Terraform : Infrastructure as Code de A à Z

Guide complet pour comprendre et maîtriser Terraform : concepts fondamentaux, syntaxe HCL, modules, state management, et intégration avec Ansible et Kubernetes.

DevOps 45 min de lecture Niveau Intermédiaire

Document généré le 13/05/2026 à 11h14 · nouvy.fr/wiki/terraform-infrastructure-as-code

Sommaire

72 section(s) · 45 min de lecture

L''idée centrale

Terraform sert à décrire ce que tu veux, pas comment le faire.

Tu dis :

> *"Je veux que ça ressemble à ça."*

Terraform se débrouille.

Sans Terraform

Tu veux un serveur :

- Tu cliques
- Tu choisis des options
- Tu oublies lesquelles
- Personne ne peut refaire pareil
- Si ça casse → panique

Avec Terraform

Tu écris un fichier texte :

```
Je veux :
- 1 serveur
- il s''appelle web
- il écoute sur le port 80
```

Copier

Et tu peux dire :

- Refais-le
- Détruis-le
- Change-le
- Recrée-le

Sans réfléchir, sans mémoire humaine

Les 5 mots-clés Terraform (ultra simples)

Mot	Image mentale	Description
Provider	Qui fait le boulot	AWS, Azure, GCP, local, etc.
Resource	Ce que tu veux	Serveur, fichier, réseau, etc.
State	Le carnet de chantier	Mémoire de ce qui existe
Plan	Ce qui va changer	Différence entre voulu et réel
Apply	Exécuter	Appliquer les changements

Ce qui se passe vraiment

1. **Tu écris ce que tu veux** → Fichiers `.tf`
2. **Terraform regarde ce qui existe** → Lit le state
3. **Il calcule la différence** → `terraform plan`
4. **Il applique juste ce qu''il faut** → `terraform apply`

Jamais tout casser pour rien

Exemple concret (local, simple)

```
resource "local_file" "exemple" {
  filename = "bonjour.txt"
  content  = "Bonjour"
}
```

Copier

Tu ne dis **PAS** :

- "crée un fichier"
- "ouvre le disque"

Tu dis :

> *"Je veux ce fichier avec ce contenu"*

Terraform gère le reste.

Pourquoi les entreprises adorent ça

- **1 infra = 1 repo Git** → Versionnement complet
- **Historique clair** → Qui a fait quoi et quand
- **Rollback possible** → Retour en arrière facile

- **Zéro "c''était comment déjà ?"** → Tout est documenté dans le code

---

Installation de Terraform

Sur macOS

```
brew tap hashicorp/tap
brew install hashicorp/tap/terraform
```

Copier

Sur Debian/Ubuntu

```
# Installer GPG si nécessaire
sudo apt update
sudo apt install -y gpg

# Ajouter la clé GPG de HashiCorp
wget -O - https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/hashicorp-archive-keyring.gpg

# Ajouter le dépôt HashiCorp
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/hashicorp-
archive-keyring.gpg] https://apt.releases.hashicorp.com $(grep -oP
''(?<=UBUNTU_CODENAME=).*'' /etc/os-release || lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/hashicorp.list

# Mettre à jour et installer Terraform
sudo apt update && sudo apt install terraform
```

Copier

Sur Linux (méthode manuelle)

Si vous préférez installer manuellement :

```
# Télécharger la dernière version
wget https://releases.hashicorp.com/terraform/1.6.0/terraform_1.6.0_linux_amd64.zip
unzip terraform_1.6.0_linux_amd64.zip
sudo mv terraform /usr/local/bin/
```

Copier

Sur Windows

Téléchargez depuis terraform.io/downloads ou utilisez Chocolatey :

```
choco install terraform
```

Copier

Vérifier l''installation

```
terraform version
```

Copier

Structure d''un projet Terraform

Le dossier du projet

```
terraform-local-tp/
├── main.tf          # Ressources principales
├── variables.tf     # Variables d''entrée
├── outputs.tf       # Valeurs de sortie
├── terraform.tfstate # État (généré automatiquement)
└── modules/         # Modules réutilisables
    ├── network/
    └── compute/
```

