



Samuel ANTUNES
Consultant Ingénieur DevSecOps
OCTO Technology
Email : contact@samuelantunes.fr



ICEBREAKER

Pourquoi Terraform ?

- Qu'est-ce que Terraform ?
- Terraform et le DevOps
- Comment ça marche ?

Les premiers pas

- Installation et Configuration
- Le CLI

Le HCL

- Les bases de la syntaxe (resources)
- Rendre son code paramétrable
- Syntaxe avancée

Passage à l'échelle

- Les modules
- La gestion du state
- Les layers
- Panique à bord

“Terraform”

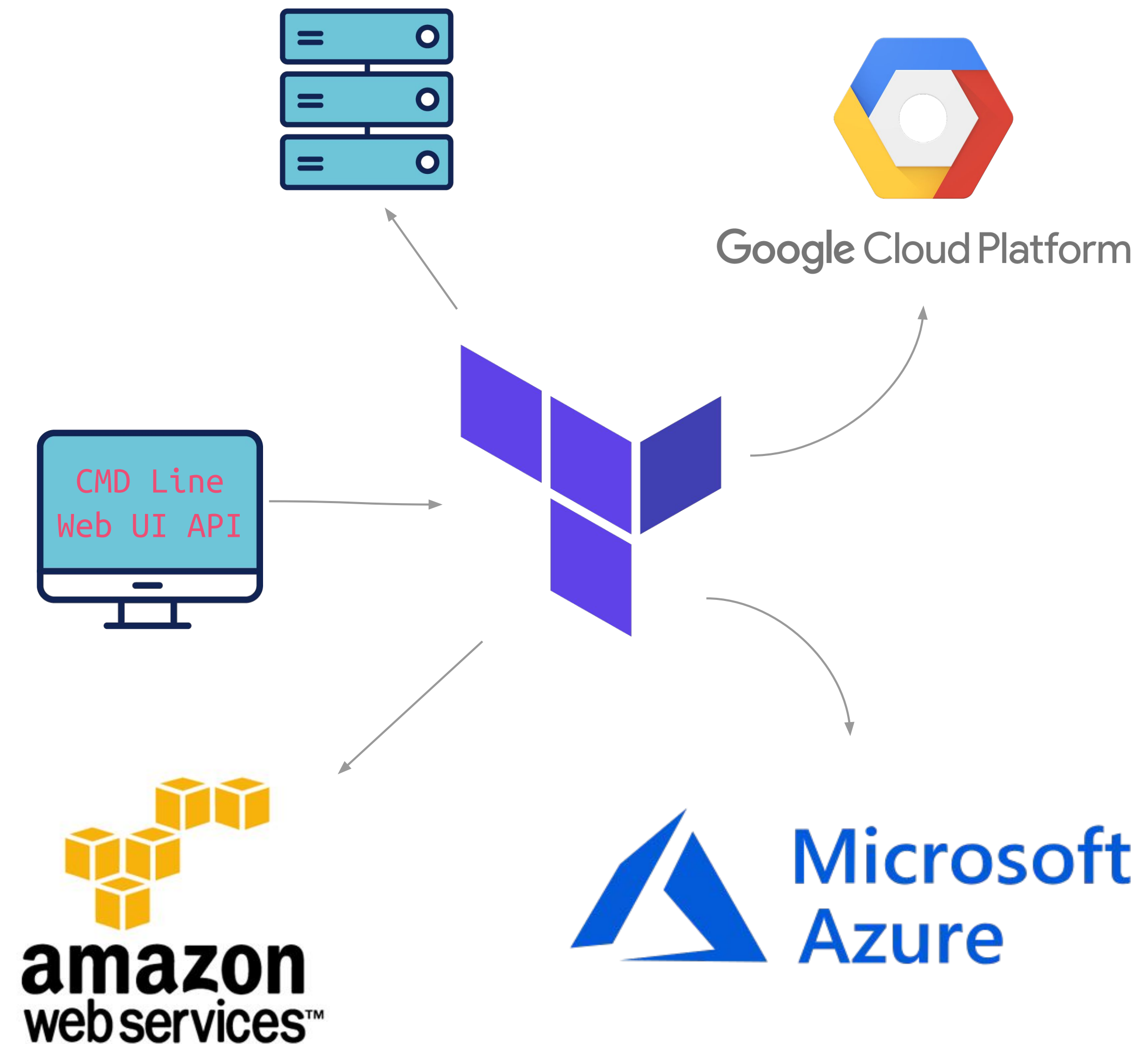


Un outil **d'orchestration** couplé avec un fournisseur cloud peut provisionner des serveurs, des base de données, des loads-balancers et tous autres éléments d'infrastructure

Il agit comme un chef d'orchestre


Par exemple, dans le cas d'une application :


- Création des composants réseau
- Déploiement d'un load balancer
- Mise en place d'instances à partir d'une image de l'application
- Ouverture des flux





UN PROJET OPENSOURCE


- Sur le Github d'HashiCorp
- Premier commit : 22/05/2014


 hashicorp / terraform


 Watch ▾ 1.1k


 Star 19.1k


 Fork 5.1k


 Code

 Issues 1,086

 Pull requests 165


 Actions


 Security


 Insights


Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned. <https://www.terraform.io/>


[go](#) [graph](#) [infrastructure-as-code](#) [terraform](#)


 24,864 commits


 92 branches


 131 releases


 1,333 contributors


 MPL-2.0


 Go 98.8%

 HTML 0.7%

 Shell 0.4%

 Makefile 0.1%

 HCL 0.0%

 Ruby 0.0%

Branch: master ▾


New pull request

Create new file


Upload files

Find file

Clone or download ▾

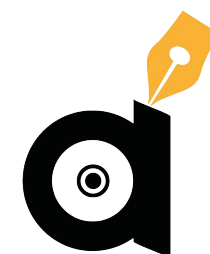
 pselle Merge pull request #23105 from hashicorp/f-tfignore ...

Latest commit d08daa2 1 hour ago

 .github

Revise our contributing/development documentation

2 days ago



PROVISION, SECURE, CONNECT, AND RUN

ANY INFRASTRUCTURE FOR ANY APPLICATION

PROVISION



Vagrant



Packer



Terraform

SECURE



Vault

CONNECT



Consul

RUN



Nomad

BUILD

TEST

PACKAGE

PROVISION

SECURE

MAINTAIN

DEPLOY

Seven elements of the modern Application Lifecycle

Terraform donne la capacité de gérer des **ressources** sur **différents providers** :



TERRAFORM PERMET DE :

- ⦿ Manipuler des composants d'infrastructure sur des clouds providers
 - > Créer une infrastructure réseaux sur aws
 - > Déployer un Kubernetes sur GCP

- ⦿ Interagir avec des produits d'infrastructure
 - > Insérer des secrets dans un Vault
 - > Déployer une application sur Rancher

- ⦿ Configurer des services d'infrastructure
 - > Créer un projet/groupe/user sur GitLab
 - > Initialiser une base de données PostgreSQL

ET DE MANIERE CONCRETE ?

Avant tout, un outil en **ligne de commande**

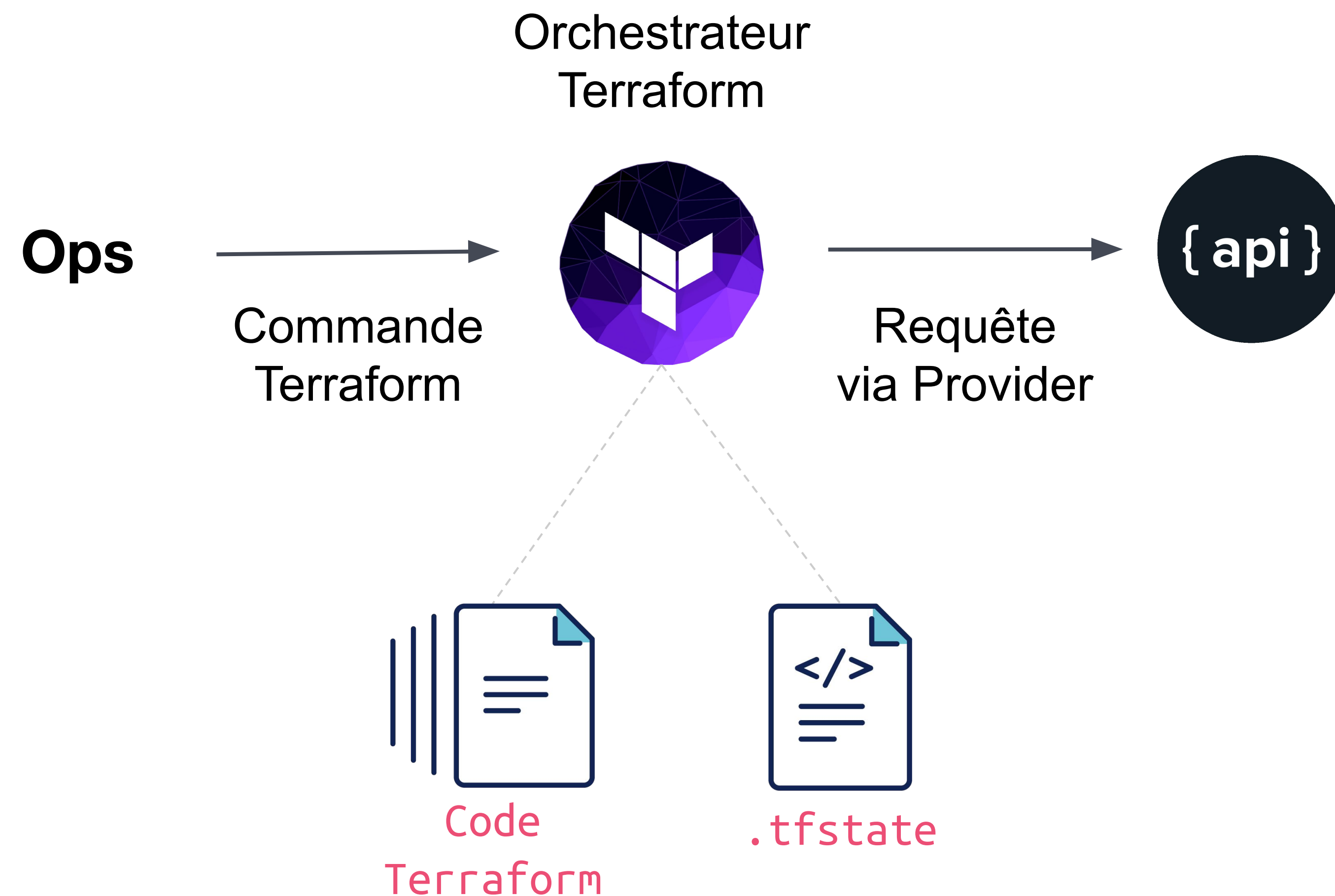
Terraform s'installe sur un **poste de travail** pouvant accéder au service à interroger

Terraform pilote toutes ses ressources:

- Via des appels d'**API** sur les différents providers
- Chaque provider est indépendant des autres
- Sans agent

Terraform lit les **fichiers de description** contenant les informations de configuration et ordonnance les appels





L'Orchestrateur dispose :

- Des flux réseau ouverts pour appeler les APIs
- Des credentials pour se connecter sur les APIs
- Le Binaire Terraform
- Le code Terraform

L'Orchestrateur produit:

- Un Fichier d'état: Le TfState

La mise en œuvre DevOps, c'est :

- une **culture de rapprochement** des Dev et des Ops*
- **de l'automatisation** : tout doit être automatisé de la création de ressources (serveurs, stockage) jusqu'au monitoring de production
- **de la mesure** : on doit être en capacité d'avoir des métriques techniques et métiers avec un objectif d'amélioration continue
- du **partage de connaissance**

* OPS : équipes opérationnelles / d'exploitation

Un outil 100% en phase avec la démarche DevOps

- ▷ **Automatisation**
 - > Création d'environnements réalisés sans intervention
 - > Moins d'erreurs liées aux actions manuelles
- ▷ **Répétabilité (ou idempotence)**
 - > Lancer deux fois Terraform sur un même provider avec les même fichiers de configuration produit le même résultat
- ▷ **Auditabilité**
 - > Capacité de connaître l'état des *resources* sur un provider avec la sortie de l'exécution de Terraform
- ▷ **Infrastructure as Code**
 - > Description de l'infrastructure sous forme de code
- ▷ **Intégration & déploiement continu**
 - > Participation au déploiement des applications
 - > Intégration aux outils classiques d'intégration continue (Jenkins, ...)

