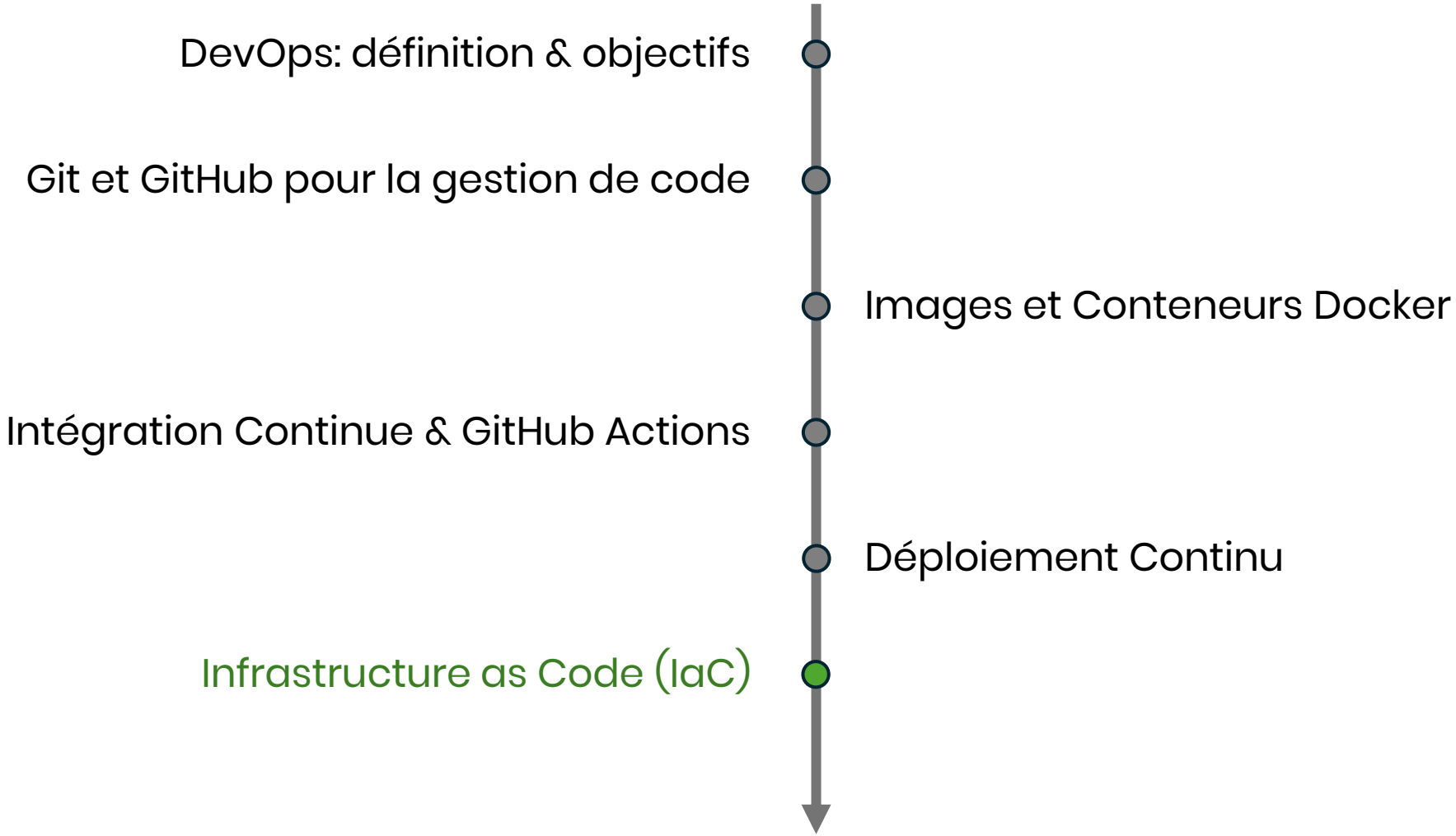
Projet Tutoré & DevOps

Hiba Najjar

S7, GInf4

UPF Université Privée de Fès
الجامعة الخاصة لفاس
Private University of Fez

On a vu jusqu'à aujourd'hui



Infrastructure: avant/après

avant IaC

Aministrateur **A** -> console AWS -> crée manuellement:
- 3 serveurs EC2
- 1 load balancer
- 1 base de données

2-3 heures

Après 2 mois:
Administrateur **B** doit créer un environnement de test similaire :
- Se rappeler de la procédure
- Fait quelques erreurs

2-3 heures

Après 6 mois:
Administrateur **A** part en vacances, quelque chose casse :
- Personne ne sait exactement comment l'infrastructure était configurée
- Difficile de corriger le problème rapidement

après IaC

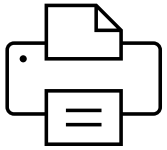
Aministrateur **A**
- Écrit un fichier de code
- Exécute le fichier
- L'infrastructure est créée

5-10 minutes

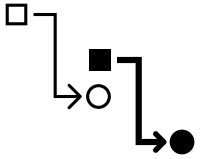
Après 2 mois:
Administrateur **B** utiliser le même code pour créer un environnement de test identique

Après 6 mois:
Administrateur **A** part en vacances, quelque chose casse :
- Le code documente la configuration de l'infrastructure
- Les versions sont sauvegardées sur Git.

