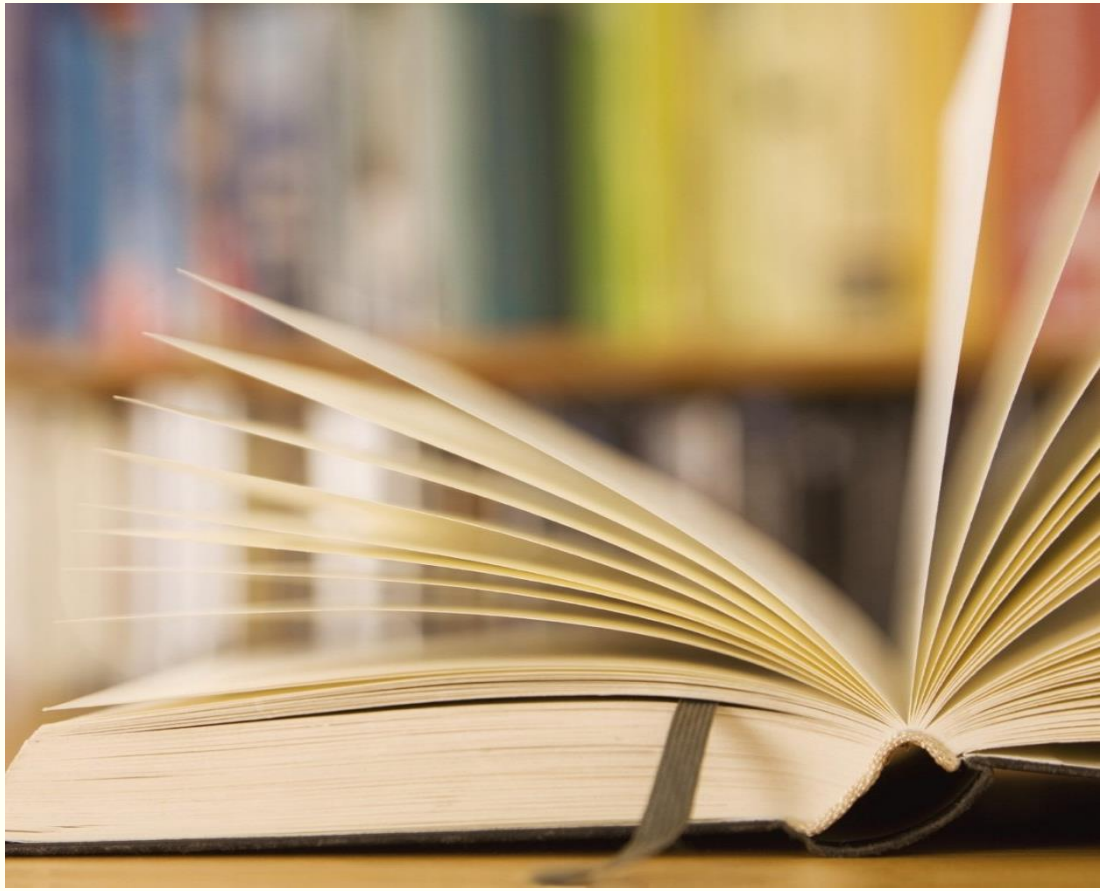