Pour être bien utilisé, il faut en plus

- ▷ Adopter des **pratiques issues du développement**
 - > Systématiquement gérer les fichiers de configuration de Terraform dans un **gestionnaire de sources** (Git, SVN...)
 - > Avoir recours à des pratiques de codage
 - Revue de code
 - Pair programming
 - Écriture de tests
 - Documentation
- ▷ Respecter les **principes d'écriture de code Terraform**
 - > Modularité / abstraction

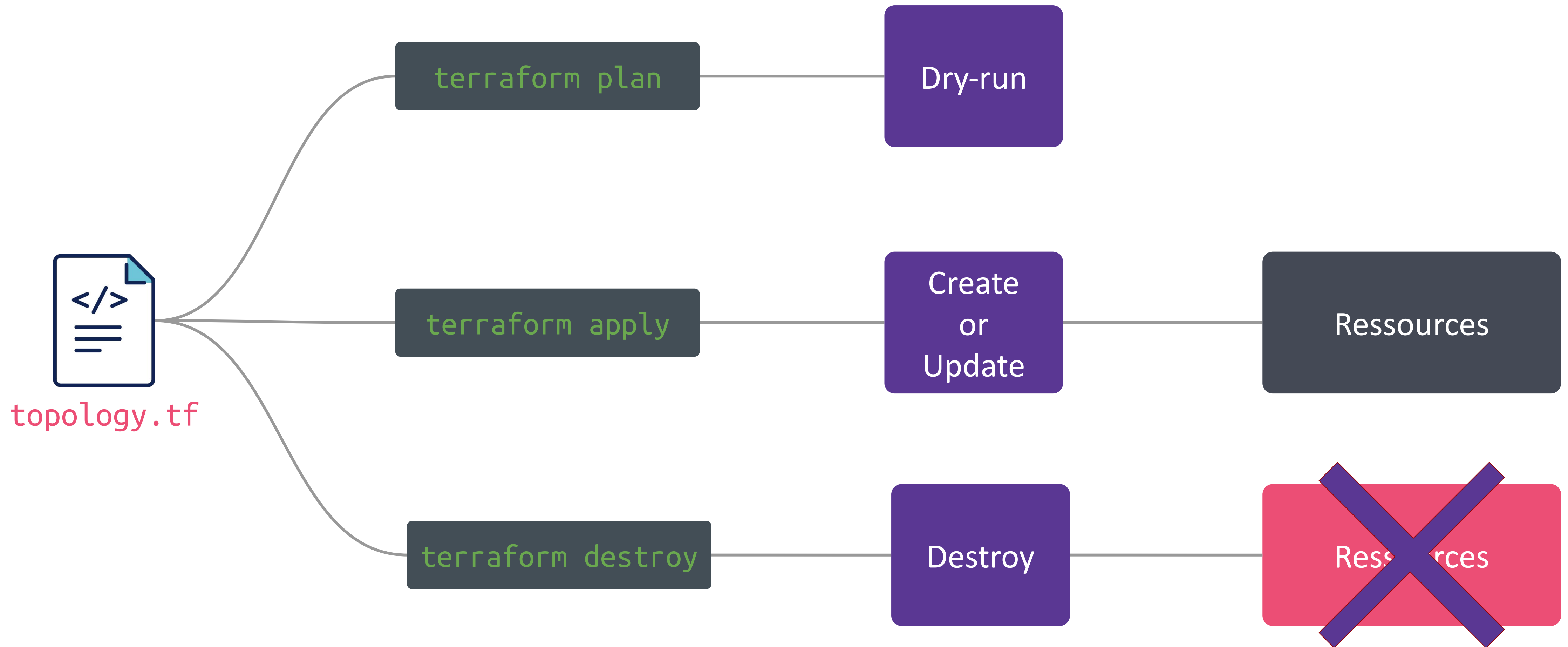


“The first time you hit that blue button that says ‘Launch Instance’, is the first technical debt you have accumulated.” - Soofi Safavi

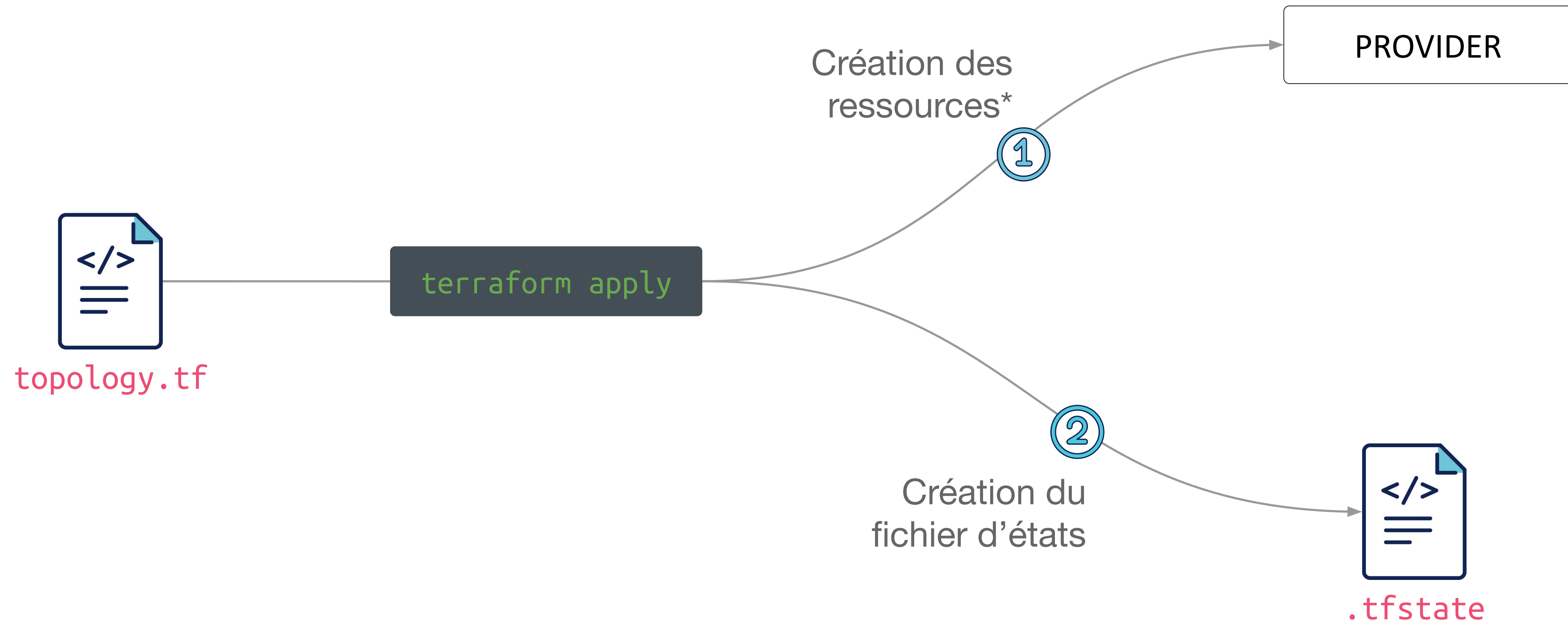
Launch Instance

- ⦿ Comment tester une machine créée à la main ?
- ⦿ Comment l'intégrer dans une CI ?
- ⦿ Quelle est sa configuration ?
- ⦿ Comment reconstruire la machine facilement ?

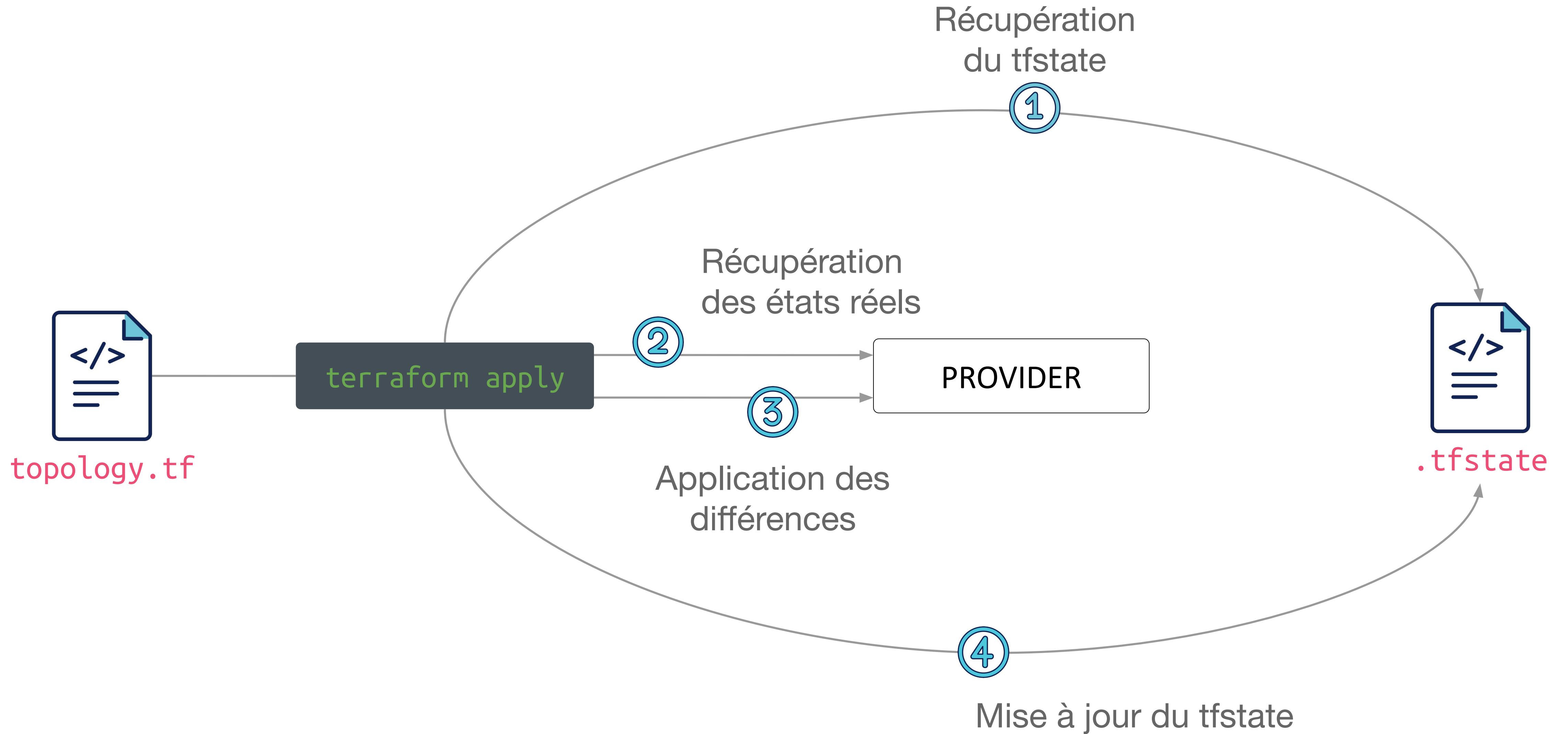
**STAND UP TO
TECHNICAL
DEBT**

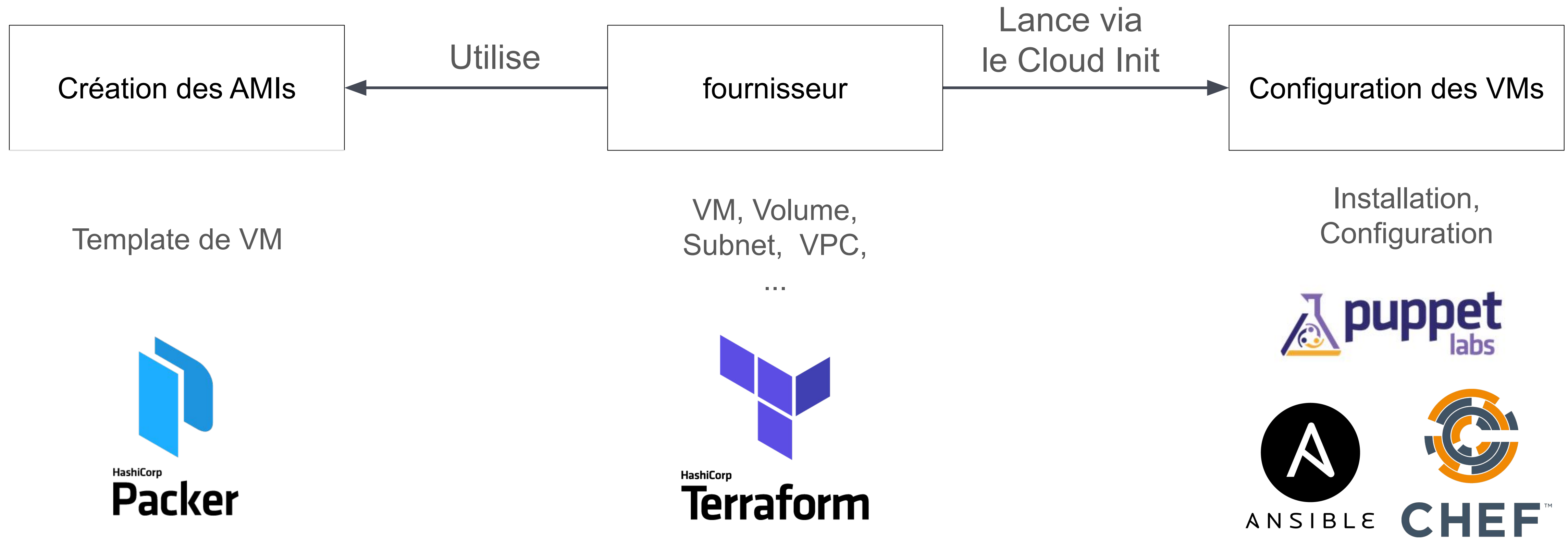


Les fichiers *.tf sont écrit en
HashiCorp Configuration Language



** Terraform construit un graphe pour ordonnancer lui-même les appels d'API*





“Les premiers
pas”



- ⦿ Les providers servent de wrapper aux API d'infrastructure (AWS, Azure ...)
- ⦿ Les providers sont externalisés du binaire terraform
- ⦿ Sources des providers : <https://github.com/terraform-providers>
- ⦿ Releases des providers : <https://releases.hashicorp.com>
- ⦿ La commande `providers` permet de lister les providers utilisés

Exemple :

```
$ terraform providers
.
├── provider.aws
├── module.database
│   └── provider.aws
├── module.webserver
│   └── provider.aws
```

```
provider "aws" {
  region    = "us-east-2"
}
```

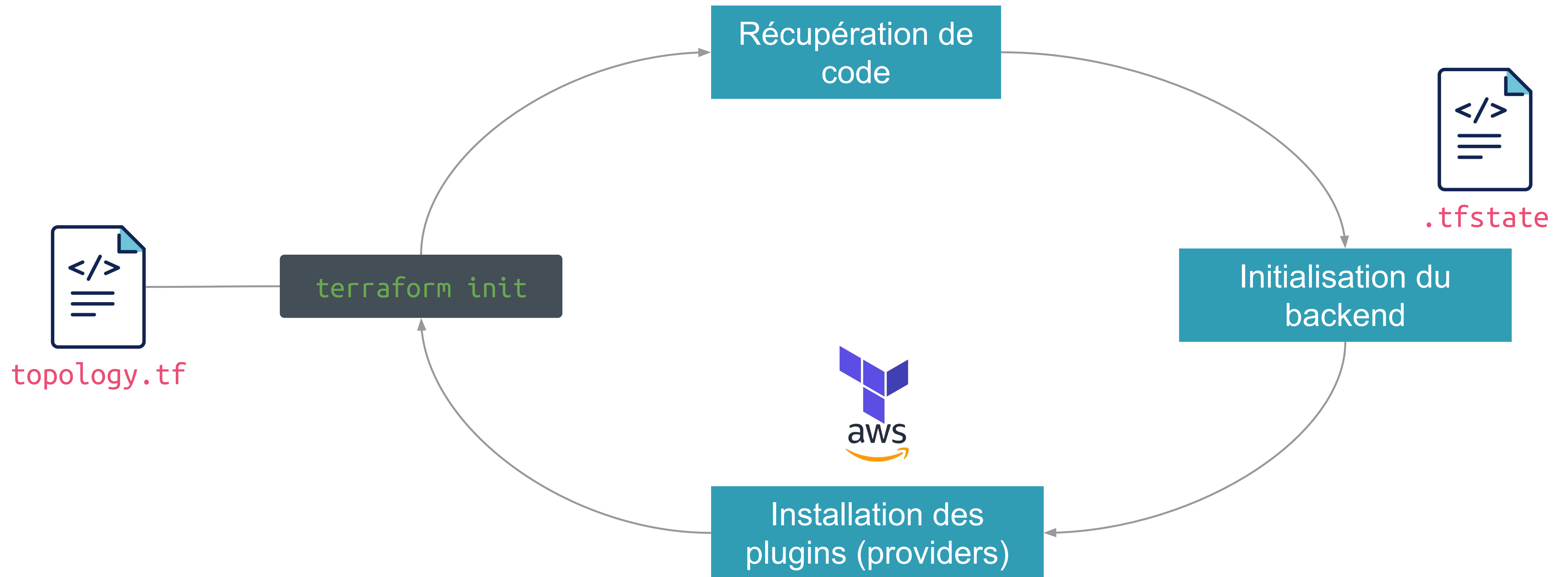


git clone

https://5AIW:5LeHYHdZ_Eyffdkzs61o@gitlab.com/santunes-formations/terraformed.git

- ⦿ Exemple de fichier `topology.tf` décrivant :
 - > La création d'une instance AWS.
 - > L'output de l'id de l'instance.

```
resource "aws_instance" "example" {  
    ami          = "ami-760aaa0f"  
    instance_type = "t2.micro"  
}  
  
output "id" {  
    value = aws_instance.example.id  
}
```

Il est préférable de fixer la version des providers en production pour éviter les *breaking changes*

□ Permet de **visualiser le plan d'exécution** sans appliquer de changement sur l'infrastructure managée par terraform.