Avantages de l'IaC

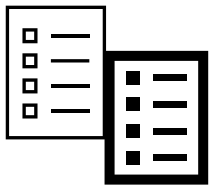


Reproductible
Recréer l'infrastructure identique



Documentée
Le code est la documentation

Versionnable
Suivre tous les changements dans Git



Collaborative
N'importe qui peut recréer l'infrastructure

Cloud Computing

serveurs physiques → ressources informatiques hébergées dans des centres de données distants

- Caractéristiques
  - Accès à la demande
  - Accès réseau large bande
  - Élasticité Rapide
  - Service & paiement mesuré
- Avantages
  - Économiques
  - Techniques
  - Organisationnels
- Défies
  - Sécurité & Confidentialité
  - Dépendance fournisseur
  - Latence: distance & internet
  - Coûts des mauvaises pratiques
- Fournisseurs
  - AWS
  - Microsoft Azure
  - GCP
  - IBM Cloud
  - Oracle Cloud
  - Alibaba Cloud

Régions aws

AWS n'a pas de centre de données physiquement au Maroc

Choisir la région parente la plus proche
eu-west-3 (Paris), la plus proche géographiquement.

Utiliser les Wavelength Zones
une zone Wavelength est disponible via un opérateur partenaire: Orange à Casablanca -> y déployer des ressources tout en restant rattaché à une région parente.

- ✓ les données n'ont pas besoin de traverser tout Internet pour atteindre une région lointaine ; elles s'arrêtent au centre de données de ton opérateur mobile local
- ✓ déploiement des services AWS directement au bord des réseaux 5G des opérateurs télécoms.
- ✓ idéales pour des applications qui ne supportent aucun délai (chirurgie à distance, voitures autonomes, streaming 4K)

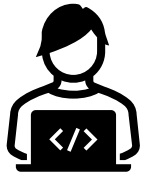
https://aws.amazon.com/fr/about-aws/whats-new/2025/01/aws-wavelength-zone-casablanca/

IaaS: Infrastructure as a Service

Amazon EC2, Amazon S3
Microsoft Azure VMs
Google Compute Engine

l'infrastructure informatique de base (serveurs virtuels, stockage, réseaux)

système d'exploitation et applications



- Contrôle maximal sur l'infrastructure
- Évolutivité selon le besoin
- Flexibilité dans la configuration

Autres:

Platform as a Service (PaaS)

- Gestion automatisée de l'infrastructure
- Outils intégrés de développement
- Concentration sur le développement

E.g.: Google App Engine, Microsoft Azure App Service

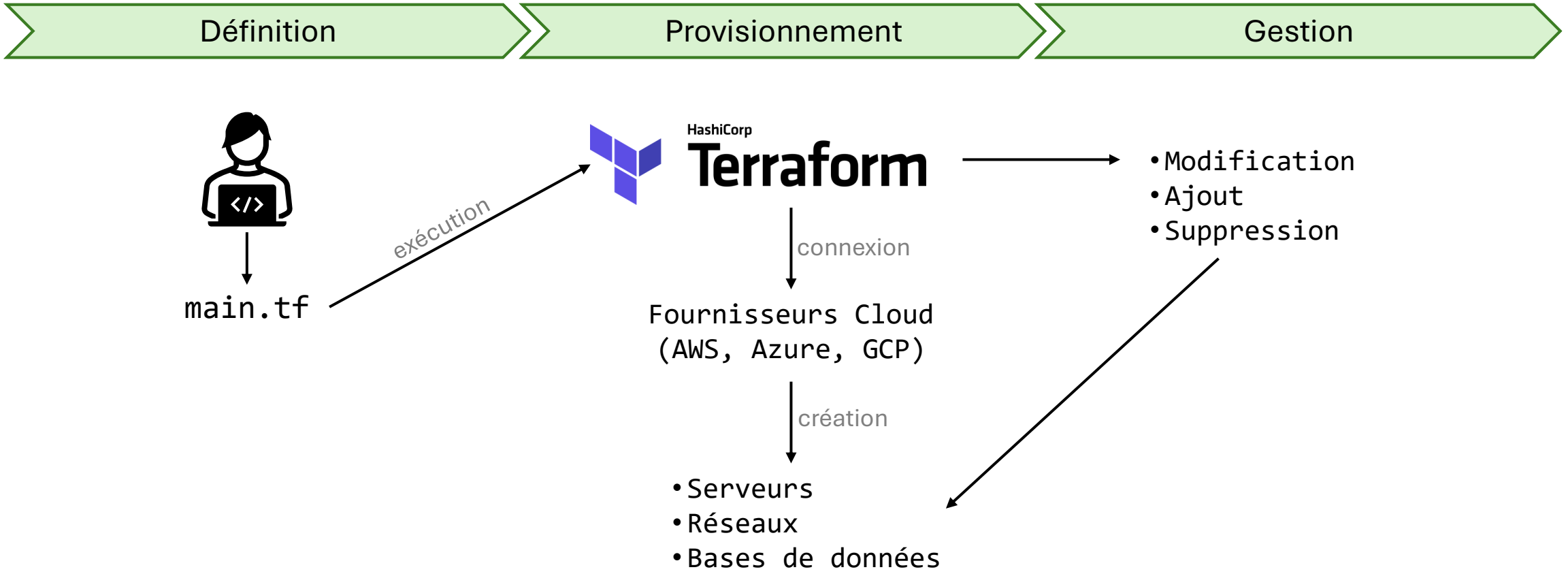
Software as a Service (SaaS)