Copier

C''est le classeur du chantier

Modules = plans de pièces

Module network

```
# modules/network/main.tf
resource "local_file" "vpc_config" {
  filename = "vpc.txt"
  content  = "VPC: ${var.vpc_name}"
}
```

Copier

Plan du terrain :

- **VPC (Virtual Private Cloud)** : C''est comme délimiter ton terrain privé dans le cloud. Un VPC est un réseau virtuel isolé où tu vas placer tes ressources (serveurs, bases de données, etc.). C''est l''équivalent d''un réseau local privé, mais dans le cloud. Tu peux définir des règles de sécurité, des plages d''adresses IP, et contrôler qui peut communiquer avec quoi.
- **Subnet** : Ce sont des sous-réseaux à l''intérieur du VPC. Comme diviser ton terrain en zones (zone publique pour les serveurs web accessibles depuis Internet, zone privée pour les bases de données non accessibles directement).
- (en local : fichiers texte)

Module compute

```
# modules/compute/main.tf
resource "local_file" "server_config" {
  filename = "server.txt"
  content  = "Server: ${var.server_name}"
}
```

Copier

Plan de la maison :

- Serveur
- Installation Nginx (simulée)

Un module = un plan réutilisable

Variables = choix du client

```
# variables.tf
variable "environment" {
  description = "Environnement de déploiement"
  type        = string
  default     = "dev"
}

variable "server_count" {
  description = "Nombre de serveurs"
  type        = number
  default     = 1
}
```

Copier

```
# terraform.tfvars
environment  = "dev"
server_count = 2
```

Copier

Comme dire :

- "c''est un chantier de test"
- "ou c''est la vraie maison"

Environnements (dev / prod)

Même plan Mais :

- Matériaux différents
- Finitions différentes

Même code, résultats adaptés

Exemple avec workspaces

```
# Créer un workspace pour dev
terraform workspace new dev
terraform workspace select dev

# Créer un workspace pour prod
terraform workspace new prod
terraform workspace select prod
```

Copier

```
# Utilisation dans le code
resource "local_file" "config" {
  filename = "config-${terraform.workspace}.txt"
  content  = "Environment: ${terraform.workspace}"
}
```

Copier

Le State = carnet de chantier (IMPORTANT)

Terraform note :

- Ce qui existe
- Où c''est
- Comment c''est fait

Si tu supprimes le carnet

Terraform est perdu

En entreprise : carnet partagé et sécurisé

Backend remote (recommandé)

```
# backend.tf
terraform {
  backend "s3" {
    bucket = "mon-terraform-state"
    key    = "terraform.tfstate"
    region = "eu-west-1"
  }
}
```

Copier

Alternatives :

- **S3** (AWS)
- **Azure Storage** (Azure)

- **GCS** (Google Cloud)
- **Terraform Cloud** (HashiCorp)

Exemple complet : Infrastructure locale

1. Créer le projet

```
mkdir terraform-exemple
cd terraform-exemple
```

Copier

2. Créer `main.tf`

```
terraform {
  required_version = ">= 1.0"
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "~> 2.0"
    }
  }
}

# Variable
variable "server_name" {
  description = "Nom du serveur"
  type        = string
  default     = "web-server"
}

# Ressource
resource "local_file" "server_config" {
  filename = "${var.server_name}.txt"
  content  = <<-EOT
    Server Name: ${var.server_name}
    Port: 80
    Status: Running
  EOT
}

# Output
output "server_file" {
  value       = local_file.server_config.filename
  description = "Fichier de configuration créé"
}
```

Copier

3. Initialiser Terraform

```
terraform init
```

Copier

Cette commande :

- Télécharge les providers nécessaires
- Prépare le backend
- Crée le dossier `.terraform/`

4. Voir ce qui va être créé

```
terraform plan
```

Copier

Résultat :

```
Terraform will perform the following actions:

  # local_file.server_config will be created
  + resource "local_file" "server_config" {
      + content  = "Server Name: web-server\nPort: 80\nStatus: Running\n"
      + filename = "web-server.txt"
    }

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

Copier

5. Appliquer les changements

```
terraform apply
```

Copier

Terraform demande confirmation, puis crée le fichier.

6. Vérifier