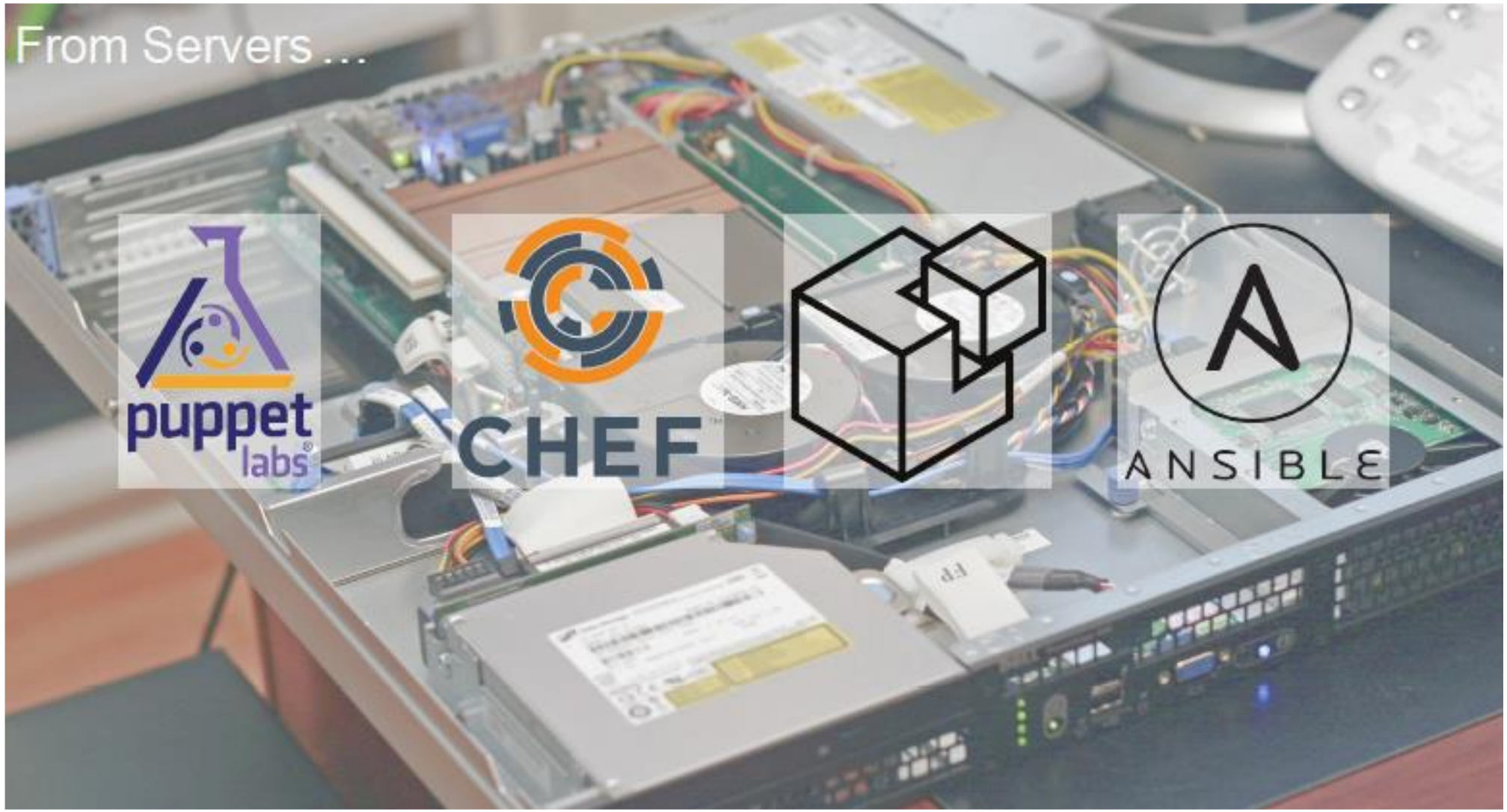
Cloud Configuration Management