- Aucune installation nécessaire
- Mises à jour automatiques
- Accessibilité depuis n'importe où

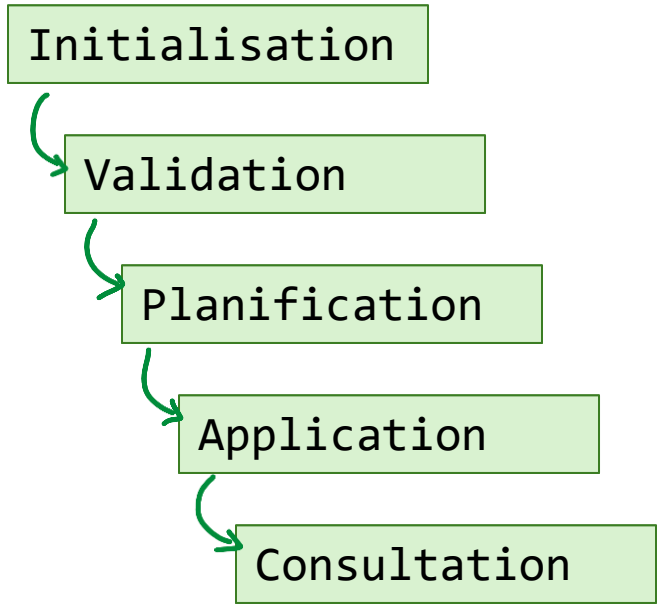
E.g.: Gmail, Microsoft 365, Dropbox

Terraform

Terraform permet de **définir, provisionner et gérer** l'infrastructure cloud de manière **déclarative**.

