



# Training Terraform



Bonjour



Bonjour





**Plan**

# Plan

Que va-t-on faire durant 2 jours ?

- Qu'est-ce que Terraform ?
- Pourquoi Terraform ?
- Workflow
- Bases de la syntaxe HCL
- Terraform states
- Modules
- Syntaxe HCL avancée
- Layering
- Workspaces
- Utiliser Terraform en équipe

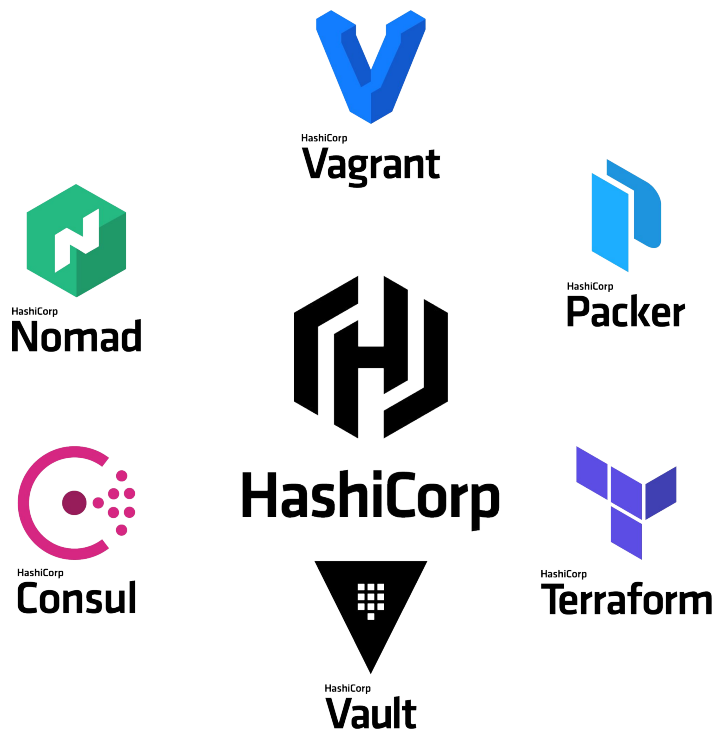




**Qu'est-ce que Terraform ?**

# Hashicorp

## Photo de famille



# Terraform

Ça sert à quoi ?

- Gérer son infrastructure comme du code
- Planifier les modifications
- Créer des architectures reproductibles





# Variables

## Les concepts

Permet de passer des valeurs en entrée de notre code Terraform. Il est possible de créer des variables “locales”, qui ne sont pas des paramètres, pour simplifier le code. Les variables sont typées :

- String
- Boolean
- Integer
- Float
- List
- Map



# Providers

## Les concepts

Connecteur vers l'API d'un fournisseur.

- AWS
- GCP
- OpenStack
- Alicloud
- VMWare
- DigitalOcean
- Datadog
- NewRelic
- Icinga2
- Rundeck
- Bitbucket
- Gitlab
- Kubernetes
- Docker
- Nomad
- Consul
- Cobbler
- et plus encore ...

# Datasources

## Les concepts

Variable créée par le requêtage d'un provider. Permet de collecter de l'information réutilisable simplement.

- `aws_db_instance`
- `aws_ebs_snapshot`
- `aws_vpc`
- `aws_iam_user`
- `aws_ami`
- `aws_eip`

# Ressources

## Les concepts

Objet à créer auprès d'un provider.

- `aws_db_instance`
- `aws_ebs_snapshot`
- `aws_vpc`
- `aws_iam_user`
- `aws_ami`
- `aws_eip`

# TfState

## Les concepts

Un fichier tfstate est produit en sortie d'un lancement Terraform pour stocker les attributs techniques des datasources et des ressources produites auprès des providers.

# Outputs

## Les concepts

Variable construite par Terraform afin d'être consommées après la création des ressources. Une donnée technique de ressource ne peut être consommée que si elle est pointée par un output.

# Backends

## Les concepts

Connecteur vers un espace ou une API de stockage qui permet de stocker le tfstate. Sert à mettre à disposition les outputs à d'autres équipes et afin de mutualiser le tfstate.