□ Mode '**dry-run**'.

□

Pas d'appel aux APIs d'AWS pour créer de nouveau(x) object(s)

```
$ terraform plan
```

```
-----
-

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
```

```
+ create
```

```
Terraform will perform the following actions:
```

```
# aws_instance.example will be created
```

```
+ resource "aws_instance" "example" {
+   ami                = "ami-760aaa0f"
+   arn                = (known after apply)
+   associate_public_ip_address = (known after apply)
+   availability_zone   = (known after apply)
+   cpu_core_count      = (known after apply)
+   cpu_threads_per_core = (known after apply)
+   get_password_data    = false
+   host_id             = (known after apply)
+   id                 = (known after apply)
+   instance_state      = (known after apply)
+   instance_type       = "t2.micro"
```

```
...
```

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

```
-----
-
```

(known after apply) => remplacé par AWS lors de l'appel d'API

- ⦿ Exemple :
 - > Création d'1 instance 'aws_instance'.

①

```
resource "aws_instance" "example" {
  ami           = "ami-760aaa0f"
  instance_type = "t2.micro"
  tags {
    Name = "default"
  }
}

output "id" {
  value = aws_instance.example.id
}
```

②

```
$ terraform plan
```

```
-----
-

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
```

```
+ create
```

```
Terraform will perform the following actions:
```

```
# aws_instance.example will be created
+ resource "aws_instance" "example" {
  + ami           = "ami-760aaa0f"
  + arn           = (known after apply)
  + associate_public_ip_address = (known after apply)
  + availability_zone           = (known after apply)
  + cpu_core_count             = (known after apply)
  + cpu_threads_per_core       = (known after apply)
  + get_password_data          = false
  + host_id                   = (known after apply)
  + id                       = (known after apply)
  + instance_state            = (known after apply)
  + instance_type             = "t2.micro"
  ...
}
```

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

- ⦿ Permet d'**appliquer le plan d'exécution** pour créer ou mettre à jour l'infrastructure managée par Terraform.
- ⦿ Exemple:
 - > Ressource 'aws_instance' créée.

1 seul Ctrl+C => déclenche un graceful shutdown
Plusieurs Ctrl+C => corruption du tfstate

```
$ terraform apply
```

```
...  
plan is displayed here  
...
```

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

```
Do you want to perform these actions?  
  Terraform will perform the actions described above.  
  Only 'yes' will be accepted to approve.
```

```
Enter a value: yes
```

```
aws_instance.example: Creating...  
aws_instance.example: Still creating... [10s elapsed]  
aws_instance.example: Still creating... [20s elapsed]  
aws_instance.example: Still creating... [30s elapsed]  
aws_instance.example: Creation complete after 32s  
[id=i-033910abf29bd2be0]
```

```
Apply complete! Resources: 1 added, 0 changed, 0  
destroyed.
```

```
Outputs:
```

```
id = i-033910abf29bd2be0
```

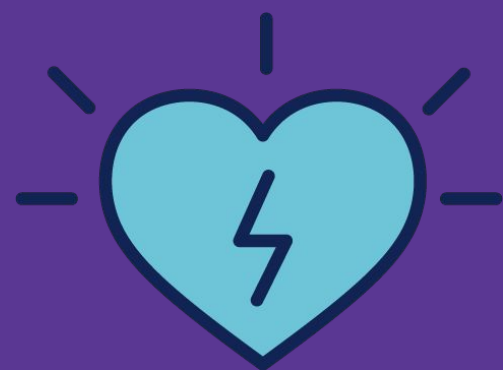

*Fichier d'état
de l'infrastructure déployée*



.tfstate

exemple

```
{
  "version": 4,
  "terraform_version": "0.12.3",
  "serial": 2,
  "lineage": "acfbaf59-896c-a101-5f06-09126a76f669",
  "outputs": {
    "id": {
      "value": "i-0f33cf3fda3deda9f",
      "type": "string"
    }
  },
  "resources": [
    {
      "mode": "managed",
      "type": "aws_instance",
      "name": "example",
      "provider": "provider.aws",
      "instances": [
        {
          "schema_version": 1,
          "attributes": {
            "ami": "ami-760aaa0f",
            "arn":
"arn:aws:ec2:eu-west-1:218232161888:instance/i-0f33cf3fda3deda9f",
            "associate_public_ip_address": true,
            "availability_zone": "eu-west-1b",
            "cpu_core_count": 1,
            ...
          }
        }
      ]
    }
  ]
}
```



Corruption du Tfstate
panique à bord !

- ⦿ Exemple:
 - > Modification du tag 'Name' de l'instance 'aws_instance'.

①

```
resource "aws_instance" "example" {
  ami           = "ami-760aaa0f"
  instance_type = "t2.micro"
  tags {
    Name = "formation"
  }
}

output "id" {
  value = aws_instance.example.id
}
```

②

```
$ terraform plan
```

```
-----

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
```

```
~ update in-place
```

```
Terraform will perform the following actions:
```

```
# aws_instance.example will be updated in-place
~ resource "aws_instance" "example" {
    ami           = "ami-760aaa0f"
    arn           = "arn::instance/i-0fc40c7a5325128e5"
    associate_public_ip_address = true
    availability_zone = "eu-west-1b"
    cpu_core_count  = 1
    cpu_threads_per_core = 1
    disable_api_termination = false
    ebs_optimized    = false
    get_password_data = false
    id               = "i-0fc40c7a5325128e5"
    instance_state   = "running"

    ...

    ~ tags = {
      ~ "Name" = "default" -> "formation"
    }

    ...
}
```

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

-----
```

- ⦿ Exemple:
 - > Modification du type d'ami de l'instance 'aws_instance'.

①

```
resource "aws_instance" "example" {  
  ami          = "ami-02ace471"  
  instance_type = "t2.micro"  
  tags {  
    Name = "formation"  
  }  
}  
  
output "id" {  
  value = aws_instance.example.id  
}
```



```
$ terraform plan
```

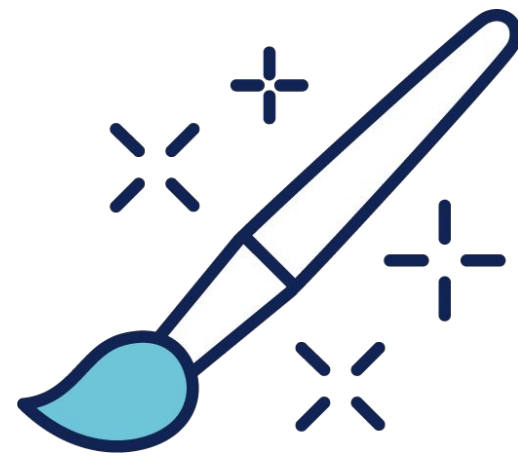
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:

-/+ destroy and then create replacement

Terraform will perform the following actions:

```
# aws_instance.example must be replaced
-/+ resource "aws_instance" "example" {
  ~ ami                  = "ami-02ace471" -> "ami-760aaa0f" # forces replacement
  ~ arn                  = "arn:aws:ec2:eu-west-1:218232161888:instance/i-0fc40c7a5325128e5" -> (known after apply)
  ~ associate_public_ip_address = true -> (known after apply)
  ~ availability_zone       = "eu-west-1b" -> (known after apply)
  ~ cpu_core_count          = 1 -> (known after apply)
  ~ cpu_threads_per_core    = 1 -> (known after apply)
  - disable_api_termination  = false -> null
  - ebs_optimized            = false -> null
    get_password_data        = false
  + host_id                = (known after apply)
  ~ id                     = "i-0fc40c7a5325128e5" -> (known after apply)
  ~ instance_state          = "running" -> (known after apply)
    instance_type            = "t2.micro"
  ...
```

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



- ⦿ Permet de **corriger le style et le format** des fichiers de configuration terraform '*.tf'.

- ⦿ Exemple:
 - > Correction de l'indentation
 - > Alignement des "="

①

```
resource "aws_instance" "example" {  
  ami = "ami-760aaa0f"  
  instance_type = "t2.micro"  
}  
  
output "id" {  
  value = aws_instance.example.id  
}
```

②

```
$ terraform fmt  
topology.tf
```

③

```
resource "aws_instance" "example" {  
  ami           = "ami-760aaa0f"  
  instance_type = "t2.micro"  
}  
  
output "id" {  
  value = aws_instance.example.id  
}
```


- ⦿ Permet de détruire l'infrastructure managée par terraform.
- ⦿ Exemple:
 - > Ressource 'aws_instance' détruite.
- ⦿ Équivalent à la suppression du code et d'un apply

Le Tfstate n'est pas supprimé après un destroy

```
$ terraform destroy
aws_instance.example: Refreshing state... [id=i-0fc40c7a5325128e5]

Terraform will perform the following actions:

  # aws_instance.example will be destroyed
  ...

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

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

aws_instance.example: Destroying... [id=i-0fc40c7a5325128e5]
aws_instance.example: Still destroying... [id=i-0fc40c7a5325128e5, 10s elapsed]
aws_instance.example: Still destroying... [id=i-0fc40c7a5325128e5, 20s elapsed]
aws_instance.example: Destruction complete after 29s

Destroy complete! Resources: 1 destroyed.
```

Terraform CLI

EXPAND ALL | FILTER

› Configuration Language

› Commands (CLI)

› Import

› State

▼ Providers

- Major Cloud
- Cloud
- Infrastructure Software
- Network
- VCS
- Monitor & System Management
- Database
- Misc.
- Community

› Provisioners

› Modules

› Backends

› Plugins

Providers

Terraform is used to create, manage, and update infrastructure resources such as virtual machines, switches, containers, and more. Almost any infrastructure type can be represented in Terraform.

A provider is responsible for understanding API interactions and exposing them in Terraform. Providers are available for many cloud providers (e.g. Amazon, Alibaba Cloud, AWS, GCP, Microsoft Azure, OpenStack), PaaS (e.g. Heroku, Netlify), and other services (e.g. DNSimple, CloudFlare).

Use the navigation to the left to find available providers by type or scroll through the list below.

- [ACME](#)
- [Akamai](#)
- [Alibaba Cloud](#)
- [Archive](#)
- [Arukas](#)
- [Avi Vantage](#)
- [Aviatrix](#)
- [AWS](#)
- [Azure](#)
- [Azure Active Directory](#)
- [GitLab](#)
- [Google Cloud Platform](#)
- [Grafana](#)
- [Gridscale](#)
- [Hedvig](#)
- [Helm](#)
- [Heroku](#)
- [Hetzner Cloud](#)
- [HTTP](#)
- [HuaweiCloud](#)

Information sur le CLI

Documentation sur les
différents providers

<https://www.terraform.io/docs/>

```
git clone  
https://5AIW:5LeHYHdZ_Eyffdkzs61o@gitlab.com/santunes-formations/terraformed.git
```

“Le HCL”



- ⦿ Langage de configuration
- ⦿ Langage utilisé par la plupart des produits Hashicorp
 - > Terraform
 - > Consul
 - > Vault
- ⦿ Compatible JSON
 - > Identique
 - `"String"`
 - `Number`
 - `[list]`
 - `{hash}`
 - `Boolean`
 - > **Plus flexible**
 - `#` Commentaire
 - `/*` Autre commentaire `*/`

Pour s'adapter aux **spécificités** des providers, les objets sont **différents** entre chaque providers.

Création d'une instance sur AWS

```
resource "aws_instance" "web" {  
  ami          = data.aws_ami.ubuntu.id  
  instance_type = "t2.micro"  
  tags {  
    Name = "HelloWorld"  
  }  
}
```

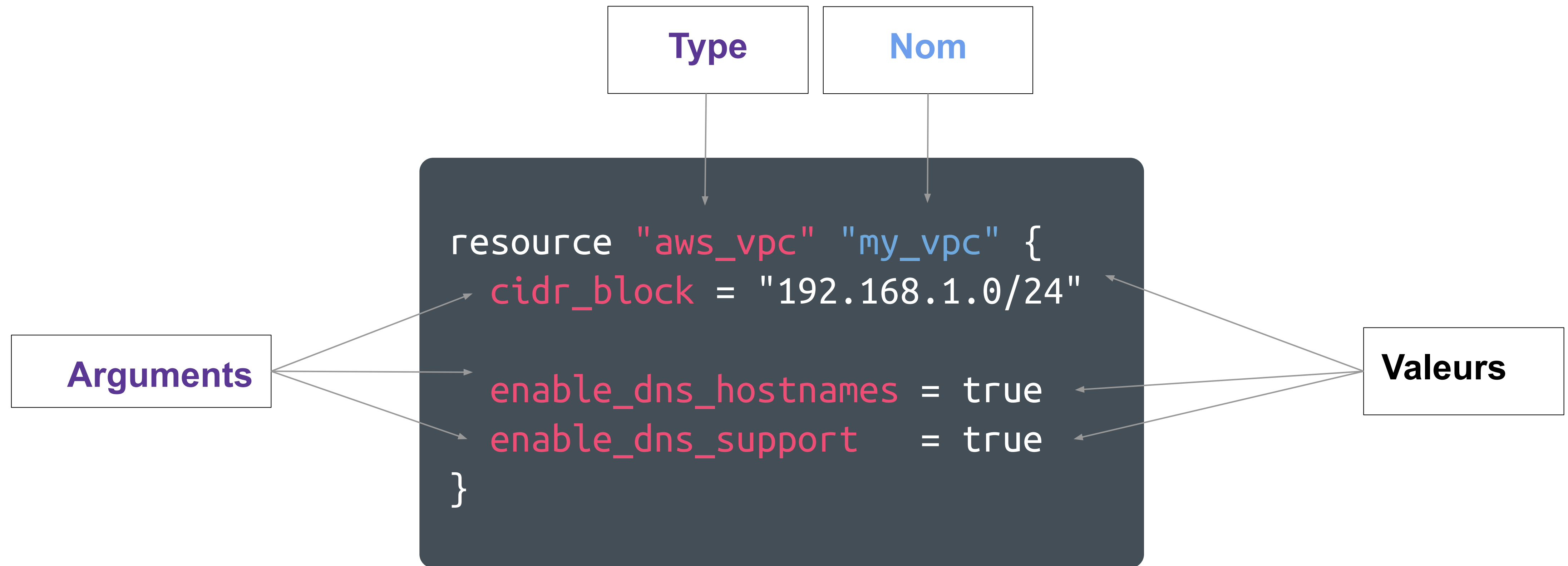


Création d'une instance sur GCE