```
cat web-server.txt
```

Copier

7. Modifier et réappliquer

Modifiez `main.tf` :

```
resource "local_file" "server_config" {
  filename = "${var.server_name}.txt"
  content  = <<-EOT
    Server Name: ${var.server_name}
    Port: 8080  # Changé de 80 à 8080
    Status: Running
  EOT
}
```

Copier

```
terraform plan  # Voir les changements
terraform apply # Appliquer
```

Copier

8. Détruire l''infrastructure

```
terraform destroy
```

Copier

Cas d''usage réel : Déployer Laravel avec Docker sur Debian

Scénario

Tu as déjà un serveur Debian avec Docker installé et qui tourne. Tu veux déployer ton projet Laravel dessus en utilisant Terraform pour gérer toute la configuration.

Prérequis

1. **Serveur Debian** avec Docker et Docker Compose installés
2. **Accès SSH** au serveur
3. **Terraform** installé sur ta machine locale

Structure du projet

```
terraform-laravel/
├── provider.tf
├── main.tf
├── variables.tf
├── outputs.tf
└── docker-compose.tf
```

Copier

Configuration complète

1. `provider.tf` - Configuration des providers

```
terraform {
  required_version = ">= 1.0"
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0"
    }
    null = {
      source  = "hashicorp/null"
      version = "~> 3.0"
    }
    local = {
      source  = "hashicorp/local"
      version = "~> 2.0"
    }
  }
}

# Provider Docker (se connecte au daemon Docker du serveur)
provider "docker" {
  host = "ssh://${var.server_user}@${var.server_ip}:${var.server_ssh_port}"
}
```

Copier

2. `variables.tf` - Variables du projet

```
variable "server_ip" {
  description = "Adresse IP du serveur Debian"
  type        = string
}

variable "server_user" {
  description = "Utilisateur SSH pour se connecter au serveur"
  type        = string
}

variable "server_ssh_port" {
  description = "Port SSH du serveur Debian"
  type        = number
}

variable "app_name" {
  description = "Nom de l''application"
  type        = string
}

variable "app_port" {
  description = "Port sur lequel l''application sera accessible"
  type        = number
}

variable "db_password" {
  description = "Mot de passe root de la base de données MySQL"
  type        = string
  sensitive   = true
}

variable "db_name" {
  description = "Nom de la base de données"
  type        = string
}
```

```
variable "docker_image" {
  description = "Image Docker à utiliser pour l''application Laravel"
  type        = string
}

# VARIABLES AJOUTEES
variable "git_repo" {
  description = "URL du repo Git"
  type        = string
}

variable "github_token" {
  description = "Token GitHub pour accéder aux dépôts privés"
  type        = string
  sensitive   = true
}

variable "git_branch" {
  description = "Branche Git à cloner"
  type        = string
  default     = "main"
}

variable "app_type" {
  description = "Type d''application (laravel / generic)"
  type        = string
  default     = "laravel"
}

variable "run_migrations" {
  description = "Exécuter les migrations Laravel"
  type        = bool
  default     = true
}

variable "run_seeders" {
  description = "Exécuter les seeders Laravel"
  type        = bool
  default     = false
}
```

Copier

3. `docker-compose.tf` - Créer le fichier docker-compose.yml

```
# Création du docker-compose.yml
resource "local_file" "docker_compose" {
  filename = "docker-compose.yml"
  content  = <<-EOT
version: ''3.8''

services:
  app:
    image: ${var.docker_image}
    container_name: ${var.app_name}-app
    ports:
      - "${var.app_port}:80"
    environment:
      GIT_REPO: "${var.git_repo}"
      GITHUB_TOKEN: "${var.github_token}"
      GIT_BRANCH: "${var.git_branch}"
      APP_TYPE: "${var.app_type}"
      RUN_MIGRATIONS: "${var.run_migrations}"
      RUN_SEEDERS: "${var.run_seeders}"
      APP_NAME: "${var.app_name}"
      APP_ENV: "dev"
      APP_DEBUG: "true"
      APP_URL: "http://${var.server_ip}:${var.app_port}"
      DB_CONNECTION: "mysql"
      DB_HOST: "mysql"
      DB_PORT: 3306
      DB_DATABASE: "${var.db_name}"
      DB_USERNAME: "root"
      DB_PASSWORD: "${var.db_password}"
    networks:
      - laravel-network
    depends_on:
      - mysql
    restart: unless-stopped