**Pourquoi Terraform ?**



# Terraform

## Critères de choix

- Multi providers
- Syntaxe unique
- Workflow unique
- Calcul de plan d'exécution
- Partage d'informations
- Vendor lock-in faible



# Workflow

# Important à savoir

## Workflow

**Toute commande Terraform est à lancer dans un répertoire contenant des fichiers Terraform.**

# terraform **init**

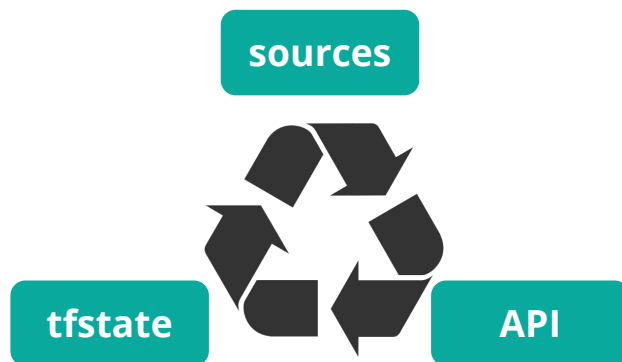
## Workflow

- installation des modules
- installation des plugins de providers
- initialisation des backends
- récupération du tfstate

# terraform **plan**

## Workflow

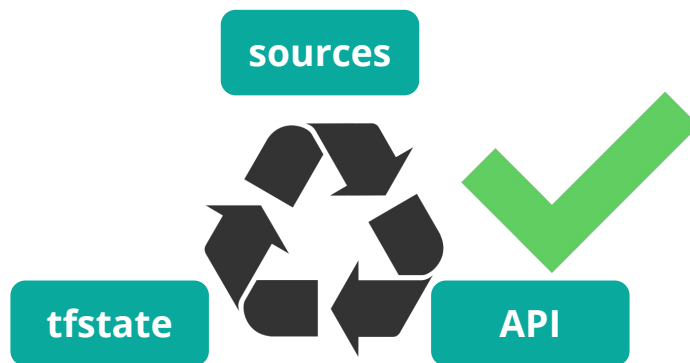
Calcul et présente le plan d'exécution qui serait lancé en cas de apply.



# terraform **apply**

## Workflow

Lance un **plan**, le propose et sur acceptation, lance les opérations du plan.



# terraform **others**

## Workflow

Parmi les autres actions, on trouve :

- destroy
- output
- fmt
- validate

# Install

- PyCharm Community
- PyCharm Plugin HCL/Terraform
- Terraform



# Mise en pratique



<https://gitlab.com/wescalefr/training-terraform>

15 minutes





# Bases de la syntaxe HCL

# Formalisme

## Syntaxe HCL

- Hashicorp Configuration Language
- JSON allégé

```
provider "aws" {  
  alias    = "paris"  
  region  = "eu-west-3"  
}
```

# Providers

## Syntaxe HCL

- Client avec les API d'un fournisseur de service
- Chaque ressource peut avoir son provider
- Plusieurs instance de provider vers le même fournisseur

```
provider "aws" {  
  alias    = "paris"  
  region   = "eu-west-3"  
}
```

# Ressources

## Syntaxe HCL

- Correspond à un objet créé
- Lien entre les objets
- Forcer la création

```
resource "aws_instance" "web" {  
    instance_type = "t2.micro"  
}
```

# Variables

## Syntaxe HCL

- Variable...
- Peut avoir un type (string, int, list, map)
- Peut avoir une valeur par défaut

```
variable "stack_name" {  
    type      = "string"  
    default = "unnamed"  
}
```

# Usage des variables

## Syntaxe HCL

- Les variables peuvent être injecté dans le reste du code Terraform
- Une librairie de fonction de transformation est disponible

```
resource "aws_instance" "web" {  
  subnet = "${var.subnet}"  
  count   = "${var.db_instance_count + 1}"  
}
```



# Locales

## Syntaxe HCL

- Variable construite durant le calcul du plan d'exécution
- Sert à aérer le code

```
locals {  
  deployment_name = "${var.stack_name}-${var.deploy_region}"  
  env_name        = "${var.stack_name}-${var.deploy_env}"  
}
```

# Usage des locales

## Syntaxe HCL

- Les locales peuvent être injecté dans le reste du code Terraform
- Une librairie de fonction de transformation est disponible

```
resource "aws_instance" "web" {  
  name = "${local.deployment_name}-1"  
}
```

# Datasources

## Syntaxe HCL

- Récupère de l'information auprès d'API
- Permet de se raccorder à l'existant

```
data "aws_ami" "centos7_ami" {
  filter {
    product_code = "aw0evgkw8e5c1q413zgy5pjce"
    product_type = "marketplace"
  }
  most_recent = "true"
}

resource "aws_instance" "web" {
  ami = "${data.aws_ami.centos7_ami.id}"
}
```

# Outputs

## Syntaxe HCL

- Affichage en fin d'apply
- Consultable via la commande `terraform output`
- Consultable via Datasource de type `terraform_remote_state`

```
output "s3_arn" {  
  value = "${aws_s3_bucket.my_bucket.arn}"  
}
```

# Backends

## Syntaxe HCL

- Sauvegarde de l'état de l'infrastructure
- Configurable par CLI, HCL ou mixte
- Ressource partagée par l'équipe

```
terraform {  
  backend "s3" {  
    bucket = "mybucket"  
    key     = "path/to/my/key"  
    region = "us-east-1"  
  }  
}
```

# Mise en pratique



<https://gitlab.com/wescalefr/training-terraform>

15 minutes





# Terraform states



# Vérité

## Tfstates

- Fichier en JSON
- Le Tfstate contient les données détaillées de ce qui a été réalisé.
- Il sert de comparatif pour calculer le plan d'exécution.
- Contient les valeurs des outputs.