```
resource "google_compute_instance" "default" {  
  name          = "test"  
  machine_type  = "n1-standard-1"  
  zone          = "us-central1-a"  
  tags          = ["foo", "bar"]  
  disk {  
    image = "debian-cloud/debian-0"  
  }  
}
```



*Une “resource” est un **objet déployé** sur le provider cible*



- ⦿ La meilleure façon de connaître le provider (resources, datasources...) est d'ouvrir la documentation
- ⦿ La documentation contient :
 - > La description
 - > Des exemples d'implémentation
 - > Les arguments (en entrée)
 - > Les attributs (en sortie)
 - > Des remarques par rapport aux comportements sur le provider

Resource: aws_vpc

JUMP TO SECTION ▾

Provides a VPC resource.

Example Usage

Basic usage:

```
resource "aws_vpc" "main" {  
  cidr_block = "10.0.0.0/16"  
}
```

Basic usage with tags:

```
resource "aws_vpc" "main" {  
  cidr_block      = "10.0.0.0/16"  
  instance_tenancy = "dedicated"  
  
  tags = {  
    Name = "main"  
  }  
}
```

Argument Reference

The following arguments are supported:

- `cidr_block` - (Required) The CIDR block for the VPC.
- `instance_tenancy` - (Optional) A tenancy option for instances launched into the VPC
- `enable_dns_support` - (Optional) A boolean flag to enable/disable DNS support in the VPC. Defaults true.

ATTRIBUTS DES RESSOURCES

- ⦿ Terraform expose des attributs pour chaque resource
- ⦿ Les attributs peuvent être utilisés directement (*) dans la déclaration d'autres ressources
 - > `<type>.<nom>.<attribut>` à partir de Terraform 0.12
 - > `"${<type>.<nom>.<attribut>}"` interpolation dans une string et avant Terraform 0.12
- ⦿ Exemple : Création d'un subnet passant en argument l'attribut "Id" de son VPC

Type Nom

↓ ↓

```
resource "aws_vpc" "my_vpc" {  
  cidr_block = "192.168.1.0/24"  
  
  enable_dns_hostnames = true  
  enable_dns_support   = true  
}
```

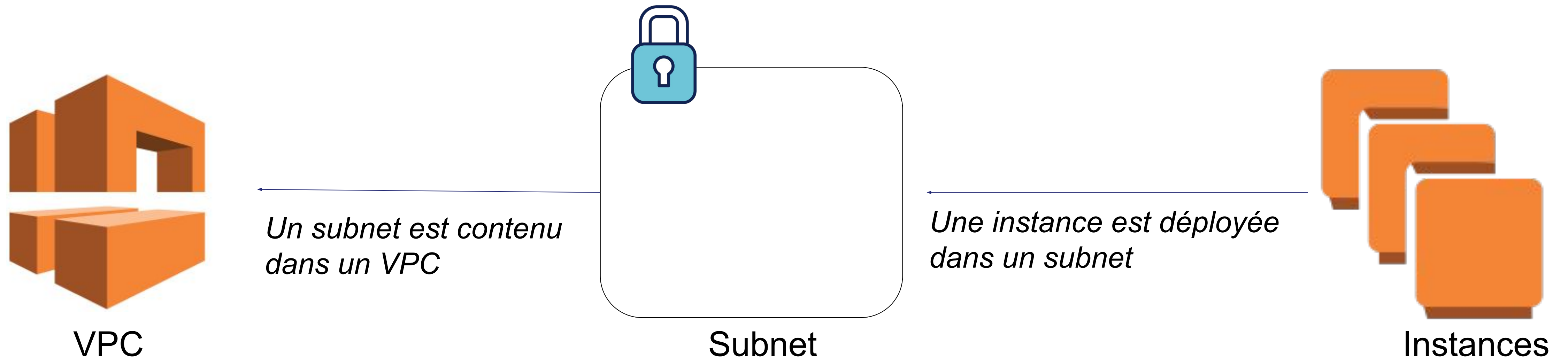
```
resource "aws_subnet" "subnet1" {  
  vpc_id      = aws_vpc.my_vpc.id  
  cidr_block = "192.168.1.0/25"  
}
```

```
resource "aws_subnet" "subnet1" {  
  vpc_id      = "${aws_vpc.my_vpc.id}"  
  cidr_block = "192.168.1.0/25"  
}
```

(*): (directement = sans datasource) Uniquement au sein d'une même layer

GRAPHE DE DEPENDANCE DES RESSOURCES

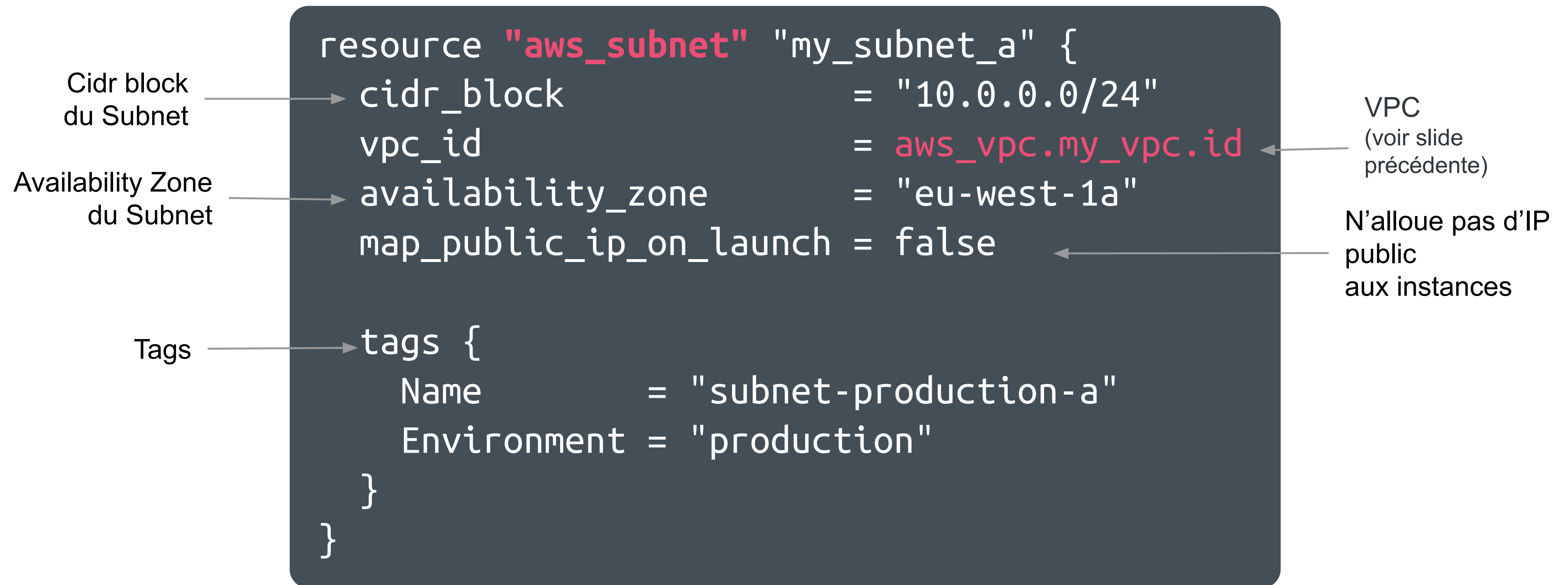
- ⦿ Certaines ressources AWS ont comme dépendance d'autres ressources AWS
 - > Un subnet a besoin d'un VPC
 - + Pour créer le subnet, on a besoin de connaître l'id du VPC
 - > Une instance a besoin d'un subnet
 - + Pour créer une instance, on a besoin de connaître l'id du subnet dans lequel l'instance sera déployée




```
resource "aws_vpc" "my_vpc" {  
  cidr_block = "10.0.0.0/16"  
  
  tags {  
    Name          = "vpc-production"  
    Environment   = "production"  
  }  
}
```

Tags

Range IP



Nom
(préfix + random)

Tags

```
resource "aws_security_group" "my_security_group" {  
  name_prefix = "my_security_group"  
  vpc_id      = aws_vpc.my_vpc.id  
  
  tags {  
    Environment = "production"  
    Type        = "fw-ssh"  
  }  
}
```

VPC
(voir slide
précédente)

À ce stade, le Security Group n'a pas de règle. Il est important de les mettre à part pour la gestion des états et gérer plus finement le cycle de vie.

EXEMPLE : CREATION D'UNE REGLE DANS LE SECURITY GROUP

Type de la règle

- Ingress = flux entrant
- Egress = flux sortant

Security Group
(voir slide
précédente)

Port(s)

Ranges d'IPs
autorisées

```
resource "aws_security_group_rule" "ssh_open_rule" {  
  security_group_id = aws_security_group.my_security_group.id  
  type              = "ingress"  
  from_port         = "22"  
  to_port           = "22"  
  protocol          = "tcp"  
  cidr_blocks       = ["0.0.0.0/0"]  
}
```

Plutôt que de référencer un bloc d'IPs autorisées, il est possible de spécifier un Security Group.

```
resource "aws_security_group" "my_security_group" {
  name_prefix = "my_security_group"
  vpc_id      = aws_vpc.my_vpc.id

  ingress {
    from_port = 22
    to_port   = 22
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

```
resource "aws_security_group" "my_security_group" {
  name_prefix = "my_security_group"
  vpc_id      = aws_vpc.my_vpc.id
}

resource "aws_security_group_rule" "ssh_open_rule" {
  security_group_id = aws_security_group.my_security_group.id
  type              = "ingress"
  from_port         = 22
  to_port           = 22
  protocol          = "tcp"
  cidr_blocks       = ["0.0.0.0/0"]
}
```

Les 2 modèles ne peuvent pas être utilisés en même temps

Image de
l'instance

Type
d'instance

Tags

Subnet
(voir slides précédentes)

Security Group
(voir slide précédente)

```
resource "aws_instance" "my_instance" {  
  ami           = "ami-760aaa0f"  
  instance_type = "t2.micro"  
  subnet_id     = aws_subnet.my_subnet_a.id  
  vpc_security_group_ids = [aws_security_group.my_security_group.id]  
  
  tags = {  
    Name     = "my-instance"  
    Role     = "Web"  
  }  
}
```

git clone

https://5AIW:5LeHYHdZ_Eyffdkzs61o@gitlab.com/santunes-formations/terraformed.git

Types simple:


- *string* ex: "example"
- *number* ex: 123
- *bool* ex: true

Type composé :

- *list(<TYPE>)* ex: *list(string)*
- *set(<TYPE>)* ex: *set(string)*
- *map(<TYPE>)* ex: *map(any)*

Type structuré:

- object
- tuple



```
variable "team" {  
  type      = "string"  
  default    = "toto"  
  description = "Team name"  
}
```

Valeur par défaut
(optionnel)La description de
la variable

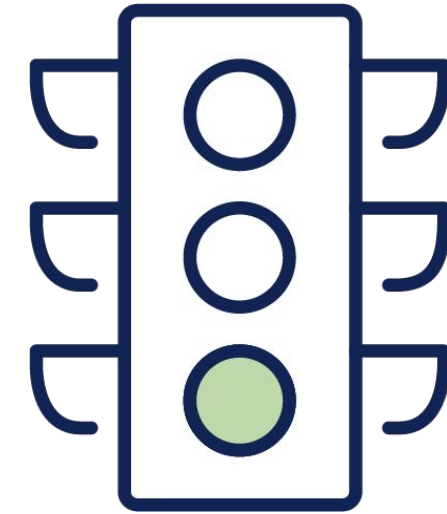
Conversion automatique sur les types simples
"true" => true, 15 => "15", ...

type = any ou type = list(any)
Any = n'importe quel type est acceptable

```
variable "instance_type" {
  type    = "string"
  default = "t2.micro"
}

variable "project_id" {
  type    = "number"
  default = null
}

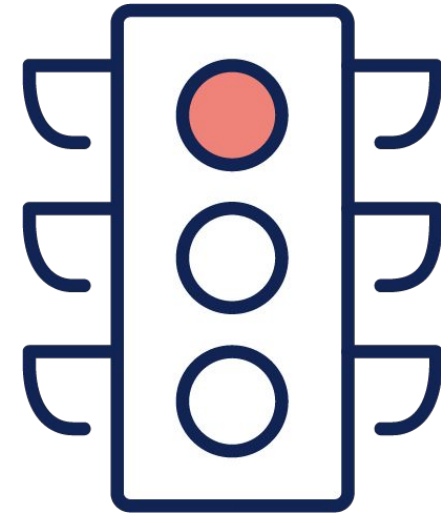
variable "tags" {
  type = "map(string)"
  default = {
  }
}
```



```
project_id = 734
tags       = {
  "app"      = "blog"
  "component" = "nginx"
  "service"  = "front"
}
```



```
instance_type = "t2.medium"
tags          = {
  "app"      = "blog"
  "component" = "nginx"
  "service"  = "front"
}
```



```
project_id = "734"
tags       = {
  "app"      = "blog"
  "component" = "nginx"
  "service"  = "front"
}
```

```
instance_type = "t2.medium"
tags          = "toto"
```

La valeur **null** correspond à une absence de valeur et la valeur par défaut est utilisé

Déclaration & assignation :

```
variable "dns_records" {  
  description = "List of DNS records"  
  type = list(object({  
    dns_record_name      = string  
    dns_record_type      = string  
    dns_record_ttl       = number  
    dns_record_ips       = list(string)  
  }))  
}
```

```
dns_records = [{  
  dns_record_name      = "toto.example.com"  
  dns_record_type      = "A"  
  dns_record_ttl       = 300  
  dns_record_ips       = ["1.1.1.1"]  
},  
{  
  dns_record_name      = "tata.example.com"  
  dns_record_type      = "A"  
  dns_record_ttl       = 300  
  dns_record_ips       = ["8.8.8.8"]  
}]
```

Toutes les entrées DNS sont présentes dans la même liste.

