""vm"` : définit une machine virtuelle avec l'id `vm`.
- `name = "redis-${provider::google.project}"` : le nom de la VM sera construit à partir de *redis-* suivi du nom de projet dans le provider.
- `machine_type = "e2-medium"` spécifie le type de machine.
- `network_interface { network = "default"}` rattache l'instance au réseau par défaut.

HCL: un langage de configuration déclaratif

Quelques autres blocs

Le bloc `data`

Permet de récupérer des informations existantes depuis un provider. Le contenu est en lecture-seule.

Le bloc `module`

Regroupe plusieurs ressources et variables dans une unité réutilisable. Facilite la factorisation et la réutilisation du code Terraform. Un fichier **.tf** est lui même un module.

Le bloc `check`

Permet de définir des conditions de validation dans la configuration. Terraform échoue si la condition n'est pas respectée ⇒ utile pour garantir des contraintes métiers.

Le bloc `import`

Relie une ressource réelle déjà existante dans le cloud à une ressource définie dans le code Terraform. Important : permet d'intégrer de l'infra existante sans la recréer.

HCL: un langage de configuration déclaratif

Variables

Les modules peuvent avoir trois types de variables:

- Inputs
- Outputs
- Locals

Ces variables sont définies avec les blocs: `variable`, `output` et `locals`.

HCL: un langage de configuration déclaratif

Références

- Les attributs d'une ressource sont référencées par le type et le nom de la ressource, suivi du nom de la ressource. Par exemple: `docker_image.redis.image_id`
- Pour référencer depuis un bloc **`data`**, on utilise le mot clé **`data`**. Par exemple: **`data`**`.docker_image.redis.image_id`
- Pour les variables de type *input* et *local*, on utilse les mots clés `var` et `local`. Par exemple:
  `var.my_input_var`
  `local.my_local_var`

HCL: un langage de configuration déclaratif

```
terraform {
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = "5.6.0"
    }
  }
}

provider "google" {
  project = "my-gcp-project-id"
  region  = "europe-west1"
  credentials = file(gcp_key.json)
}

data "google_compute_image" "debian" {
  family  = "debian-11"
  project = "debian-cloud"
}

locals {
  prefix = "app"
}
```

```
resource "google_compute_instance" "redis" {
  name = "redis-01"
  machine_type = "e2-micro"
  boot_disk {
    initialize_params {
      image = data.google_compute_image.debian.
          self_link
    }
  }
  network_interface {
    network = "default"
    access_config {}
  }
}
resource "google_compute_instance" "app" {
  name = "${local.prefix}-${
      google_compute_instance.redis.name}"
  machine_type = "e2-medium"
  boot_disk {
    initialize_params {
      image = data.google_compute_image.debian.
          self_link
    }
```

Quelques bonnes pratiques

Convention de nommage

Un code d'IaC est aussi source de documentation.

Bonne pratique 1

- Donner toujours une description de la ressource au sein de son nom
  - Environnement
  - Type de la ressource
  - But, usage, intention
  - Exemple: `dev-firewall-rule-allow-helloworld-to-database`

Variables et constantes

Bonne pratique 2

- De manière générale: Privilégiez les constantes plutôt que les variables
- Si le changement d'une valeur affecte les dépendances de l'infrastructure ou compromets des informations sensibles, utilisez une variable

Avoid troubleshooting

Bonne pratique 3

- Bien lire et comprendre chaque déclarations et le plan avant de faire un `terraform apply`
- Versionner le code Terraform, sans commit de secrets
- Intégrer du CI/CD sur votre infrastructure Terraform
- Stocker vos fichiers d'état (.tfstate) sur des stockages distants avec des mecanismes de lock