# Isolation

## Tfstates

- Isoler vos groupes de ressources par périmètre technique.
- Isoler vos groupes de ressources par périmètre de responsabilités.

# Partage

## Tfstates

- Les tfstates peuvent servir de datasources.
- Stocker vos tfstates dans des emplacements partagés.

# Keep & Share

## Tfstates

- **Perdre un tfstate est très embêtant !**
- **Ne pas partager un tfstate peut être handicapant !**
- **Les secrets sont stockés en dur dans le fichier de state.**



# Manipulation du fichier de state

## Tfstates

- Permet d'écraser une version distante (backends).
- Permet de récupérer une version en local.

```
terraform state push $PATH  
terraform state pull > local.tfstate
```

# Inspection de ressources

## Tfstates

- Consulter le détail du state

```
terraform show  
terraform state list  
terraform state show type.label
```

# Manipulation de ressources

## Tfstates

- Marquer des ressources à garder/reconstruire

```
terraform taint type.label  
terraform untaint type.label
```

# Manipulation de ressources

## Tfstates

- Permet de déplacer les ressources dans un state
- Permet de déplacer les ressources d'un state à un autre

```
terraform state mv
```



# Manipulation de ressources

## Tfstates

- Permet de déplacer les ressources dans un state
- Permet de déplacer les ressources d'un state à un autre

```
terraform state rm
```

# Mise en pratique



<https://gitlab.com/wescalefr/training-terraform>

15 minutes





# Modules

# Factoriser

## Modules

- Permet de cloisonner et factoriser du code
- Différentes sources

```
module "tfbackend" {  
  source = "git::https://github.com/aurelienmaury/terraform-aws-tfbackend.git"  
  deploy_region      = "eu-west-1"  
  bucket_tfstates_name = "wescalefr-organization"  
}
```

# Partager

## Modules

- Répertoire local
- Github
- Bitbucket
- Git générique
- Mercurial générique
- HTTP URL
- S3 Bucket
- <https://registry.terraform.io/>

# Structure

## Modules

- C'est une stack Terraform, ni plus ni moins.
- Variables et outputs définissent les entrées sorties possibles.

# Structure

## Modules

- Doit avoir au moins ces 3 fichiers :
  - main.tf
  - variables.tf
  - output.tf



# Quand les utiliser ?

## Modules

- Dès que vous détectez une duplication de code.
- Pour encapsuler une stack qui doit servir de composant normé.
- Toute stack Terraform est un potentiel module.

# Utiliser les outputs

## Modules

- Les outputs d'un module sont exportés à la stack appelante.

```
attribut = "${module.label.output_label}"
```

# Etude de cas

## Modules

- <https://github.com/claque2000/terraform-aws-tfbackend>

# Mise en pratique



<https://gitlab.com/wescalefr/training-terraform>

15 minutes





# Syntaxe HCL avancée

# count

## Syntaxe HCL avancée

- Multiplications de ressources identiques
- Ne fonctionne pas sur les appels aux modules
- Penser à la variable `${count.index}` ou au attributs `*_prefix`

```
resource "aws_instance" "web" {  
  count = 2  
  instance_type = "t2.micro"  
}
```

# Manipulations de listes

## Syntaxe HCL avancée

- `concat(list1, list2)`
- `contains(list, element)`
- `flatten(list of lists)`
- `index(list, element)`
- `join(delimiter, list)`
- `length(list)`
- `sort(list)`
- `liste.index`
- `list.*.truc`



# Manipulations de map

## Syntaxe HCL avancée

- `keys(map)`
- `values(map)`
- `lookup(map, "key", [default])`
- `map["key"]`
- `merge(map1, map2)`

# Manipulations de string

## Syntaxe HCL avancée

- `lower(string)`
- `upper(string)`
- `sha1(string)`
- `split(delimiter, string)`
- `uuid()`

# Calcul de CIDR

## Syntaxe HCL avancée

- cidrsubnet
- cidrhost
- cidrnetmask

# Arithmétique

## Syntaxe HCL avancée

- `min(num1, num2, ...)`
- `max(num1, num2, ...)`

# Opérateur ternaire

## Syntaxe HCL avancée

- Le très fameux... `CONDITION ? VALUE_WHEN_TRUE : VALUE_WHEN_FALSE`
- Seul moyen de faire des ressources de façon conditionnelle.