Command line

```
$ terraform apply -var 'instance_type=t2.large'
-var-file ../env.tfvars
```

*.auto.tfvars

```
instance_type = "t2.medium"
```

terraform.tfvars

```
instance_type = "t2.small"
```

VAR ENV

```
TF_VAR_instance_type = "t2.micro"
```

Valeur par défaut

```
variable "instance_type" {
  type = "string"
  default = "t2.nano"
}
```

Précédence

+ prioritaire

- prioritaire

Déclaration & assignation :

```
variable "instance_type" {  
  type    = string  
  default = "t2.micro"  
}
```

Déclaration :

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

Assignation :

```
instance_type = "t2.micro"
```

terraform.tfvars

Utilisation dans la Topology :

```
resource "aws_instance" "my_instance" {  
  ami           = "ami-760aaa0f"  
  instance_type = var.instance_type  
}
```

Déclaration & assignation :

```
variable "instance_types" {  
  type      = list(string)  
  default = ["t2.micro", "t2.medium", "t2.large"]  
}
```

```
variable "instance_infos" {  
  type      = map(string)  
  default = {  
    name      = "web"  
    user_data = "apt install nginx"  
  }  
}
```

Utilisation dans la Topology :

```
resource "aws_instance" "my_instance" {  
  ami           = "ami-760aaa0f"  
  instance_type = var.instance_types[0]  
  user_data     = var.instance_infos["user_data"]  
  tags = {  
    Name = var.instance_infos.name  
  }  
}
```

- Comme les variables mais ne peuvent pas être surchargés par un tfvars ou par la ligne de commande
- Permettent de faire de l'interpolation parmi les variables

```
variable "tp" {  
  type    = string  
  default = "tp42"  
}
```

Invalide

```
variable "bdd_name" {  
  type    = string  
  default = "${var.tp}-bdd"  
}
```

```
data aws_ami "tp_ami" {  
  name_regex = var.bdd_name  
}
```

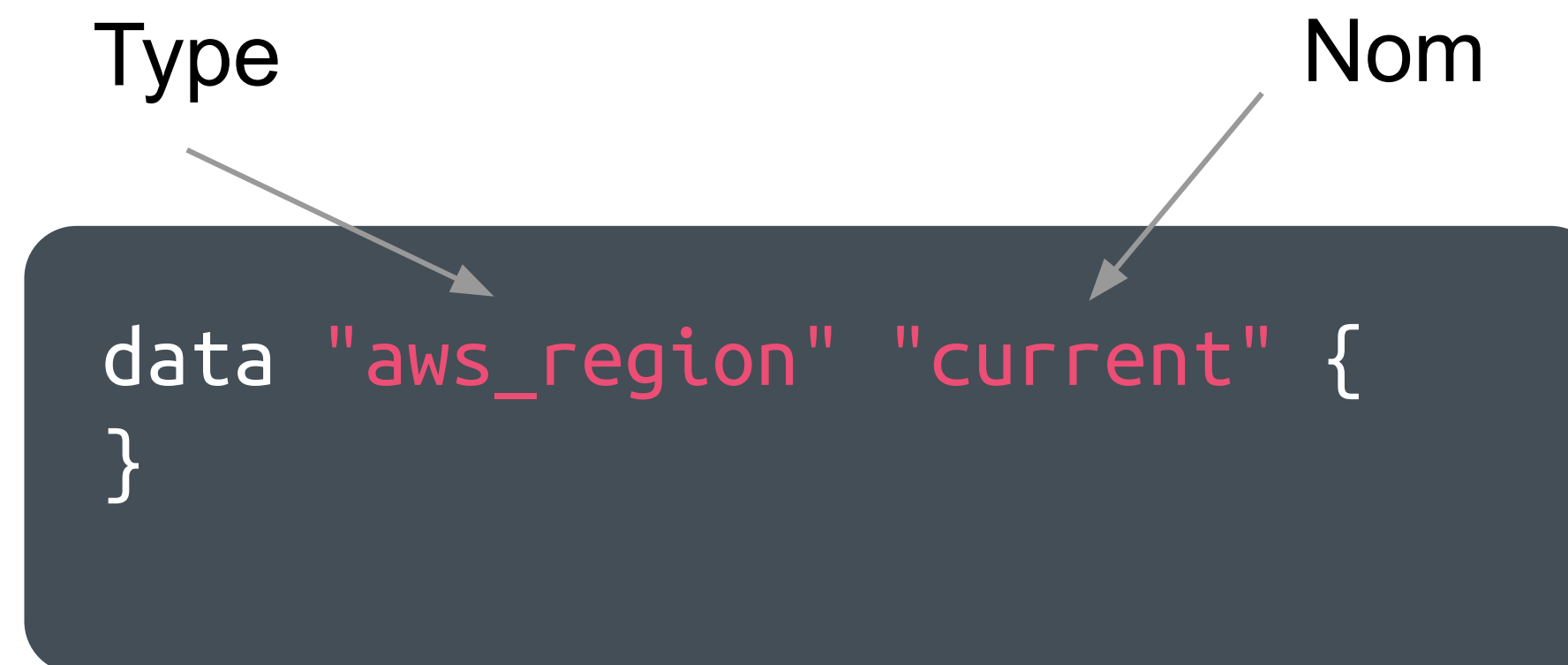
Valide

```
locals {  
  bdd_name = "${var.tp}-bdd"  
  web_name = "${var.tp}-web"  
}
```

```
data aws_ami "tp_ami" {  
  name_regex = local.bdd_name  
  owners     = ["self"]  
}
```

DATASOURCE - DECLARATION

- ⦿ Une datasource permet de récupérer des informations
 - > non managé par Terraform (ex : region, ami ...)
 - > managé par Terraform (ex : resource d'un autre tfstate)
- ⦿ Exemple : Récupération du nom de la région



```
data "aws_region" "current" {  
}
```

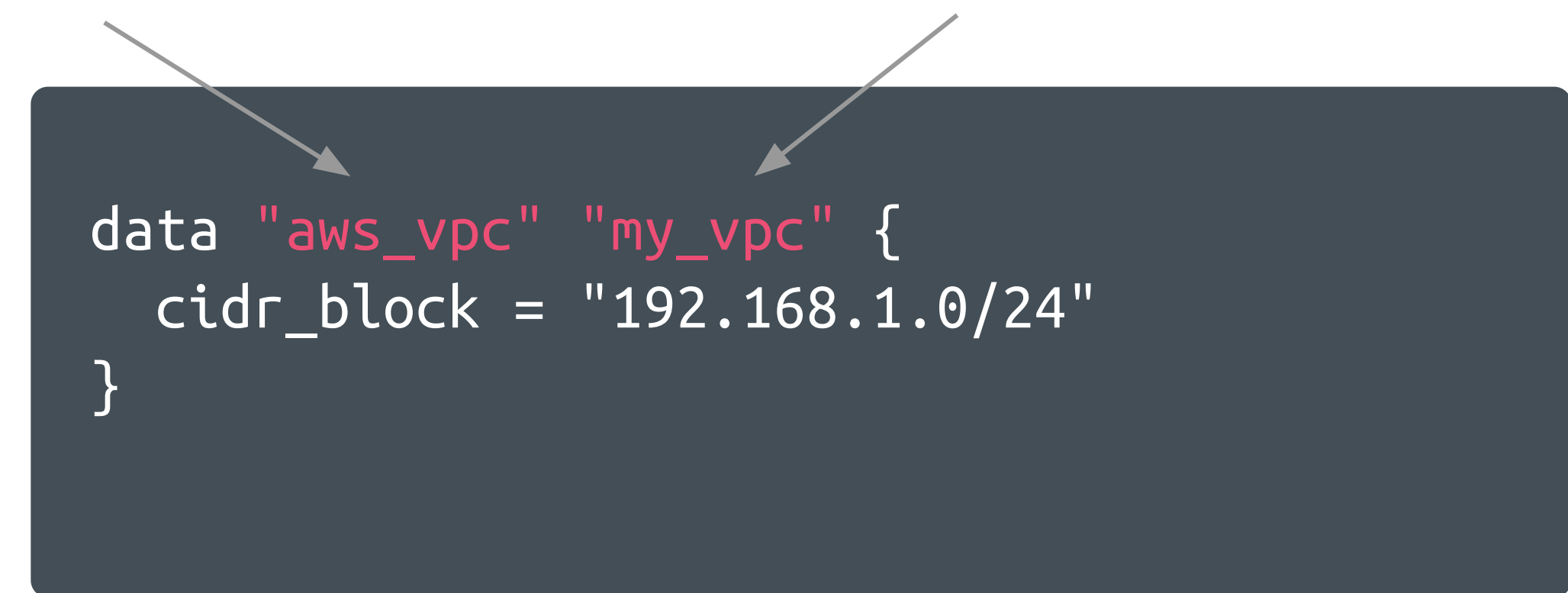
Les filtres doivent permettre de récupérer **1 unique** résultat sinon data retourne une erreur.

ATTRIBUTS DE DATASOURCE

- ⦿ Une datasource expose les attributs de l'objet récupéré
- ⦿ Utilisation :
 - > `data.<type>.<nom>.<attribut>`
- ⦿ Exemple : Création d'un subnet passant en argument l'attribut "ID" de son VPC (récupéré d'une datasource)

Type

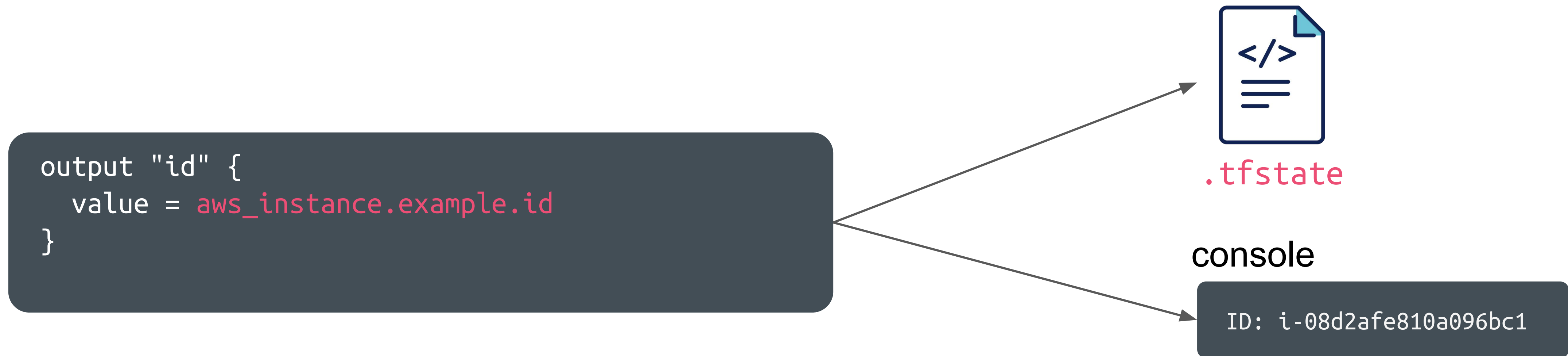
Nom



```
data "aws_vpc" "my_vpc" {  
  cidr_block = "192.168.1.0/24"  
}
```

```
resource "aws_subnet" "subnet1" {  
  vpc_id      = data.aws_vpc.my_vpc.id  
  cidr_block  = "192.168.1.0/25"  
}
```


Permet d'afficher en console et de sauver dans le tfstate la valeur d'un attribut



`Sensitive = true` permet d'obfusquer la valeur dans la console mais pas dans le tfstate

- ⦿ Permet d'extraire **la valeur d'une variable output** depuis le tfstate

```
$ terraform output  
id = i-0d1df83f5dfd91367
```

- ⦿ Exemple:
 - > Affiche la valeur de l'id de l'instance.

Nécessite que la layer soit déjà existante !