  mysql:
    image: mysql:8.0
    container_name: ${var.app_name}-mysql
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: "${var.db_password}"
      MYSQL_DATABASE: "${var.db_name}"
    volumes:
      - mysql_data:/var/lib/mysql
    networks:
      - laravel-network
    ports:
      - "3307:3306"  # Port externe pour accès direct si besoin

volumes:
  mysql_data:

networks:
  laravel-network:
    driver: bridge
  EOT
}
```

Copier

4. `main.tf` - Déployer les conteneurs

```
resource "null_resource" "prepare_remote_dir" {
  triggers = {
```

```
    app_name      = var.app_name
    server_user   = var.server_user
    server_ip     = var.server_ip
    server_ssh_port = var.server_ssh_port
  }

  provisioner "remote-exec" {
    inline = [
      "mkdir -p /home/${var.server_user}/${var.app_name}"
    ]

    connection {
      type        = "ssh"
      host        = var.server_ip
      user        = var.server_user
      port        = var.server_ssh_port
      private_key = file("~/.ssh/id_eisi25_docker")
    }
  }

  provisioner "remote-exec" {
    when = destroy
    inline = [
      "rm -rf /home/${self.triggers.server_user}/${self.triggers.app_name}"
    ]

    connection {
      type        = "ssh"
      host        = self.triggers.server_ip
      user        = self.triggers.server_user
      port        = self.triggers.server_ssh_port
      private_key = file("~/.ssh/id_eisi25_docker")
    }
  }
}

# Exécuter docker-compose up sur le serveur
resource "null_resource" "deploy_laravel" {
  depends_on = [null_resource.prepare_remote_dir]

  triggers = {
    docker_compose = local_file.docker_compose.content
    app_port       = var.app_port
    db_password    = var.db_password
    app_name       = var.app_name
    server_user    = var.server_user
    server_ip      = var.server_ip
    server_ssh_port = var.server_ssh_port
  }
  # Copier le fichier docker-compose.yml sur le serveur
  provisioner "file" {
    source      = "docker-compose.yml"
    destination = "/home/${var.server_user}/${var.app_name}/docker-compose.yml"
    connection {
      type        = "ssh"
      host        = var.server_ip
      user        = var.server_user
      port        = var.server_ssh_port
      private_key = file("~/.ssh/id_eisi25_docker")
    }
  }

  # Exécuter docker-compose
  provisioner "remote-exec" {
    inline = [
```

```
      "cd /home/${var.server_user}/${var.app_name}",
      "docker compose down || true",        # Arrêter si déjà en cours
      "docker compose pull",                # Télécharger la dernière version de l''image
      "docker compose up -d",               # Démarrer en arrière-plan
      "sleep 10",                           # Attendre que les conteneurs démarrent et que
l''entrypoint.sh fasse son travail
      "docker compose ps",                  # Vérifier l''état des conteneurs
      "docker compose logs app | tail -20"  # Voir les derniers logs pour vérifier
    ]
    connection {
      type        = "ssh"
      host        = var.server_ip
      user        = var.server_user
      port        = var.server_ssh_port
      private_key = file("~/.ssh/id_eisi25_docker")
    }
  }

  provisioner "remote-exec" {
    when = destroy
    inline = [
      "cd /home/${self.triggers.server_user}/${self.triggers.app_name} || exit 0",
      "docker compose down -v || true",
      "docker compose rm -f || true"
    ]
    connection {
      type        = "ssh"
      host        = self.triggers.server_ip
      user        = self.triggers.server_user
      port        = self.triggers.server_ssh_port
      private_key = file("~/.ssh/id_eisi25_docker")
    }
  }
}
```

Copier

> ***Note** : L''image Docker personnalisée (`ghcr.io/nouvy/laravel-app:latest`) contient déjà un `entrypoint.sh` qui :*
>
> - Clone le dépôt Git (via `GIT_REPO`, `GITHUB_TOKEN`, `GIT_BRANCH`)
> - Configure Laravel (permissions, dépendances Composer)
> - Crée le fichier `.env` avec les variables d''environnement passées
> - Génère la clé d''application Laravel
> - Lance les migrations si `RUN_MIGRATIONS=true`
> - Lance les seeders si `RUN_SEEDERS=true`
> - Démarre Apache
>
> *Les variables d''environnement passées dans `docker-compose.yml` (comme `DB_HOST`, `DB_PASSWORD`, `GIT_REPO`, etc.) seront utilisées par l''application Laravel et l''entrypoint.sh.*