TERRAFORM

Mr. Mohamed Salah MEDDEB



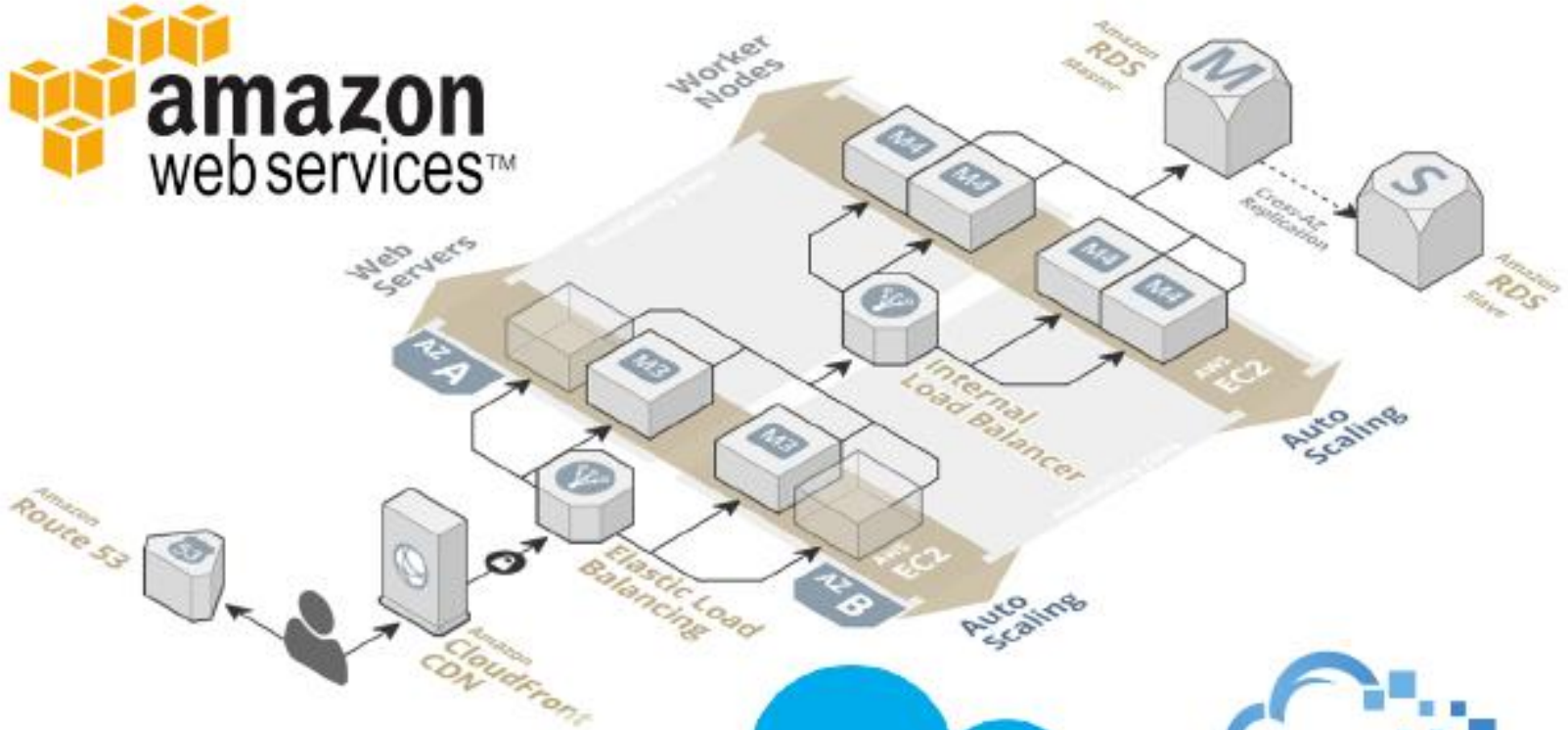
Plan


From Servers …
puppet labs
CHEF
ANSIBLE

…to Services

openstack
CLOUD SOFTWARE

amazon web services™



heroku

Google Cloud Platform

Microsoft Azure

DigitalOcean

Infrastructure as code (IaC) - Besoin

- L'infrastructure en tant que code (IaC) est l'automatisation de l'infrastructure utilisant des principes et des pratiques de développement logiciel.
- L'idée est que vous traitez votre infrastructure comme un logiciel, puis que vous écrivez, testez et exécutez du code pour définir, déployer, mettre à jour et détruire votre infrastructure.
- Vous écrivez du code pour gérer vos serveurs, bases de données, réseaux, journaux, déploiement et configuration d'applications.
- Lorsque vous souhaitez apporter des modifications à votre infrastructure, vous apportez des modifications au code, vous le testez, puis vous l'appliquez à vos systèmes.