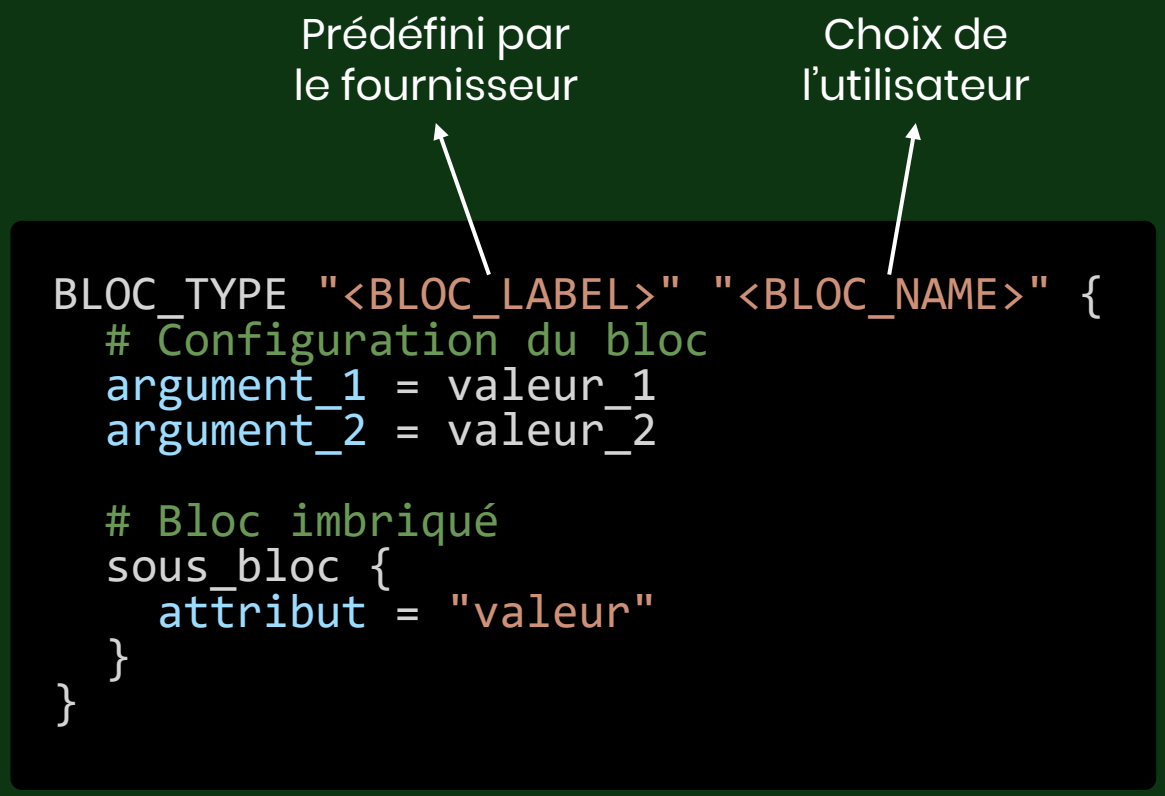


Terraform: Workflow




```
# initialiser le projet
terraform init

# vérifier la syntaxe et améliorer le format
terraform validate
terraform fmt

# pré-visualiser les changements et plan d’exécution
terraform plan

# exécuter le plan et les modifications
terraform apply

# afficher l'état actuel, les outputs, les ressources
terraform show
terraform output
terraform state list

# détruire l'infrastructure
terraform destroy
```

Le Langage HCL



Le Langage HCL

Types de blocs:

- **resource**: les composants d'infrastructure que vous créez

```
resource "<TYPE_DE_RESSOURCE>" "<NOM_LOCAL>" {
  # Configuration
}
```

```
# Exemple

# Créer un serveur
resource "aws_instance" "app_server" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"
}

# Créer un bucket S3 (= simple storage service)
resource "aws_s3_bucket" "website" {
  bucket = "mon-site-web-unique-123"
}


# Créer une base de données
resource "aws_db_instance" "postgres" {
  engine         = "postgres"
  instance_class = "db.t3.micro"
  allocated_storage = 20
}
```

Le Langage HCL

Types de blocs:

- **resource**: les comp
  d'infrastructure que

```
# Configuration d'un serveur

resource "aws_instance" "mon_serveur" {
  # --- OBLIGATOIRE ---
  ami           = "ami-0c55b159cbfafe1f0" # ID Ubuntu (exemple)
  instance_type = "t2.micro"              # Taille du serveur


  # --- TRÈS RECOMMANDÉ ---
  key_name      = "ma-cle-ssh"            # Pour pouvoir se connecter
  subnet_id     = "subnet-12345678"       # Pour savoir où le placer

  # Pour ouvrir les ports (ex: 80 pour le Web, 22 pour le SSH)
  vpc_security_group_ids = ["sg-0987654321"]


  # --- ORGANISATION ---
  tags = {
    Name        = "Serveur-App-Prod"
    Environment = "Production"
  }
}
```

Le Langage HCL

Types de blocs:

- **resource**: les composants d'infrastructure que vous créez
- **variable**: définies par l'utilisateur pour la flexibilité du code

```
# Déclaration
variable "nom_de_variable" {
  description = "Description ……"
  type        = string
  default     = "valeur_par_defaut"
}

# Utilisation
resource "aws_instance" "server" {
  instance_type = var.nom_de_variable
}
```

```
# Type de variables

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


# nombre
variable "instance_count" {
  type    = number
  default = 3
}


# booléen
variable "enable_monitoring" {
  type    = bool
  default = true
}


# List
variable "availability_zones" {
  type    = list(string)
  default = ["us-east-1a", "us-east-1b"]
}
```

Le Langage HCL

```
data "<TYPE_DE_DATA_SOURCE>" "<NOM_LOCAL>" {
  # Filtres pour trouver la ressource
}
```

Types de blocs:

- **resource**: les composants d'infrastructure que vous créez
- **variable**: définies par l'utilisateur pour la flexibilité du code
- **data**: lecture des informations sur des ressources existantes

```
# Exemple 1

# Récupérer l'AMI Ubuntu la plus récente
data "aws_ami" "ubuntu" {
  most_recent = true
  owners      = ["099720109477"] # compte official de Canonical

  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-*"]
  }
}


# Utiliser cette AMI
resource "aws_instance" "server" {
  ami = data.aws_ami.ubuntu.id  # ← Référence au data source
  instance_type = "t2.micro"
}
```