```
#### 5. `outputs.tf` - Informations de sortie

```hcl
output "app_url" {
  value       = "http://${var.server_ip}:${var.app_port}"
  description = "URL d''accès à l''application Laravel"
}

output "docker_compose_file" {
  value       = local_file.docker_compose.filename
  description = "Fichier docker-compose.yml créé"
}
```

Copier

6. `terraform.tfvars` - Valeurs des variables

```
server_ip      = "192.168.194.211"
server_user    = "fabrice"
server_ssh_port = 30625
git_repo       = "https://github.com/nouvy/laravel-app.git"
github_token   =
"github_pat_11AWYWPOY0HMGGl6Ohzb9i_kPEPxsviIHN3PtTPtNQP24oPzcPSkSaphKuyrOm6IMcCWZNKPOKfnuzxj
9I"
app_name       = "app-laravel-eisi25"
app_port       = 8084
db_password    = "secret"
db_name        = "app-laravel-eisi25"
docker_image   = "ghcr.io/nouvy/laravel-app:latest"
```

Copier

Commandes pour déployer

```
# 1. Initialiser Terraform
terraform init

# 2. Voir ce qui va être créé
terraform plan

# 3. Déployer l''application
terraform apply

# 4. Vérifier l''état
terraform show
```

Copier

Accéder à l''application

Une fois le déploiement terminé, accède à ton application Laravel via :

```
http://192.168.1.100:8080
```

Copier
```

Modifier et redéployer

Si tu modifies les variables (par exemple le port) :

```
# terraform.tfvars
app_port = 9090
```

Copier

```
terraform plan  # Voir les changements
terraform apply # Appliquer
```

Copier

Terraform va automatiquement :

1. Mettre à jour le fichier docker-compose.yml
2. Redémarrer les conteneurs avec la nouvelle configuration

Détruire l''infrastructure

```
terraform destroy
```

Copier

Cela va arrêter et supprimer tous les conteneurs Docker.

Pourquoi c''est puissant ?

- **Tout est versionné** : Ton infrastructure Laravel est dans Git
- **Reproductible** : N''importe qui peut déployer exactement pareil
- **Modifiable facilement** : Change une variable, réapplique
- **Documentation vivante** : Le code Terraform documente ton infrastructure

---

Terraform + Ansible

Rôle de chacun

Outil	Rôle principal
Terraform	Crée l''infrastructure (serveurs, réseaux, cloud)
Ansible	Configure ce qui est sur les serveurs (logiciels, fichiers, services)

Exemple concret

- **Terraform** crée une machine virtuelle
- **Ansible** installe dessus Nginx, MySQL, utilisateurs, certificats, etc.

Workflow type

1. `terraform apply` → La machine existe
2. Ansible prend la machine → Configure le logiciel
3. Tout est prêt, versionnable et réutilisable

C''est comme : Construire une maison (Terraform) → La meubler et la décorer (Ansible)

Terraform + Kubernetes

Rôle de chacun

Outil	Rôle principal
Terraform	Crée les ressources cloud : VM, load balancer, réseau, clusters Kubernetes
Kubernetes	Gère les applications à l''intérieur : pods, services, déploiements
Helm (optionnel)	Facilite les déploiements Kubernetes comme un gestionnaire de paquets

Exemple concret

- **Terraform** crée un cluster Kubernetes complet sur AWS/GCP/Azure
- **Kubernetes** déploie les applications en microservices, gère la scalabilité et la disponibilité

Workflow type

1. **Terraform** → "Cluster Kubernetes prêt"
2. **Kubernetes** → Déploie toutes les apps
3. **Helm / YAML** → Déploie microservices, base de données, front, etc.

Analogie maison complète

- **Terraform** → Fondations et murs
- **Ansible** → Plomberie, électricité, cuisine
- **Kubernetes** → Locataires et gestion des appartements

Concepts avancés

Data Sources

Permet de récupérer des informations existantes :