Resource Attributes = `<type>.<nom>.<attribut>`

Variable Call = `var.<nom>`

Local Call = `local.<nom>`

Datasource Call = `data.<type>.<nom>.<attribut>`

```
git clone  
https://5AIW:5LeHYHdZ_Eyffdkzs61o@gitlab.com/santunes-formations/terraformed.git
```

- ⦿ Permettent de changer le comportement de certaines ressources
 - > Forcer la dépendance entre 2 ressources
 - > Créer plusieurs fois la même ressource
 - > Modifier son cycle de vie(create, update, destroy)
- ⦿ S'utilise sur n'importe quel type de resource comme un attribut classique

```
resource "aws_instance" "foo" {  
  ami          = "ami-760aaa0f"  
  instance_type = "t2.medium"  
}
```

```
resource "aws_instance" "foo" {  
  ami          = "ami-760aaa0f"  
  instance_type = "t2.medium"  
  <meta_parameter> = "value"  
}
```


- ⦿ Terraform gère de manière implicite et automatique les dépendances entre ressources
 - > Ex : VPC > Subnet > SecurityGroups > Instances
- ⦿ **depends_on** permet de forcer de manière explicite une dépendance

```
resource "aws_instance" "db" {  
  ami          = "ami-760aaa0f"  
  instance_type = "t2.medium"  
}  
  
resource "aws_instance" "app" {  
  ami          = "ami-760aaa0f"  
  instance_type = "t2.medium"  
  depends_on   = ["aws_instance.db"]  
}
```

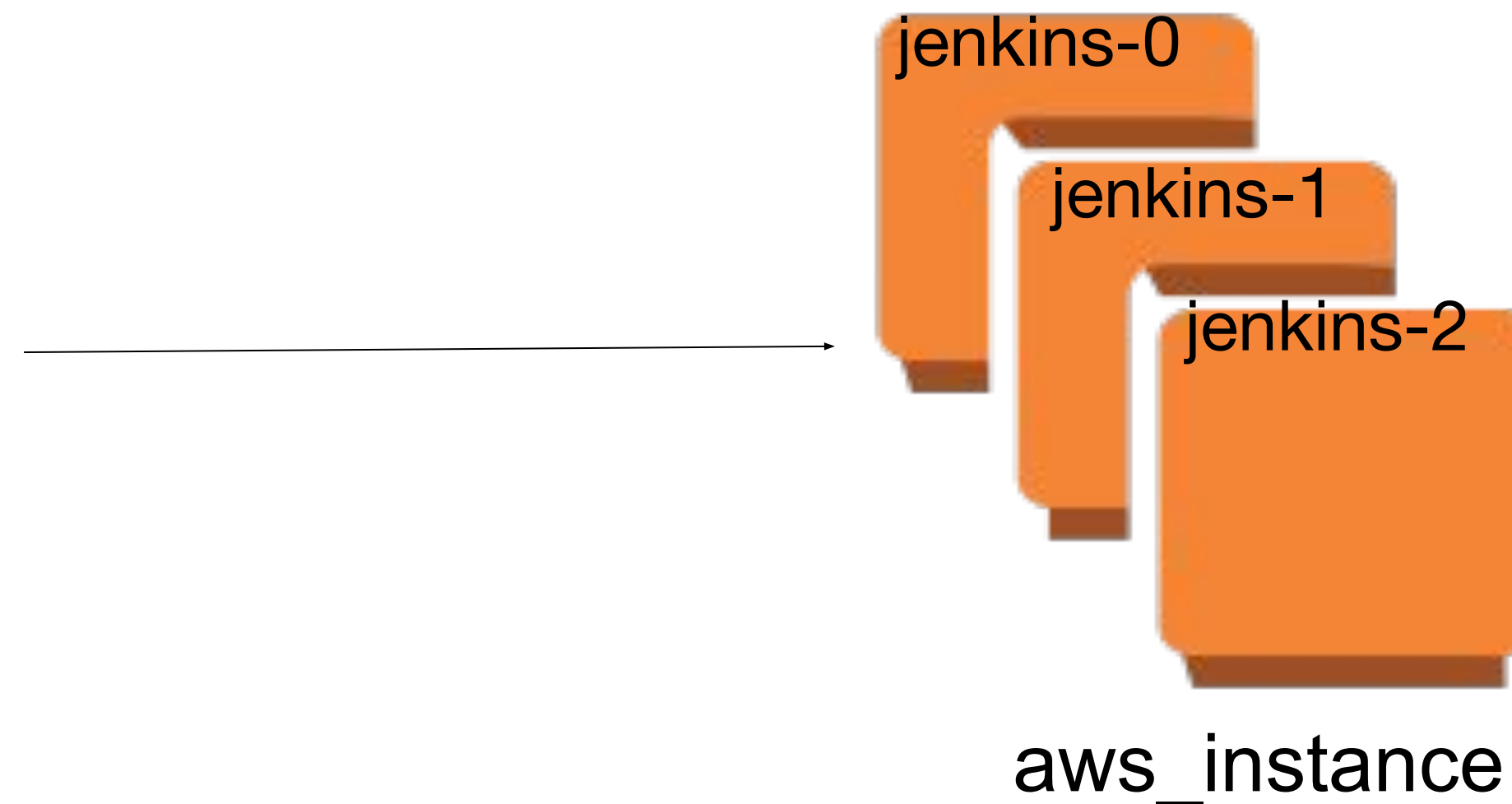
A utiliser de manière
exceptionnelle, en
connaissance de cause.

- ⦿ **create_before_destroy**
 - > Crée une nouvelle resource, puis détruit l'ancienne
 - > Ex : Ajouter une nouvelle entrée DNS, puis détruire l'ancienne entrée
- ⦿ **prevent_destroy**
 - > Permet d'éviter de détruire une resource
 - > Ex : Utile sur un environnement de production
- ⦿ **ignore_changes**
 - > Permet d'éviter de détruire/recréer une resource dans le cas d'un changement de valeur
 - > Ex : Éviter de détruire/recréer une instance dans le cas d'un changement d'AMI

```
resource "aws_instance" "foo" {  
  ami          = "ami-760aaa0f"  
  instance_type = "t2.medium"  
  lifecycle {  
    ignore_changes = ["ami"]  
  }  
}
```

- Count permet de boucler sur la création d'une ressource
 - > `count = N` occurrences
- Utilisation de l'indice d'itération :
 - > `count.index`

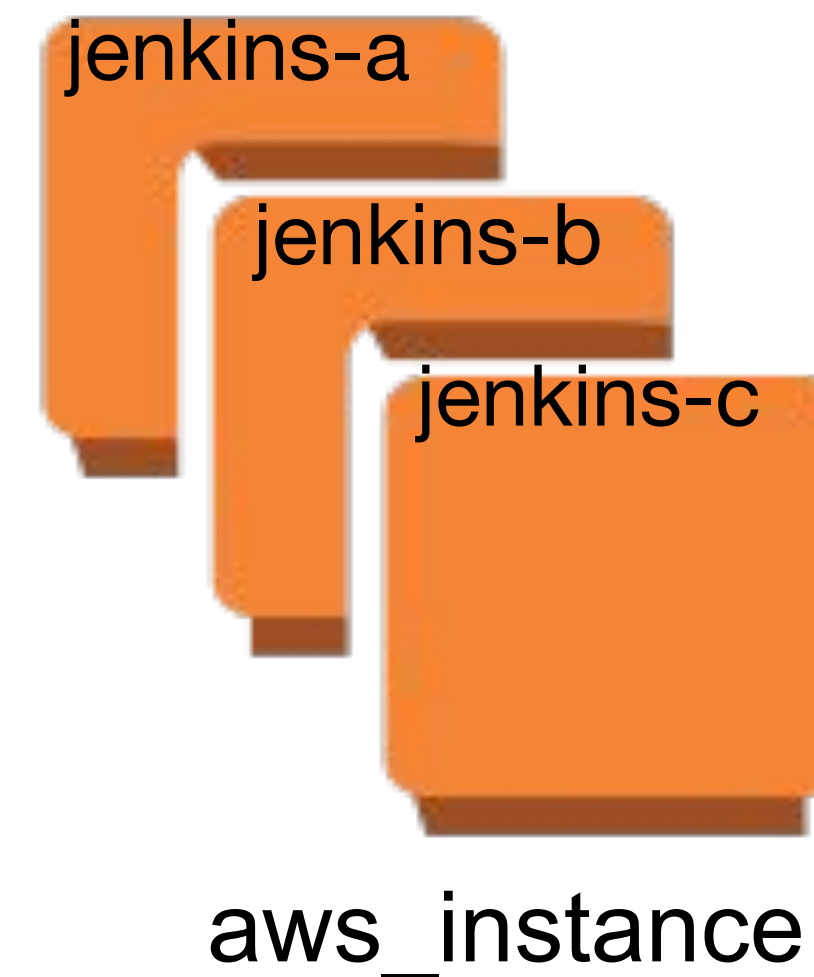
```
resource "aws_instance" "jenkins" {  
  ami          = "ami-760aaa0f"  
  instance_type = "t2.medium"  
  count        = 3  
  
  tags {  
    Name = "jenkins-${count.index}"  
  }  
}
```



Construit un tableau des ressources en output:
ex: `aws_instance.jenkins[0].id`

- Comme le count mais pour des ressources avec quelques attributs différents
- Prends en arguments une map ou un set(string)

```
resource "aws_instance" "jenkins" {  
  for_each      = toset(["a", "b", "c"])  
  ami           = "ami-760aaa0f"  
  instance_type = "t2.medium"  
  
  tags {  
    Name = "jenkins-${each.key}"  
  }  
}
```



Construit un tableau des ressources en output:
ex: aws_instance.jenkins[0].id

- ⦿ Le HCL ne se limite pas à juste de l'interpolation de variable, il possède un véritable moteur d'expression le rendant plus flexible
- ⦿ Permet de changer le comportement de certaines ressources en jouant sur les valeurs de leurs attributs:
 - > Faire référence à des valeurs injectées dynamiquement ou généré par le cloud provider
 - > Calculer des paramètres en fonction de différentes valeurs
 - > Injecter un block de code dynamiquement

```
resource "aws_security_group" "sg" {  
  name = expression  
}
```

```
resource "aws_security_group" "sg" {  
  name = "${expression}-toto-${expression}"  
}
```

EXPRESSIONS - OPERATEURS

Opérateurs Arithmétiques (numbers)

> +, -, *, /, %, retourne number

```
resource "aws_instance" "web_server" {  
  count      = var.nb_app1 + var.nb_app2  
  ami        = "ami-760aaa0f"  
  instance_type = "t2.medium"  
}
```

Opérateurs d'égalité (any)

> ==, !=, retourne bool

```
resource "aws_instance" "web_server" {  
  ami        = "ami-760aaa0f"  
  instance_type = "t2.medium"  
  
  disable_instance_termination = var.env == "prod"  
}
```

Opérateurs de comparaison (numbers)

> <, <=, >=, >, retourne bool

```
resource "aws_lambda_function" "auto_clean" {  
  function_name = "auto_clean"  
  publish       = var.nb_instance > 2  
}
```

Opérateurs logique (bool)

> &&, ||, !, retour bool

```
resource "aws_eip" "public_ip" {  
  vpc = var.is_prd && var.with_public_ip  
}
```


- ⦿ Utilisation de ternaire pour setter une valeur en fonction d'une condition
 - > `condition ? true_val : false_val`
- ⦿ Count + ternaire permettent de réaliser un test *if then else*
 - > `count = var.foo ? 1 : 0`

```
variable "jenkins_activated" {  
    default = true  
}  
  
resource "aws_instance" "jenkins" {  
    count      = var.jenkins_activated ? 1 : 0  
    ami        = "ami-760aaa0f"  
    instance_type = "t2.medium"  
}
```

- ⦿ Permettent par exemple :
 - > De manipuler des variables terraform (sous-chaînes de caractères, fusion de tableaux, récupérer les clés d'une map ...)
 - > De manipuler IP et CIDR blocks
 - > De manipuler des chemins et fichiers
 - > D'effectuer des calculs mathématiques (log, puissance ...)
 - > De calculer des hashes (sha512, md5 ...)
 - > Conversion de type
 - > ...

- ⦿ Documentation

▼ Functions

- Numeric Functions
- String Functions
- Collection Functions
- Encoding Functions
- Filesystem Functions
- Date and Time Functions
- Hash and Crypto Functions
- IP Network Functions
- Type Conversion Functions

EXPRESSION - FOR LOOP

- ⦿ Permet de transformer une valeur composé (List, Map) en une autre valeur composée (List, Map) en itérant sur chacun de ses éléments
 - > Ex: ["a", "b", "c", "d"] ==> ["A", "B", "C", "D"]
 - > Ex: ["a", "b", "c", "d"] ==> { "a" => "b", "c" => "d" }
 - > Ex: { "name" => "jenkins", "team" => "OPS" } ==> ["name", "jenkins", "team", "ops"]
- ⦿ Le type d'arrivé est spécifié autour du for
 - > [for expression] => List
 - > { for expression } => Map

```
variable "public_ips" {  
  type      = list(string)  
  default = ["1.1.1.1", "8.8.8.8"]  
}
```

```
resource "aws_security_group_rule" "open_dns" {  
  type      = "egress"  
  from_port = 53  
  to_port   = 53  
  protocol  = "udp"  
  cidr_blocks = [for i in var.public_ips : concat(i, "/32")]  
  
  security_group_id = "aws_security_group.default.id"  
}
```

Syntaxe multi-line disponible aussi

- ⦿ Méthode plus simple que le for permettant de construire une simple liste à partir d'une liste d'objet:
 - > `aws_instance.jenkins[*].id` => ["id-azerty", "id-qsdvgh", "id-wxcvbn"]
 - > `[*]` permet d'itérer sur tous les éléments du tableau

```
resource "aws_instance" "jenkins" {  
  ami          = "ami-760aaa0f"  
  instance_type = "t2.medium"  
  count        = 3  
  
  tags {  
    Name = "jenkins-${count.index}"  
  }  
}
```

```
output "instance_ip_addrs" {  
  value = aws_instance.jenkins[*].address  
}
```

Fonctionne également quand `aws_instance.jenkins` est composé d'un seul élément

EXPRESSION - DYNAMIC BLOCK

- ⦿ Permet de générer dynamiquement des blocks de configuration
 - > Agit comme une boucle for mais au niveau du HCL directement

```
resource "aws_security_group" "example" {  
  name = "example"  
  
  dynamic "ingress" {  
    for_each = var.service_ports  
    content {  
      from_port = ingress.value  
      to_port   = ingress.value  
      protocol  = "tcp"  
    }  
  }  
}
```

A ne pas trop utiliser pour garder un code simple et lisible.

Privilégier les ressources avec attachement !