Le Langage HCL

Types de blocs:

- **resource**: les composants d'infrastructure que vous créez
- **variable**: définies par l'utilisateur pour la flexibilité du code
- **data**: lecture des informations sur des ressources existantes

```
# Exemple 2

# rechercher le VPC par défaut d'AWS dans la région actuelle
data "aws_vpc" "default_vpc" {
  default = true
}


# rechercher un sous-réseau (subnet) public dans ce VPC
data "aws_subnets" "mysubnets" {
  filter {
    name   = "vpc-id"
    values = [data.aws_vpc.default_vpc.id]
  }
}


# créer un serveur dans le premier subnet trouvé
resource "aws_instance" "web" {
  ami           = "ami-0abcd1234efgh5678" # (Exemple d'ID)
  instance_type = "t2.micro"

  # On pointe vers l'ID du premier subnet de la liste récupérée
  subnet_id     = data.aws_subnets.mysubnets.ids[0]
}
```

Le Langage HCL

```
output "nom_output" {
  description = "Description de l'output"
  value       = resource.type.name.attribut
}
```

Types de blocs:

- **resource**: les composants d'infrastructure que vous créez
- **variable**: définies par l'utilisateur pour la flexibilité du code
- **data**: lecture des informations sur des ressources existantes
- **output**: affiche des informations après déploiement
- ……

```
# Exemple

# Afficher l'IP publique d'un serveur
output "server_public_ip" {
  description = "Adresse IP publique du serveur"
  value       = aws_instance.web.public_ip
}


# Afficher l'URL d'un site web
output "website_url" {
  description = "URL du site web"
  value       = "http://${aws_s3_bucket_website_configuration.
website.website_endpoint}"
}


# Output avec plusieurs valeurs
output "server_details" {
  value = {
    id         = aws_instance.web.id
    public_ip  = aws_instance.web.public_ip
    private_ip = aws_instance.web.private_ip
  }
}
```

Terraform: Structure dossier

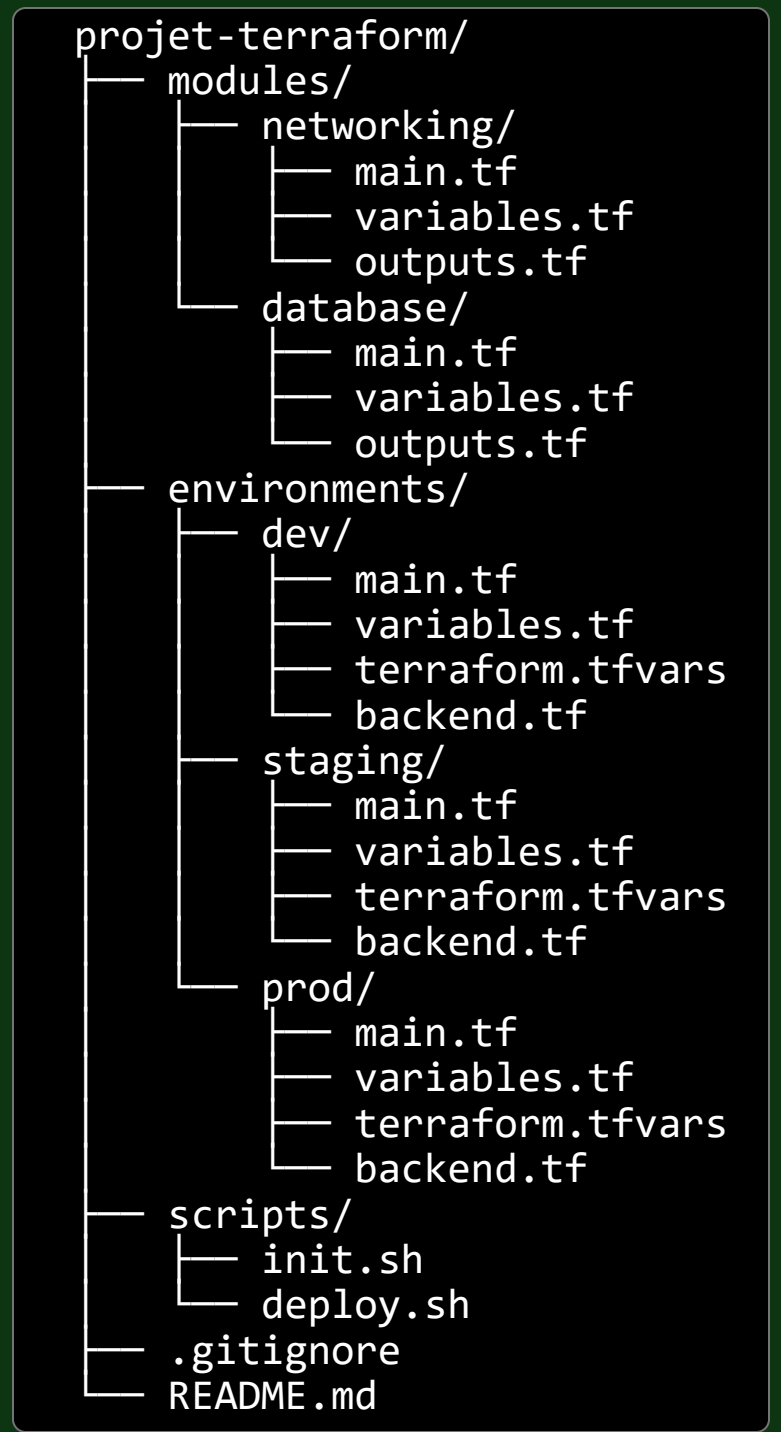
```
mon-projet/
├── provider.tf          # Configuration des fournisseurs
├── versions.tf          # Version de Terraform et providers
├── variables.tf         # Déclarations de variables
├── terraform.tfvars     # Valeurs des variables (ne pas commiter!)
├── main.tf              # Ressources principales
└── outputs.tf           # Déclarations d'outputs
```