```
data "aws_ami" "latest_amazon_linux" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }
}

resource "aws_instance" "web" {
  ami           = data.aws_ami.latest_amazon_linux.id
  instance_type = "t2.micro"
}
```

Copier

Locals

Variables calculées réutilisables :

```
locals {
  common_tags = {
    Environment = var.environment
    Project     = "MonProjet"
    ManagedBy   = "Terraform"
  }
}

resource "aws_instance" "web" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
  tags = local.common_tags
}
```

Copier

Conditions

```
resource "aws_instance" "web" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = var.environment == "prod" ? "t3.medium" : "t2.micro"
  count = var.create_instance ? 1 : 0
}
```

Copier

For Each

Créer plusieurs ressources similaires :

```
variable "servers" {
  type = map(object({
    instance_type = string
  }))
  default = {
    web1 = { instance_type = "t2.micro" }
    web2 = { instance_type = "t2.micro" }
    db   = { instance_type = "t3.small" }
  }
}

resource "aws_instance" "servers" {
  for_each      = var.servers
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = each.value.instance_type
  tags = {
    Name = each.key
  }
}
```

Copier

Bonnes pratiques

1. Versionner le code

```
git init
git add .
git commit -m "Initial Terraform configuration"
```

Copier

2. Utiliser des modules

Au lieu de tout mettre dans `main.tf`, créez des modules :

```
modules/
├── network/
│   ├── main.tf
│   ├── variables.tf
│   └── outputs.tf
├── compute/
│   ├── main.tf
│   ├── variables.tf
│   └── outputs.tf
└── database/
    ├── main.tf
    ├── variables.tf
    └── outputs.tf
```

Copier

3. Backend remote pour le state

Ne jamais commiter `terraform.tfstate` dans Git. Utilisez un backend remote.

4. Utiliser des variables

Ne jamais hardcoder les valeurs. Utilisez des variables et des fichiers `.tfvars`.

5. Documentation

Ajoutez des descriptions à toutes les variables et outputs :

```
variable "instance_type" {
  description = "Type d''instance EC2 à utiliser"
  type        = string
  default     = "t2.micro"
}
```

Copier

6. Validation des variables

```
variable "environment" {
  description = "Environnement de déploiement"
  type        = string
  validation {
    condition     = contains(["dev", "staging", "prod"], var.environment)
    error_message = "L''environnement doit être dev, staging ou prod."
  }
}
```

Copier

Erreurs courantes et solutions

Erreur : Provider not found

```
Error: Failed to query available provider packages
```

Copier

Solution : Exécutez `terraform init`

Erreur : State locked

```
Error: Error acquiring the state lock
```

Copier

Solution : Un autre processus utilise Terraform. Attendez ou forcez le déverrouillage :

```
terraform force-unlock <LOCK_ID>
```

Copier

Erreur : Resource already exists

Solution : Importez la ressource existante :