- ⦿ `template_file`
 - > Permet de générer un fichier
- ⦿ `template_cloudinit_config`
 - > Permet de configurer cloudinit

Permet d'utiliser toute la puissance du moteur de templating pour générer un fichier

```
#!/bin/bash
```

```
echo "IP = ${my_ip}" > /etc/app/app.conf
```

```
data "template_file" "init" {
  template = file("init.tpl")

  vars {
    my_ip = aws_instance.web.private_ip
  }
}

resource "aws_instance" "web" {
  ami           = "ami-760aaa0f"
  instance_type = "t2.medium"

  user_data = data.template_file.init.rendered
}
```

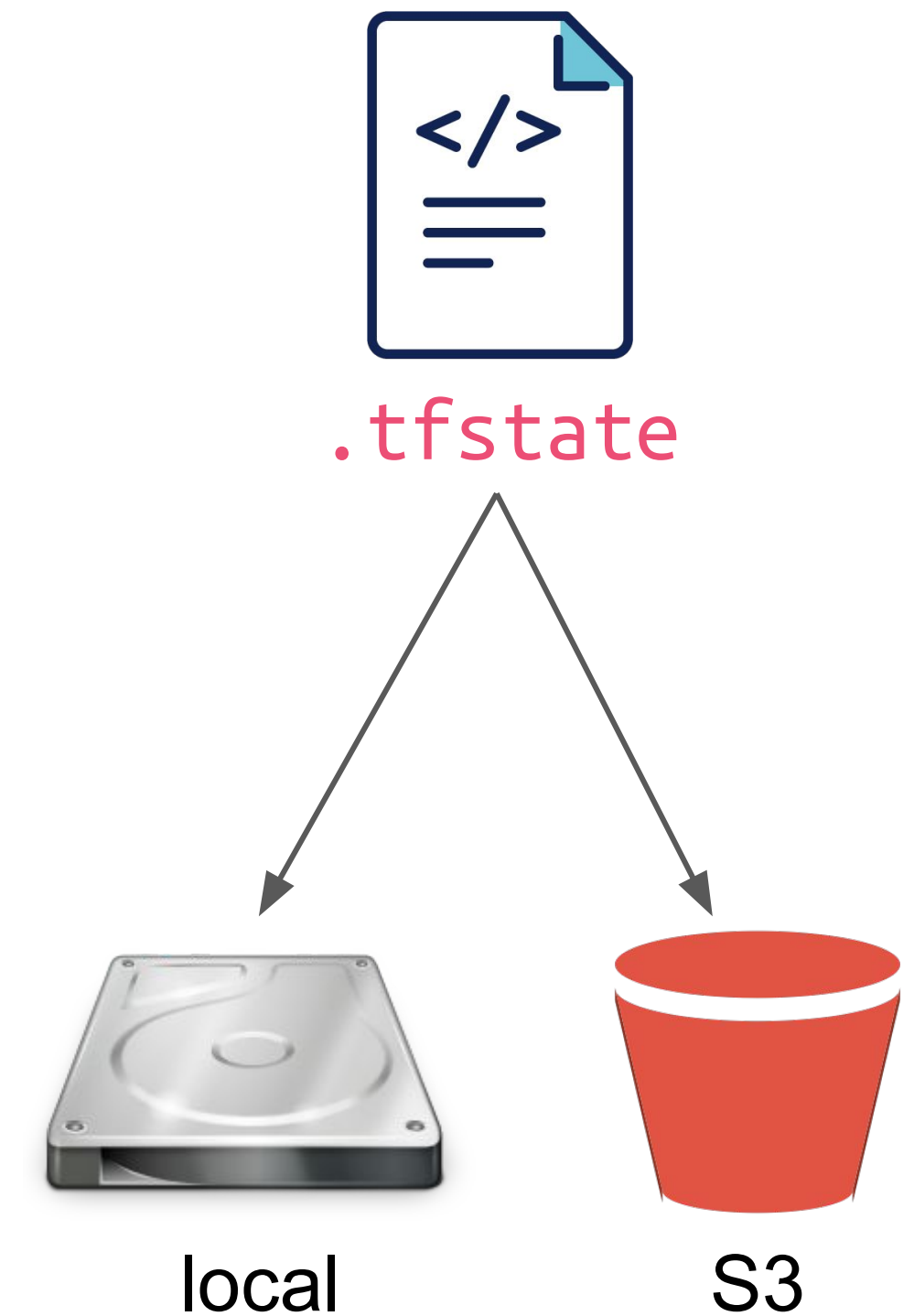


```
git clone  
https://5AIW:5LeHYHdZ_Eyffdkzs61o@gitlab.com/santunes-formations/terraformed.git
```

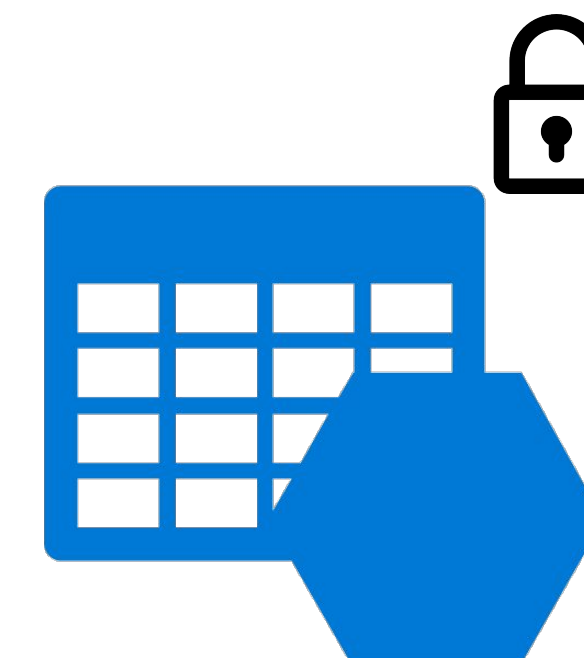
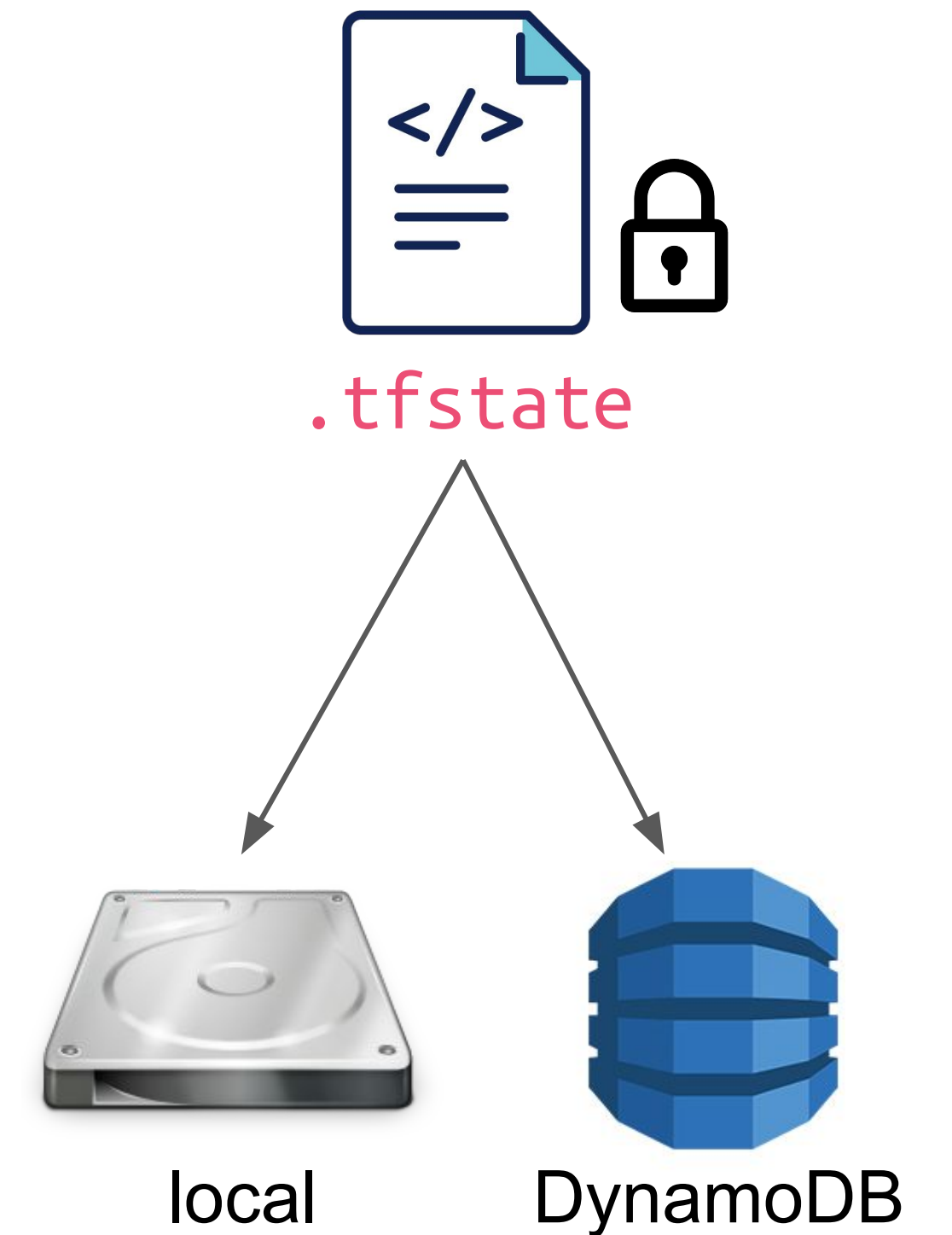
“Passage à l’
échelle”



- ⦿ En travail en équipe, le Tfstate doit être partagé
 - > Ex : sur S3
- ⦿ D'autres backends disponible aussi
 - > Azure
 - > GCS
 - > Terraform enterprise

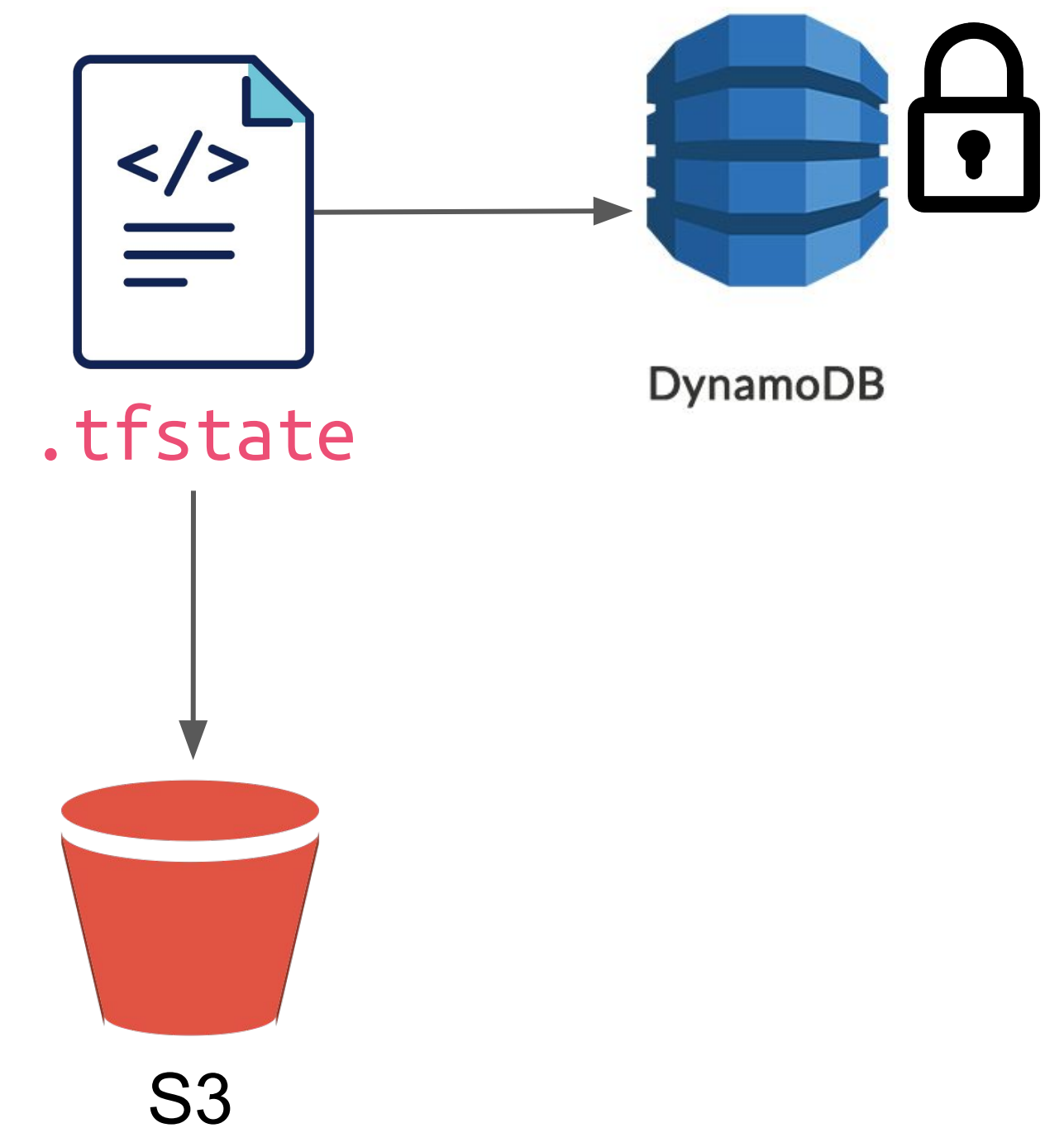


- Sur un backend partagé, pour éviter la corruption du Tfstate par un accès concurrent, il est recommandé d'utiliser une table de lock
 - Ex : lock_table sur Amazon DynamoDB
- Tous les backends ne gèrent pas le locking



- Exemple de backend sur S3 avec verrou sur une table DynamoDB sur l'environnement de **production**

```
terraform {  
  backend {  
    lock_table = "terraform-state-locking"  
    bucket     = "production"  
    key        = "my-project/terraform.tfstate"  
  }  
}
```