Terraform: Structure dossier

Contient la logique technique (les blocs resource complexes).

Contient les appels aux modules et les configurations spécifiques pour l'application concrète et avec des valeurs réelles

Utilitaires pour gagner du temps



Terraform: Application

1. Préparation des fichiers de l'app (HTML, Multiverse)
2. Préparer les fichiers Terraform, et configurer l'infrastructure:
   A. Fournisseur AWS + region
   B. Réseaux VPC et Subnet public
   C. Security Group ouvrant le port 80 pour autoriser le trafic HTTP entrant
   D. aws_instance qui servira de serveur web
3. Exécuter Terraform pour créer les ressources et copier les fichiers de l'app (user_data)
4. Terraform affiche l'URL finale via le bloc d'output

https://github.com/hibanajjar998/cours-devops-25-26/tree/main/lesson5_IaC/exemple_terraform

```
GITHUB/hibanajjar998/cours-devops-25-26/lesson5_IaC/exemple_terraform/
├── terraform/              # toute votre infra (blocs resource, data, output)
│   ├── main.tf
│   ├── variables.tf
│   └── outputs.tf
└── app/                    # fichiers de l'application
    ├── index.html
    ├── images/
    └── assets/
```

Pipeline CD

CRÉER l'infrastructure

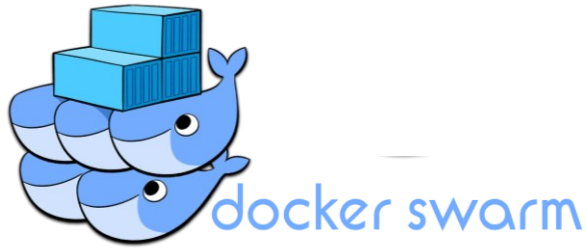
- Serveurs (EC2, VMs)
- Réseaux (VPC, Subnets)
- Bases de données
- Stockage

Nœud A	Nœud B	Nœud C
ubuntu	ubuntu	ubuntu
vide	vide	vide

CONFIGURER les serveurs

- Docker
- Initialise Docker Swarm
- Joint les workers au cluster

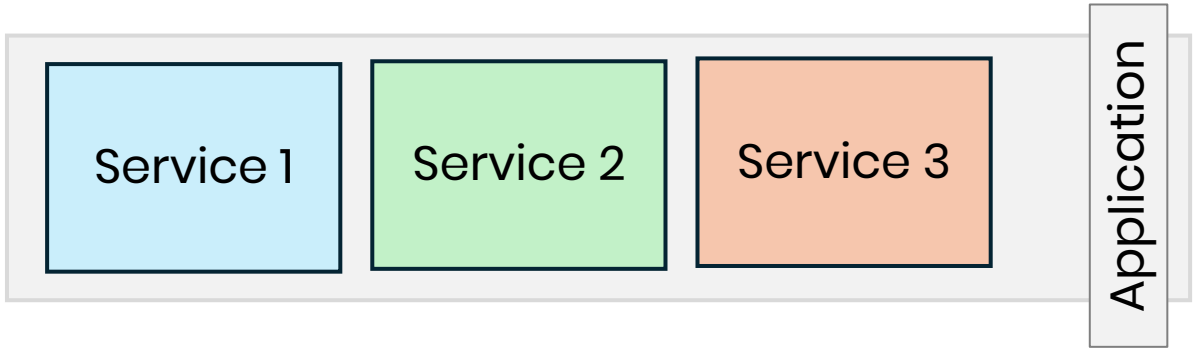
Nœud A	Nœud B	Nœud C
ubuntu	ubuntu	ubuntu
Docker	Docker	Docker
Swarm	Swarm	Swarm
Manager	**Worker**	**Worker**