```
terraform import aws_instance.web i-1234567890abcdef0
```

Copier

Commandes essentielles

Commande	Description
`terraform init`	Initialise le projet et télécharge les providers
`terraform plan`	Affiche ce qui va changer (sans appliquer)
`terraform apply`	Applique les changements
`terraform destroy`	Détruit toutes les ressources
`terraform validate`	Valide la syntaxe des fichiers
`terraform fmt`	Formate les fichiers `.tf`
`terraform show`	Affiche l''état actuel
`terraform output`	Affiche les outputs
`terraform state list`	Liste toutes les ressources dans le state
`terraform state show <resource>`	Affiche les détails d''une ressource

Résumé en 5 phrases

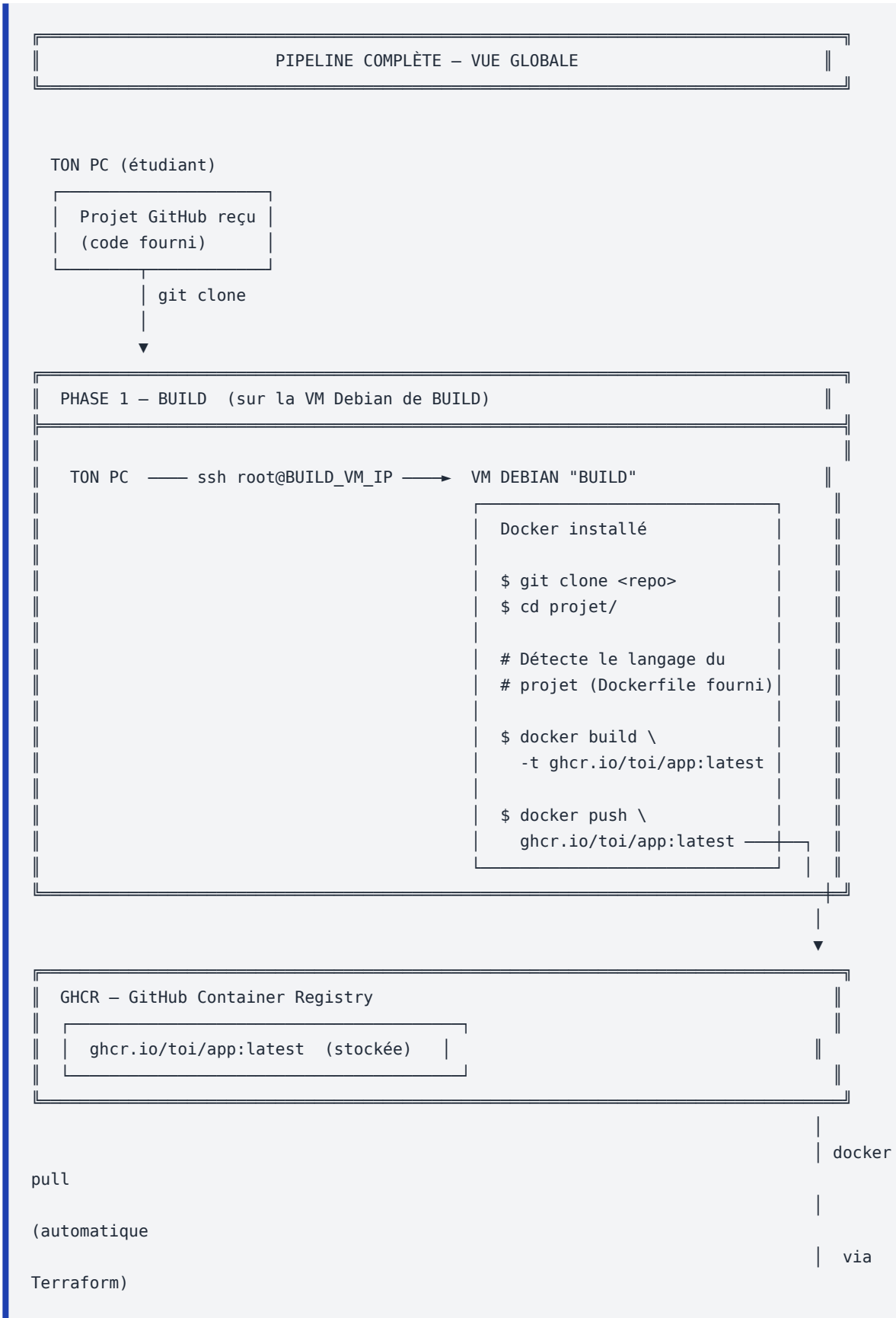
1. **Terraform décrit l''état final voulu** → Tu écris ce que tu veux, pas comment
2. **Tu peux tout refaire exactement pareil** → Infrastructure reproductible
3. **Les modules sont des plans** → Code réutilisable et organisé
4. **Les variables adaptent le résultat** → Même code, environnements différents
5. **Le state est la mémoire de Terraform** → Ne jamais le perdre, utiliser un backend remote