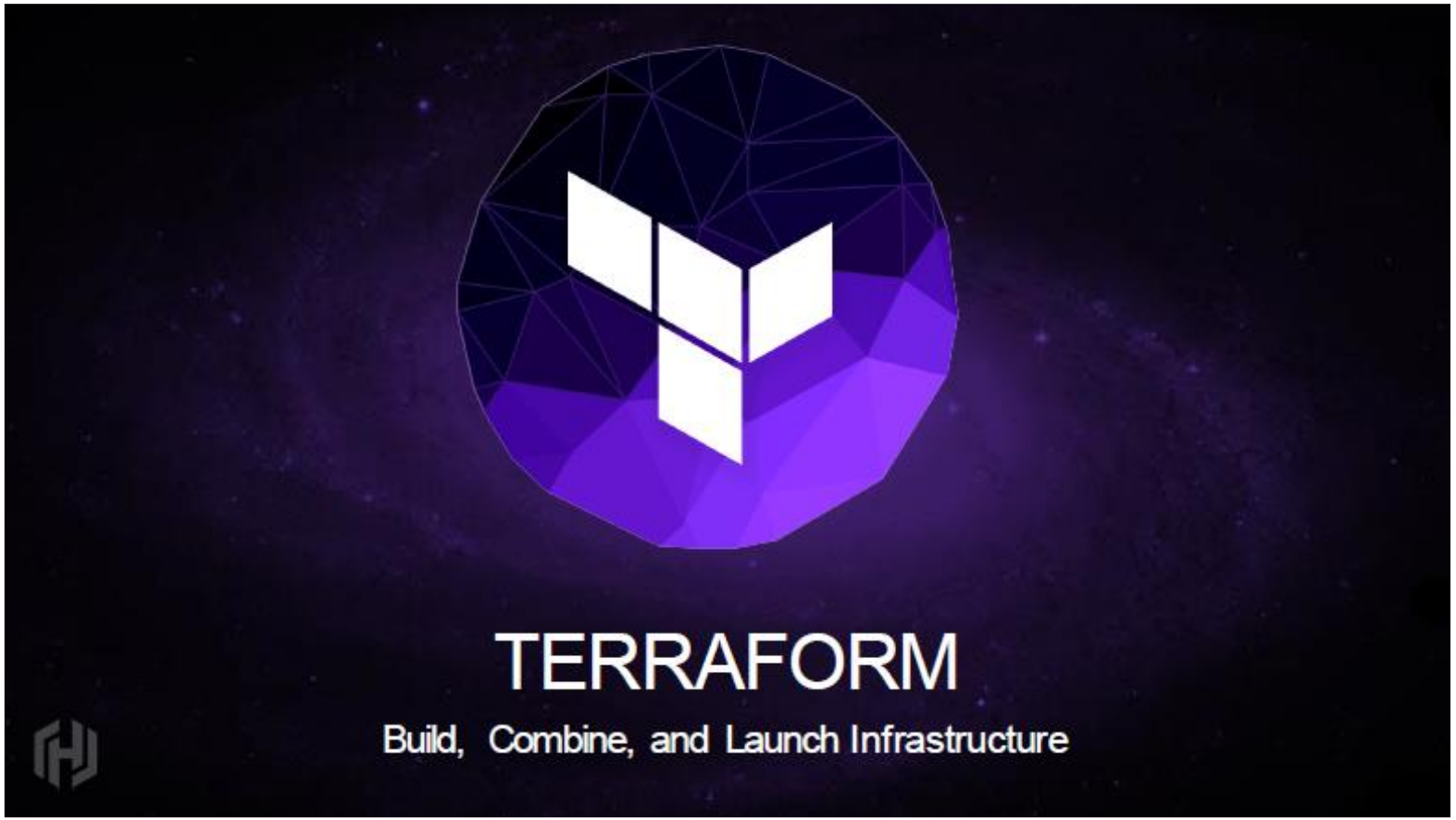
Défis de IaC

- Besoin d'apprendre à coder !!
- Je ne connais pas l'impact du changement !!
- Besoin d'annuler le changement, !!
- Impossible de suivre les modifications !!
- Impossible d'automatiser une ressource !!
- Environnements multiples pour l'infrastructure !!

➔ Terraform a été créé pour résoudre ces défis.

Défis de IaC

➔ Terraform a été créé pour résoudre ces défis.



Qu'est-ce que Terraform?

- Terraform est une infrastructure open-source en tant qu'outil de code développé par HashiCorp.
- Il est utilisé pour définir et provisionner l'infrastructure complète à l'aide d'un langage déclaratif facile à maîtriser.
- Il s'agit d'un outil de provisioning d'infrastructure dans lequel vous pouvez stocker la configuration de votre infrastructure cloud sous forme de codes.