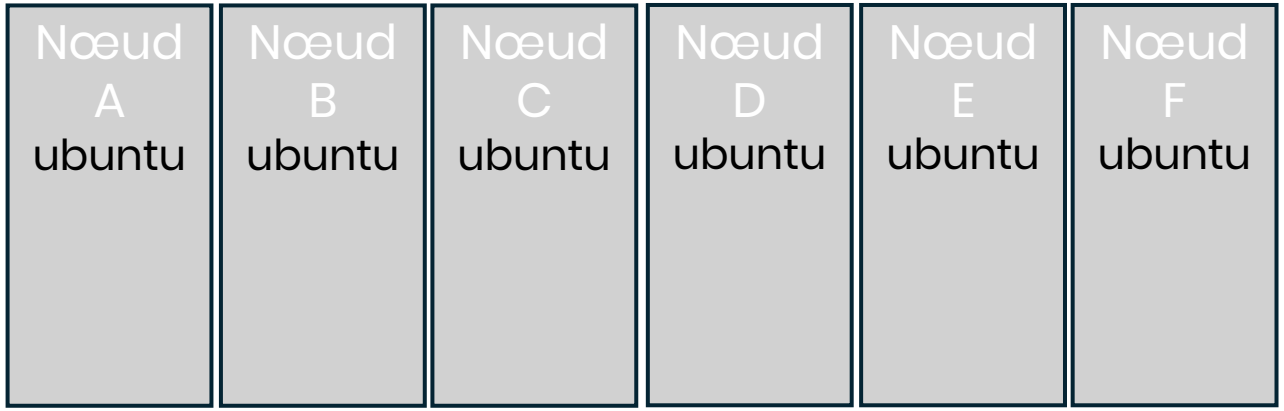


GÉRER les applications

- Déploiement de conteneurs
- Load balancing
- Self-healing
- Mise à l'échelle (Scaling)
- Rolling updates