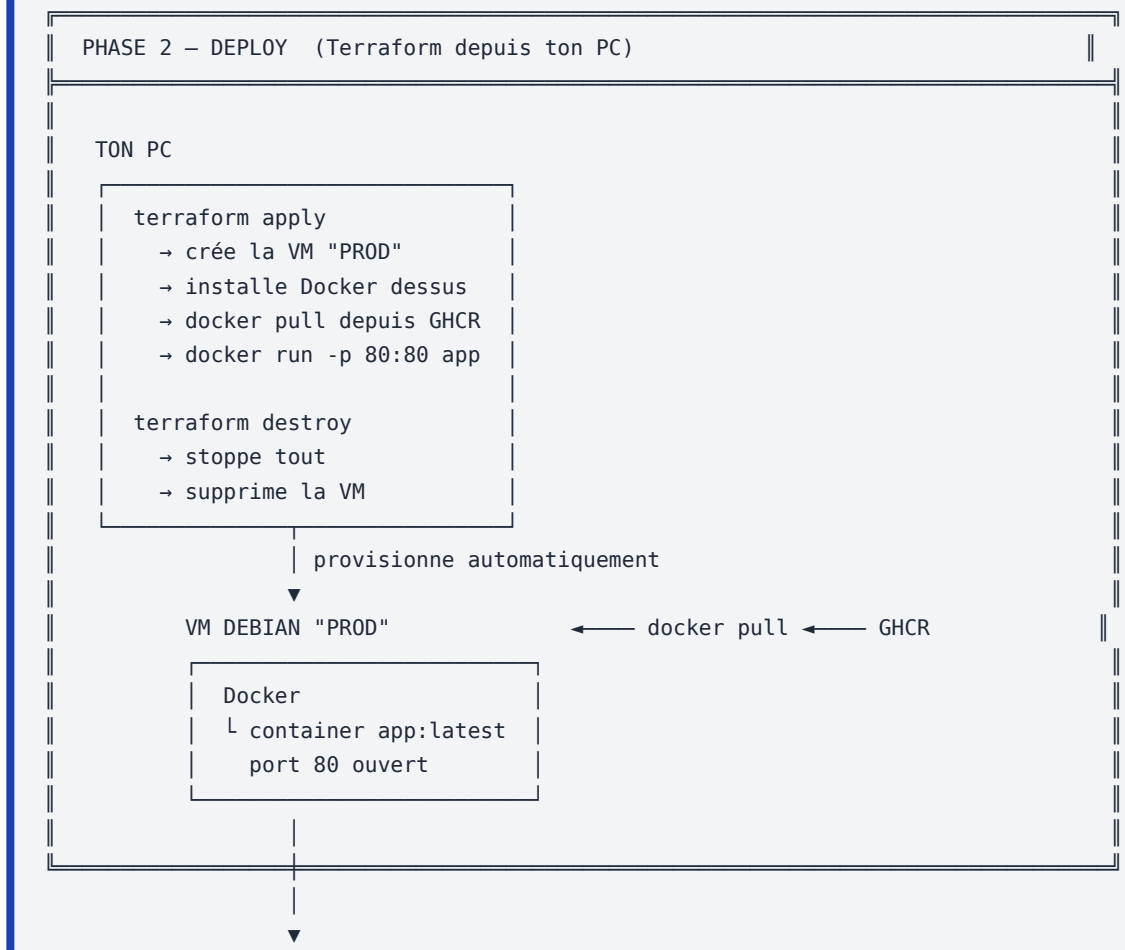
---

Ressources supplémentaires

- Documentation officielle Terraform
- Terraform Registry - Modules et providers
- Terraform Best Practices
- Terraform AWS Provider

Pipeline complète — Vue globale






```
═══════════════════════════════════════════════════════════════

  RÉSUMÉ DES RÔLES

═══════════════════════════════════════════════════════════════

  TON PC          →  git clone + ssh + terraform apply/destroy
  VM "BUILD"      →  docker build + docker push  (machine de compilation)
  GHCR            →  stockage de l''image Docker  (registre intermédiaire)
  Terraform       →  crée et détruit la VM PROD automatiquement
  VM "PROD"       →  docker pull + docker run    (machine de production)

═══════════════════════════════════════════════════════════════

  FLUX EN UNE LIGNE

  GitHub repo → SSH Build VM → docker build → GHCR → Terraform → VM Prod

═══════════════════════════════════════════════════════════════
```

Copier

Lien du repo Kernel Panic : [Nouvy/kernel-panic]