Avantages de Terraform

- Fait de l'orchestration, pas seulement de la gestion de la configuration
- Prend en charge plusieurs fournisseurs tels que AWS, Azure, GCP, DigitalOcean et bien d'autres
- Fournir une infrastructure immuable où la configuration change en douceur
- Utilise un langage facile à comprendre, HCL (langage de configuration HashiCorp)
- Facilement portable vers n'importe quel autre fournisseur
- Prend en charge l'architecture client uniquement, donc pas besoin de gestion de configuration supplémentaire sur un serveur
- Langage facile à lire, Extensible et Gratuit

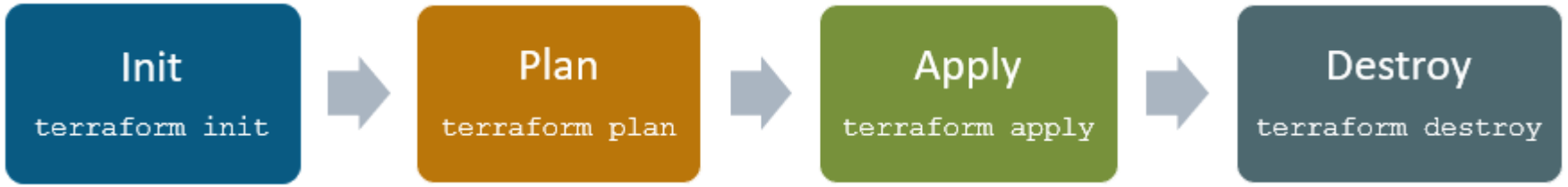
Concepts de base de Terraform (1/2)

- **Variables**: Également utilisé comme variable d'entrée, il s'agit d'une paire clé-valeur utilisée par les modules Terraform pour permettre la personnalisation.
- **Fournisseur**: C'est un plugin pour interagir avec les API de service et accéder à ses ressources associées.
- **Module**: C'est un dossier avec des modèles Terraform où toutes les configurations sont définies
- **Région**: Il se compose d'informations mises en cache sur l'infrastructure gérée par Terraform et les configurations associées.
- **Ressources**: Il s'agit d'un bloc d'un ou plusieurs objets d'infrastructure (instances de calcul, réseaux virtuels, etc.), qui sont utilisés dans la configuration et la gestion de l'infrastructure.

Concepts de base de Terraform (2/2)

- **La source de données**: Il est implémenté par les fournisseurs pour renvoyer des informations sur les objets externes à terraform.
- **Valeurs de sortie**: Ce sont les valeurs de retour d'un module terraform qui peuvent être utilisées par d'autres configurations.
- **Plan**: C'est l'une des étapes où il détermine ce qui doit être créé, mis à jour ou détruit pour passer de l'état réel / actuel de l'infrastructure à l'état souhaité.
- **Ajouter** : C'est l'une des étapes où il applique les changements d'état réel / actuel de l'infrastructure afin de passer à l'état souhaité.