# Mise en pratique



<https://gitlab.com/wescalefr/training-terraform>

À demain !





# Training Terraform (le retour)







**Refresh**

# De quoi vous souvenez-vous ?

Refresh





**Terraform en équipe**

# Personas

## Terraform en équipe



Workspace owner



Workspace contributor



Organization architect

# Workspace owner

## Terraform en équipe

- Est responsable d'un ou plusieurs projets Terraform.
- Est le point d'approbation de toute modification de code Terraform.
- Est responsable de tout changement sur l'environnement de production.

# Workspace contributor

## Terraform en équipe

- Soumet des modification de l'infrastructure via le code.
- Peut déployer sur tous les environnements hors-prod pour tests.
- Ne peuvent impacter que les variables de leurs environnements.
- Utilisent Terraform en CLI ou au travers d'un CI pipeline

# Organization Architect

## Terraform en équipe

- Organise les périmètres techniques de chaque équipe.
- Définit les interfaces entre projets Terraform (outputs, nommage, ...).
- Définit les variables communes à tous les projets de l'organisation.
- Définit les bonnes pratiques de code.

# Cloisonner

## Tfstates

- Les **Contributors** doivent lire et écrire les tfstates hors-prod.
- Le **Owner** doit lire et écrire les tfstates de tout environnement.
- L'**Architect** doit s'assurer des règles d'accès équipes-tfstates.

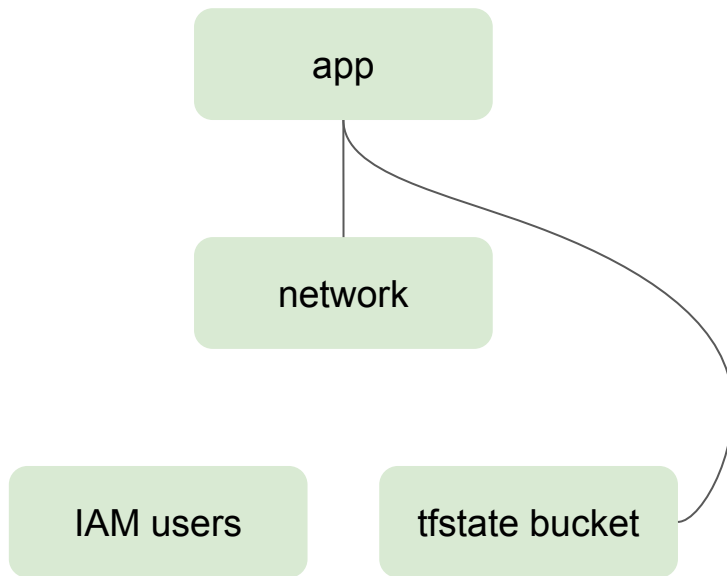




# Layering

# Séparation des préoccupations

## Layering



# Découpage

## Layering

Les périmètres techniques doivent être tracés :

- en connaissance de toutes les équipes
- en bonne intelligence
- en suivant la géographie des équipes responsables
- par l'Organization Architect.

# Tout est module

## Layering

- Vos inputs sont un contrat d'interface avec vos utilisateurs.
- Vos outputs sont un contrat d'interface avec vos utilisateurs.
- Toute stack Terraform est un module.

# Démo



15 minutes





# Workspaces

# Instanciación

## Workspaces

Les workspaces servent à obtenir plusieurs tfstates :

- pour une configuration unique de backend
- pour une stack Terraform unique

De base, vous travaillez dans le workspace implicite : “default”.



# Instanciación

## Workspaces

Fonctionne avec :

- AzureRM
- Consul
- GCS
- Local
- Manta
- S3

# Manipulation

## Workspaces

```
terraform workspace list
```

```
terraform workspace show
```

```
terraform workspace new foobar
```

```
terraform workspace select foobar
```

```
terraform workspace delete foobar
```

# Utilisation

## Workspaces

```
resource "aws_instance" "example" {  
    count = "${terraform.workspace == "default" ? 5 : 1}"  
    # ...  
}  
  
resource "aws_instance" "example" {  
    tags {  
        Name = "web - ${terraform.workspace}"  
    }  
    # ...  
}
```

# Mise en pratique



<https://gitlab.com/wescalefr/training-terraform>



**TP final**

# Mise en pratique



<https://gitlab.com/wescalefr/training-terraform>

**Aurélien Maury**  
Directeur Technique



aurelienmaury / yeswescall



aurelien.maury@wescall.fr



23 rue Taitbout 75009 Paris  
[www.wescall.fr](http://www.wescall.fr) | [blog.wescall.fr](http://blog.wescall.fr)