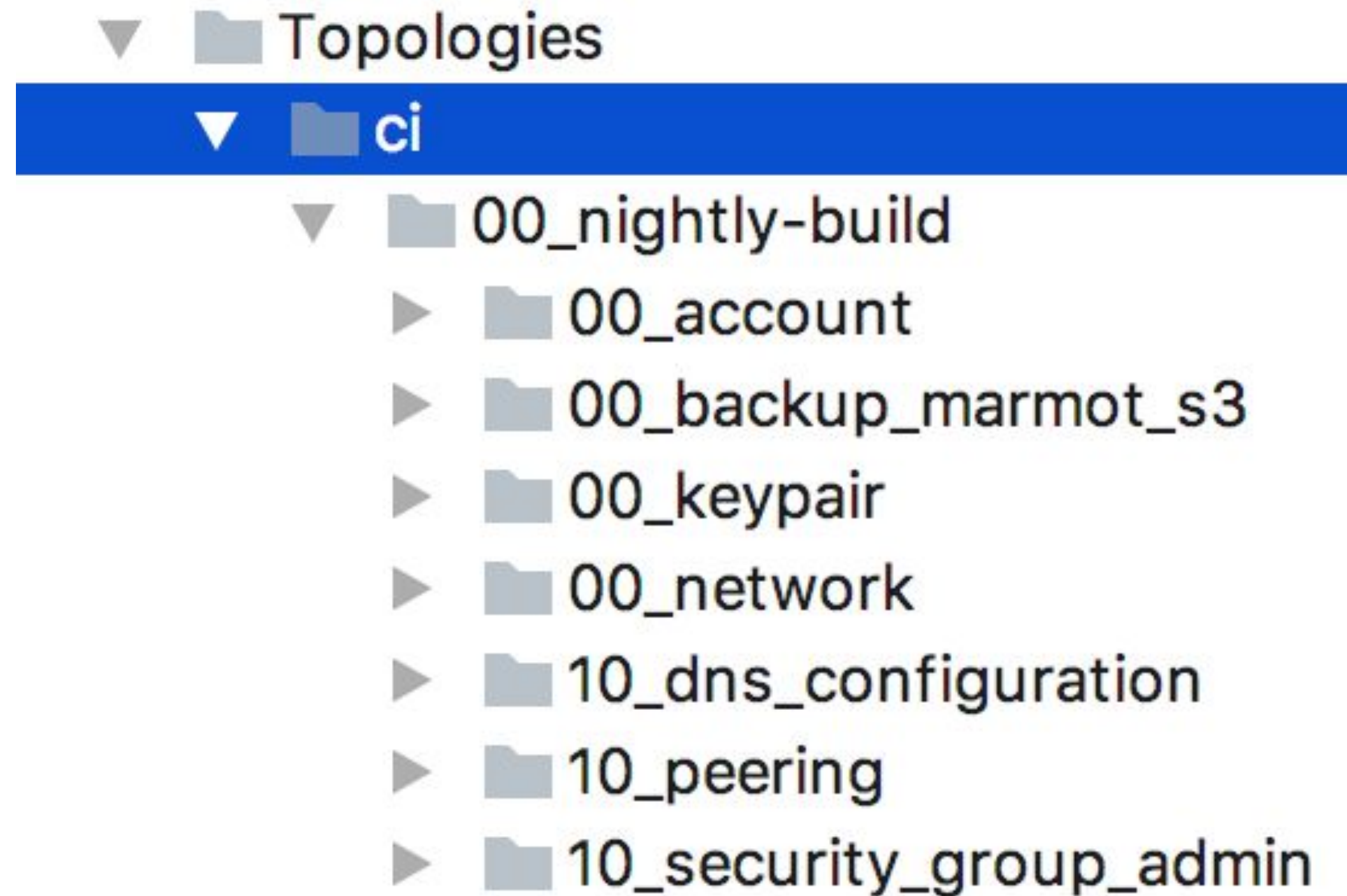
- ⦿ La version de terraform utilisé est écrite dans le Tfstate
 - > Pas de rollback sur les versions de terraform
- ⦿ Les providers terraform évoluent en permanence (nouveaux services, bugs, ...)
 - > L'intégralité des providers est disponible sur: <https://github.com/terraform-providers>
 - > Chaque provider à ses propres releases: <providers>/terraform-provider-aws/releases

```
terraform {  
  required_version = ">= 0.12.6"  
  required_providers {  
    aws      = ">= 2.26.0"  
    template = "= 2.1.2"  
  }  
  backend {...}  
}
```

Par défaut Terraform télécharge les dernières versions des providers

- ⦿ Une layer est un répertoire comportant des fichiers Terraform
- ⦿ Avantages
 - > Ne pas être bloqué (lock) si quelqu'un travaille déjà sur cette layer
- ⦿ Inconvénients
 - > Les attributs de ressources ne sont plus accessibles directement
 - Utilisation de datasources pour accéder aux ressources des autres layers
 - > Engendre de la duplication de code (datasources, locals, variables ...)

A utiliser en fonction
de la **composition de l'équipe**
pour éviter
les accès concurrents.



Lors du lancement d'une commande, Terraform charge tous les fichiers présents dans le répertoire spécifié par ordre alphabétique

Quelques bonnes pratiques pour le découpage:

- Regrouper les variables dans un **variables.tf**
- Regrouper les data sources dans un **datasources.tf**
- Regrouper les valeurs des variables dans **terraform.tfvars**
Faire varier le nom en fonction des environnements
- **backend.tf** pour la configuration du backend
- **provider.tf** pour la configuration du provider
- Pour les topologies, le découpage se fait d'un point de vue fonctionnel



- ⦿ Les workspaces permettent de lancer un layer terraform dans plusieurs contextes différents.
 - > Chaque workspace possède un tfstate qui lui est propre
 - > Le workspace par défaut se nomme "default"

Le workspace permet de lancer le même code sur plusieurs environnements.

🕒 Création d'un workspace

```
$ terraform workspace new dev
```

Created and switched to workspace "dev"!

You're now on a new, empty workspace. Workspaces isolate their state, so if you run "terraform plan" Terraform will not see any existing state for this configuration.

🕒 Changement de workspace

```
$ terraform workspace list
default
* dev
```

```
$ terraform workspace select default
```

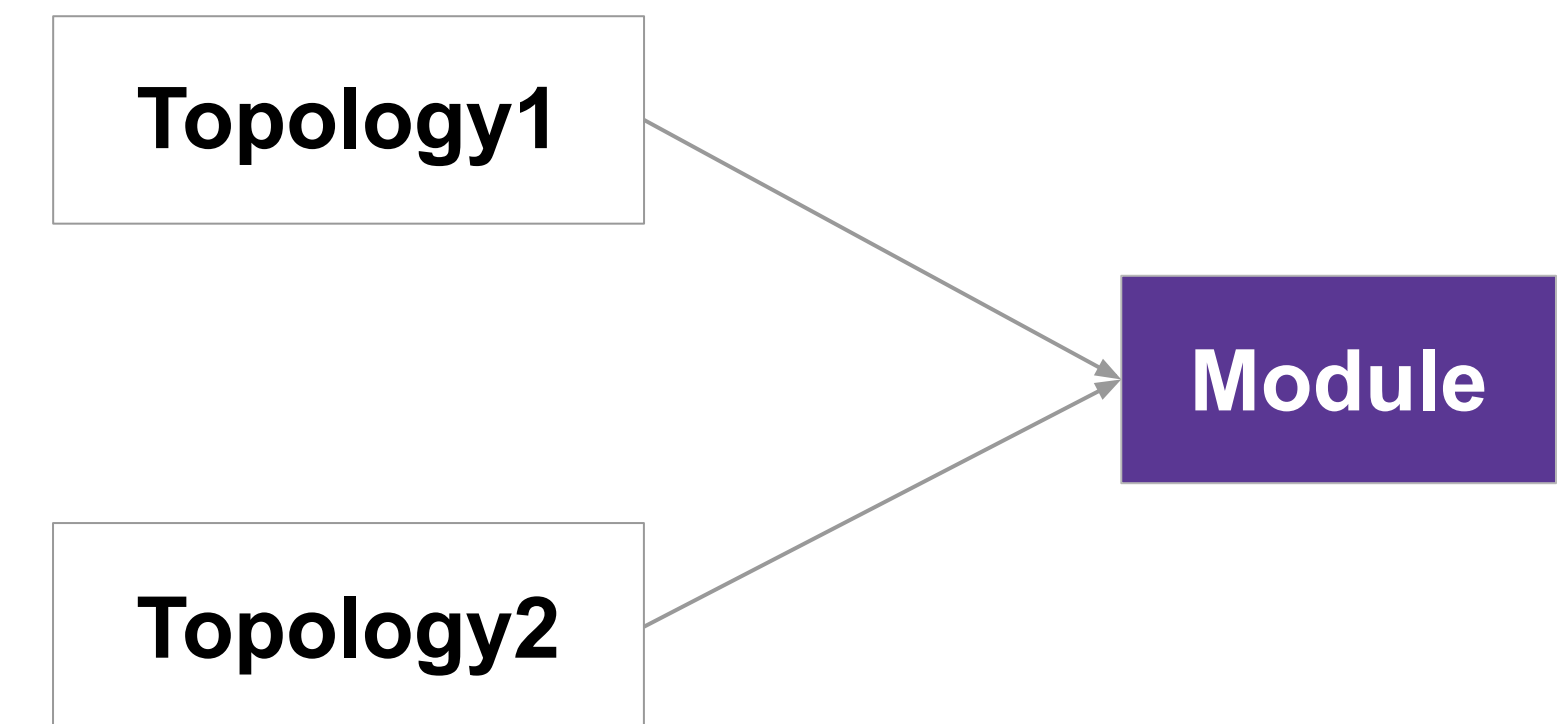
Switched to workspace "default".

- ⦿ Essayer d'avoir des **fichiers** “tf” les plus **maintenables** possible, c'est plus simple à lire.
- ⦿ **Séparer les responsabilités** des topologies, un changement sur une **petite Topologie** fait moins peur que sur une grosse.
- ⦿ Abstraire le code dans les **modules**, toutes les layers font ainsi appel au même code.
- ⦿ Pas de *hard-coding* dans les layers, uniquement des références aux variables tfvars, ainsi le paramétrage se trouve à un seul endroit (dans les tfvars).

- L'utilisation de branches dans le dépôt de code Terraform doit faire l'objet d'attentions :
 - Si le code est appliqué sur une branche, comment traquer que la source de vérité est maintenant cette branche ?
 - Une pipeline lancée sur master peut écraser les changements effectués par du code joué sur une branche.

Il est plus simple de jouer le code Terraform constamment sur la même branche.

- ⦿ Quand utiliser des modules ?
 - > Factoriser des comportements communs
 - > Abstraire la complexité
 - > Partager le code



- ⦿ Un module est simplement un **répertoire** dans lequel se trouvent des fichiers “.tf”
- ⦿ Un module est composé de :
 - > Providers (ils sont soit redéfinis soit hérités du parent)
 - > Variables (argument d'entrée)
 - > DataSources
 - > Locales
 - > Resources
 - > Output (argument exposé en sortie du module)
- ⦿ Un module peut appeler d'autres modules

Module Required Inputs

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

Module Optional Inputs

```
variable "subnet_cidr_block" {  
  type      = "string"  
  default   = "192.168.1.0/25"  
}
```

Module Resources

```
resource "aws_vpc" "my_vpc" {  
  cidr_block = var.vpc_cidr_block  
  
  enable_dns_hostnames = true  
  enable_dns_support   = true  
}  
  
resource "aws_subnet" "my_subnet" {  
  cidr_block = var.subnet_cidr_block  
  
  vpc_id = aws_vpc.my_vpc.id  
}
```

Module Attributes

```
output "vpc_id" {  
  value = aws_vpc.my_vpc.id  
}  
  
output "subnet_id" {  
  value = aws_subnet.my_subnet.id  
}  
  
output "security_group" {  
  value = aws_security_group.my_sg  
}
```

Nom du module

Source du module

(Path local, Github, Bitbucket, S3, Http, ...)

(exemple Path local : ../modules/moduleA)

Arguments

```
module "nom" {  
  source      = "source_module"  
  
  arg1        = "value1"  
  arg2        = "value2"  
  ...  
}  
  
output "resource_id_from_module" {  
  value = module.nom.resource_id  
}
```

Ex : instance_id, vpc_id ...

- ⦿ Suivre les bonnes pratiques de développement avec le code d'infrastructure
 - > Versionner le code Terraform comme le code applicatif
 - > Faire des releases des modules
 - > Jouer et tester le code Terraform au travers d'un pipeline

- ⦿ Créer des modules pour partager le code entre différentes layers Terraform

- ⦿ Hashicorp propose une registry publique pour héberger les modules.
(<https://registry.terraform.io/>)
 - > La registry contient des modules pour la plupart des cloud provider (AWS, Azure, GCP, Alibaba Cloud, Oracle Cloud, vmware)
 - > Les "Verified Modules" sont des modules validés par Hashicorp
 - > N'importe qui peut partager un module sur la registry
- ⦿ Les versions Cloud et Entreprise de Terraform offrent une registry privée aux utilisateurs authentifiés

git clone

https://5AIW:5LeHYHdZ_Eyffdkzs61o@gitlab.com/santunes-formations/terraformed.git

- ⦿ Lors du fail d'un apply lié à une coupure réseau ou un arrêt brutal de terraform (plusieurs ctrl + c), un lock peut trainer sur la topology

```
Error locking state: Error acquiring the state lock: ConditionalCheckFailedException: The conditional request failed
```

```
status code: 400, request id: IGUA4N3QNLOPSE5GH06SH88EPVVV4KQNS05AEMVJF66Q9ASUAAJG
```

```
Lock Info:
```

```
ID:          40e844a6-ebb0-fcfd-da85-5cce5bd1ec89
```

- ⦿ Plutôt que d'éditer le service gérant le lock (Dynamodb, GCS, AzureBlob, ...)

Pour connaître le `Lock_ID`,
lancez un terraform apply

```
$ terraform force-unlock 40e844a6-ebb0-fcfd-da85-5cce5bd1ec89
```

Je n'éditerai jamais le tfstate

Je n'éditerai jamais le tfstate

Je n'éditerai jamais le tfstate

Je n'éditerai jamais le tfstate

Je n'éditerai jamais le tfstate

...

- ⦿ Commande CLI permettant de manager le fichier tfstate
- ⦿ **Usage** : `terraform state `cmd` [options] PATH`
 - > List : Liste les resources du tfstate
 - > Mv : Déplace resource/modules vers un autre tfstate
 - > Pull : Télécharge et affiche le contenu du tfstate
 - > Push : Upload le contenu d'un tfstate local sur un remote
 - > Rm : Supprime des items d'un tfstate
 - > Show : Affiche le contenu d'un item d'un tfstate

TERRAFORM IMPORT

- ⦿ Permet d'importer dans le tfstate des ressources non managées par terraform
- ⦿ Le code de topology n'est pas généré (mais des outils existent)
- ⦿ **Usage:** `terraform import [options] ADDRESS ID`

Exemple avec une instance :

```
$ terraform import aws_instance.foo i-abcd1234
```

Exemple avec un module :

```
$ terraform import module.foo.aws_instance.bar i-abcd1234
```


- ⦿ Permet de voir en détails les appels aux APIs du provider
- ⦿ Très pratique lors d'erreurs d'autorisation
- ⦿ Fonctionne au travers de variables d'environnement

```
export TF_LOG=DEBUG
```

```
$ terraform plan
...
DEBUG: Request ec2/DescribeVpcs Details:
---[ REQUEST POST-SIGN ]-----
POST / HTTP/1.1
Host: ec2.us-east-1.amazonaws.com
User-Agent: aws-sdk-go/1.14.26 ...
Content-Length: 95
Authorization: AWS4-HMAC-SHA256 ...
X-Amz-Date: 20180720T142618Z
Accept-Encoding: gzip

Action=DescribeVpcs&Filter.1.Name=tag%3AFormation&Filter.1.Value.1=
terraform&Version=2016-11-15
-----
DEBUG: Response ec2/DescribeVpcs Details:
---[ RESPONSE ]-----
HTTP/1.1 200 OK
Connection: close
Transfer-Encoding: chunked
Content-Type: text/xml; charset=UTF-8
Date: Fri, 20 Jul 2018 14:26:18 GMT
Server: AmazonEC2
Vary: Accept-Encoding
```