Pipeline CD

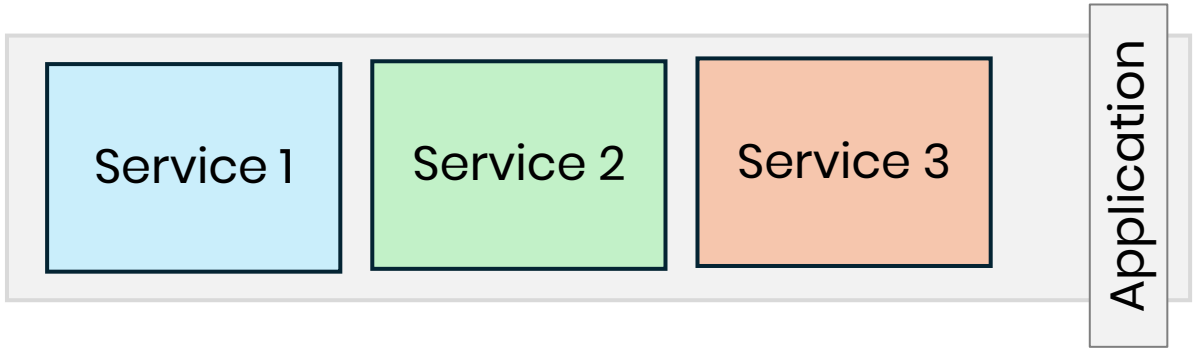




HashiCorp Terraform

Création de l'infrastructure

Pipeline CD

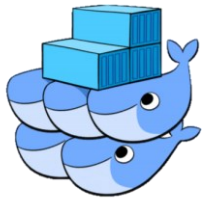


Configuration de l'infrastructure

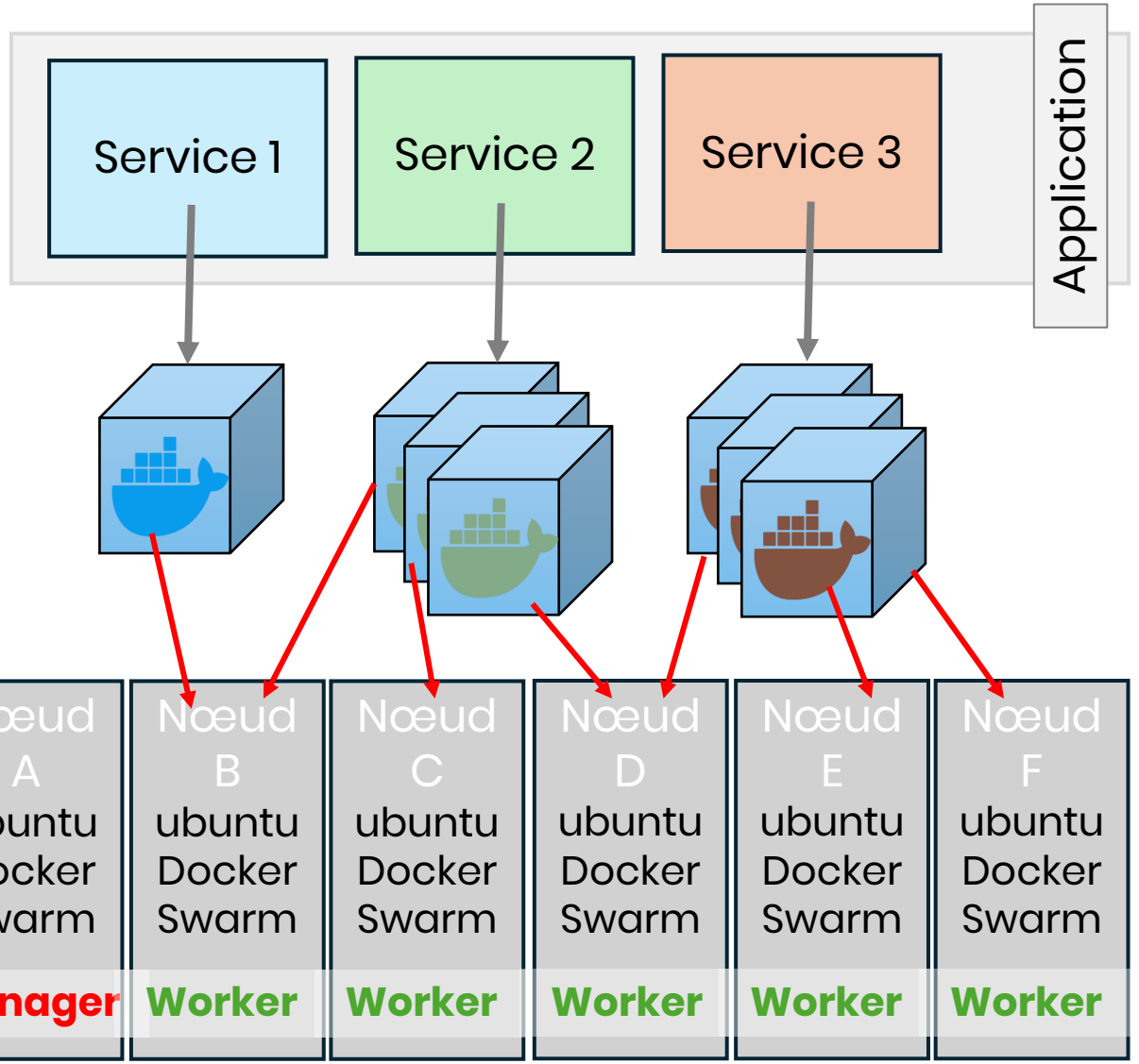
Nœud A	Nœud B	Nœud C	Nœud D	Nœud E	Nœud F
ubuntu Docker Swarm	ubuntu Docker Swarm	ubuntu Docker Swarm	ubuntu Docker Swarm	ubuntu Docker Swarm	ubuntu Docker Swarm
Manager	**Worker**	**Worker**	**Worker**	**Worker**	**Worker**

Création de l'infrastructure

Pipeline CD



Gestion et Orchestration des conteneurs



RED HAT ANSIBLE

Configuration de l'infrastructure

HashiCorp Terraform

Création de l'infrastructure

Terraform x Ansible x Docker Swarm

1. Préparation des fichiers de l'app (HTML, Multiverse)
2. Créer et publier une image Docker de l'app
3. Préparer les fichiers Terraform, et configurer l'infrastructure:
4. Exécuter Terraform pour
    1. créer les ressources
    2. récupérer les adresses IP des nœuds (manager + workers) dans un fichier `../ansible/inventory.ini` nécessaire à Ansible
5. Ansible utilise le fichier `../ansible/playbook.yml` pour continuer le reste du déploiement:
    1. se connecter aux instances via SSH et installer le moteur Docker sur chaque nœud
    2. initialiser le mode Swarm sur le Manager, récupère le jeton (token) de sécurité, et commander aux Workers de rejoindre le cluster.
    3. copier le docker-compose.yml sur le Manager.
    4. lancer docker stack deploy.

https://github.com/hibanajjar998/cours-devops-25-26/tree/main/lesson5_IaC/exemple_terraform_ansible_dockerswarm

Projets

- Pitch des projets séance prochaine
- La consigne est sur le dépôt GitHub du cours
- envoie des présentations (pptx/pdf) au plus tard le 4 Janvier pour le groupe B et le 5 janvier pour le groupe A(minuit).