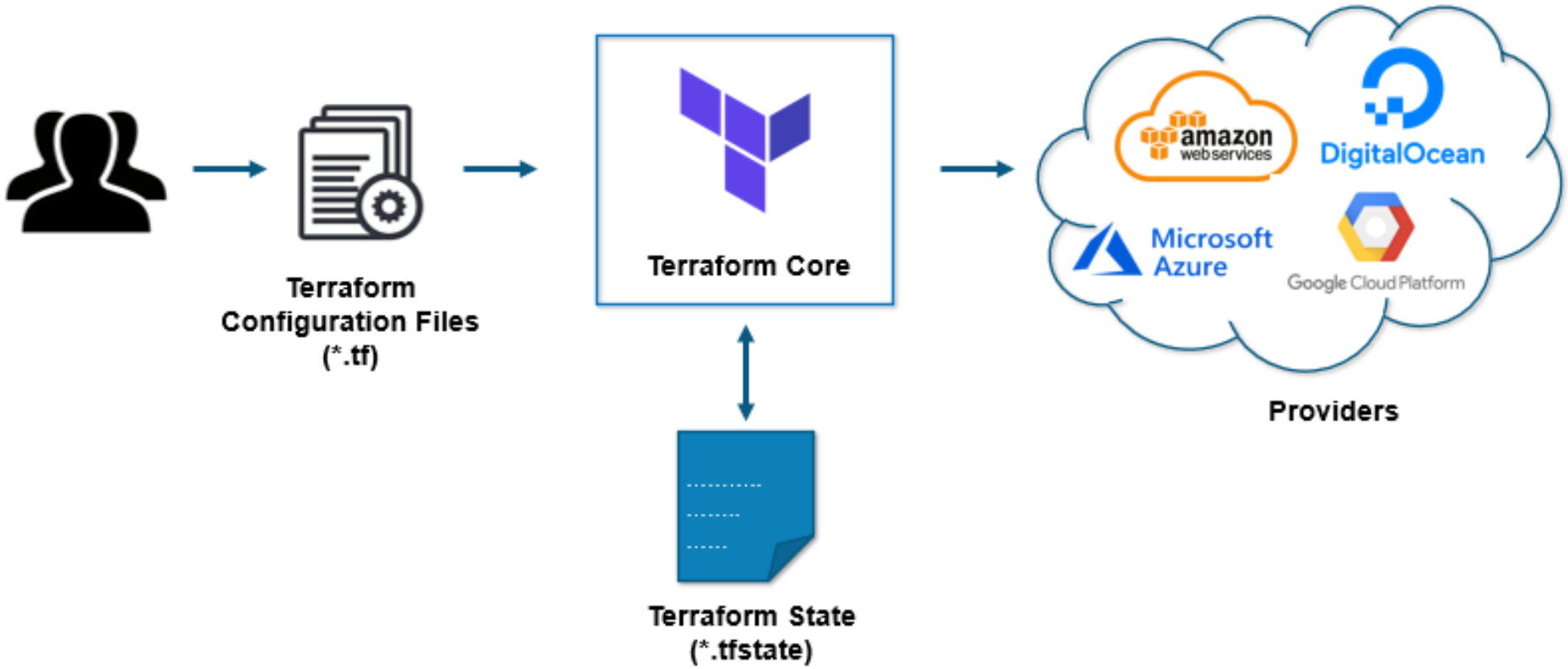
Cycle de vie Terraform



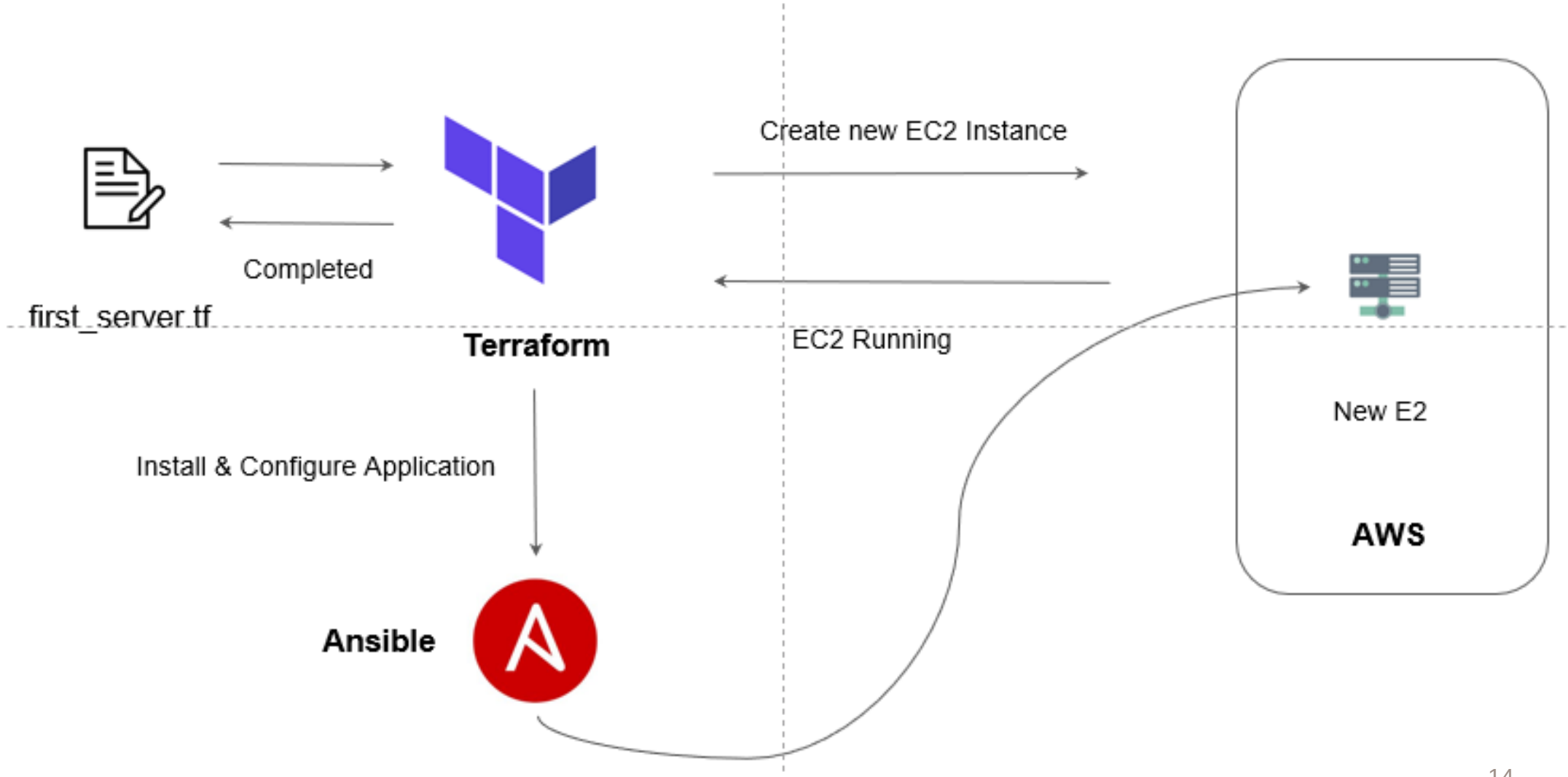
- **Terraform init** initialise le répertoire de travail qui comprend tous les fichiers de configuration
- **Le plan Terraform** est utilisé pour créer un plan d'exécution pour atteindre l'état souhaité de l'infrastructure. Les changements dans les fichiers de configuration sont effectués afin d'atteindre l'état souhaité.
- **Terraform apply** effectue ensuite les modifications de l'infrastructure telles que définies dans le plan, et l'infrastructure arrive à l'état souhaité.
- **Terraform destroy** est utilisé pour supprimer toutes les anciennes ressources d'infrastructure, qui sont marquées comme corrompues après la phase d'application

Fonctionnement de Terraform

- Terraform a deux composants principaux qui composent son architecture:
  - Terraform Core
  - fournisseurs



IaC et Configuration Management



Installation

Windows

macOS

Linux

FreeBSD

OpenBSD

Solaris

Lab 1 : Installation Terraform

Déployez vos premières ressources

RESOURCE & PROVIDERS

- Terraform s'appuie sur des plugins appelés « **providers**» pour interagir avec les systèmes distants (AWS, GCP, AZURE, Digital Ocean , OVH …), .
- Les configurations Terraform doivent déclarer les providers dont elles ont besoin, afin que Terraform puisse les installer et les utiliser. De plus, certains drivers nécessitent une configuration (comme des URL de point de terminaison ou des régions cloud) avant de pouvoir être utilisés.
- Une configuration de fournisseur est créée à l'aide d'un providerbloc :

```
provider "google" {
        project = "acme-app"
        region = "us-central1"
}
```

RESOURCE & PROVIDERS

- Les ressources sont l'élément le plus important du langage Terraform.
- Chaque bloc de ressources décrit un ou plusieurs objets d'infrastructure, tels que des **réseaux virtuels**, des **instances de calcul** ou des **composants de niveau supérieur** tels que des **enregistrements DNS**.
  - **Resource Blocks** documente la syntaxe de déclaration des ressources.
  - **Resource Behavior** explique plus en détail comment Terraform gère les déclarations de ressources lors de l'application d'une configuration.
  - Les documents de la section méta-arguments des arguments spéciaux qui peuvent être utilisés avec tous les types de ressources, y compris **depends_on**, **count**, **for_each**, **provider** et **lifecycle**.
  - **Les drivers de documents** configurant les actions de post-création pour une ressource à l'aide des blocs provisioneret connection. Étant donné que les fournisseurs sont non déclaratifs et potentiellement imprévisibles, nous vous recommandons fortement de les traiter en dernier recours.

Blocs de ressources

- Un bloc de ressource déclare une ressource d'un **type donné** ("**aws_instance**") avec un **nom local** donné ("**web**").
- Le nom est utilisé pour faire référence à cette ressource ailleurs dans le même module Terraform, mais n'a aucune signification en dehors de la portée de ce module.

```
resource "aws_instance" "web" {
        ami = "ami-a1b2c3d4"
        instance_type = "t2.micro"
}
```

- Dans le corps du bloc (entre **{**et **}**) se trouvent les arguments de configuration de la ressource elle-même.
- Chaque ressource est associée à un type de ressource unique , qui détermine le type d'objet d'infrastructure qu'elle gère et les arguments et autres attributs pris en charge par la ressource.

Lab 2 : première infrastructure AWS depuis Terraform

Déploiements dynamiques avec le variables

- Le langage Terraform comprend quelques types de blocs pour préparer des déploiements dynamiques avec le variables.
  - **Les variables d'entrée (Input Variables)** servent de paramètres pour un module Terraform, afin que les utilisateurs puissent personnaliser le comportement **sans modifier la source**.
  - **Les valeurs de sortie (Output Values)** sont comme les valeurs de retour pour un module Terraform.
  - **Les valeurs locales (Local Values)** sont une fonctionnalité pratique pour attribuer un nom court à une expression.
- Personnalisation de votre déploiement ➔ Attributs/variables
- Réparation d'informations dynamiques ➔ output

Les variables d'entrée (Input Variables)

- Les variables d'entrée sont souvent appelées simplement « variables » ou « variables Terraform »
- Permet de partager des modules sur différentes configurations Terraform, rendant votre module composable et réutilisable.
- Lorsque vous déclarez des variables dans le module racine de votre configuration, vous pouvez définir leurs valeurs à l'aide des options CLI et des variables d'environnement.
- Lorsque vous les déclarez dans les modules enfants , le module appelant doit passer des valeurs dans le module bloc.

Les variables d'entrée(Input Variables)

- Chaque variable d'entrée acceptée par un module doit être déclarée à l'aide d'un **variable bloc**
- Terraform CLI définit les arguments facultatifs suivants pour les déclarations de variables :
  - ***default*** - Une valeur par défaut qui rend alors la variable facultative.
  - ***type*** - Cet argument spécifie quels types de valeur sont acceptés pour la variable.
  - ***validation*** - Un bloc pour définir des règles de validation, généralement en plus des contraintes de type.
  - ***sensitive*** - Limite la sortie de l'interface utilisateur Terraform lorsque la variable est utilisée dans la configuration.

```
variable "image_id" {
  type = string
}
variable "image_id" {
  type   = string  description = "The id … "
  validation {condition = length(var.image_id)> 4
      && substr(var.image_id, 0, 4)
      == "ami-"
  error_message = "The image_id value must be a
      valid AMI id, starting with \"ami-\ "."
}}
variable "docker_ports" {
  type = list(object({
    internal = number
    external = number
    protocol = string
  }))
  default = [ {
      internal = 8300
      external = 8300
      protocol = "tcp"
    }
  ]
}
```

Les valeurs de sortie

- Rendent les informations sur l'infrastructure disponibles sur la ligne de commande et peuvent exposer des informations pour d'autres configurations Terraform à utiliser.
- Similaires aux valeurs de retour dans les langages de programmation.
- Les valeurs de sortie ont plusieurs utilisations :
  - Un module enfant peut utiliser des sorties pour exposer un sous-ensemble de ses attributs de ressource à un module parent.
  - Un module racine peut utiliser des sorties pour imprimer certaines valeurs dans la sortie CLI après avoir exécuté **terraform apply**.
  - Lors de l'utilisation de l'état distant , les sorties du module racine sont accessibles par d'autres configurations via une source de données **terraform_remote_state**

Les valeurs de sortie

- Chaque valeur de sortie exportée par un module doit être déclarée à l'aide d'un output bloc

```
output "instance_ip_addr" {
    value = aws_instance.server.private_ip
    description = "The private IP address of the main server instance."
}
```

- L' argument value prend une expression dont le résultat doit être retourné à l'utilisateur.

Les valeurs de sortie

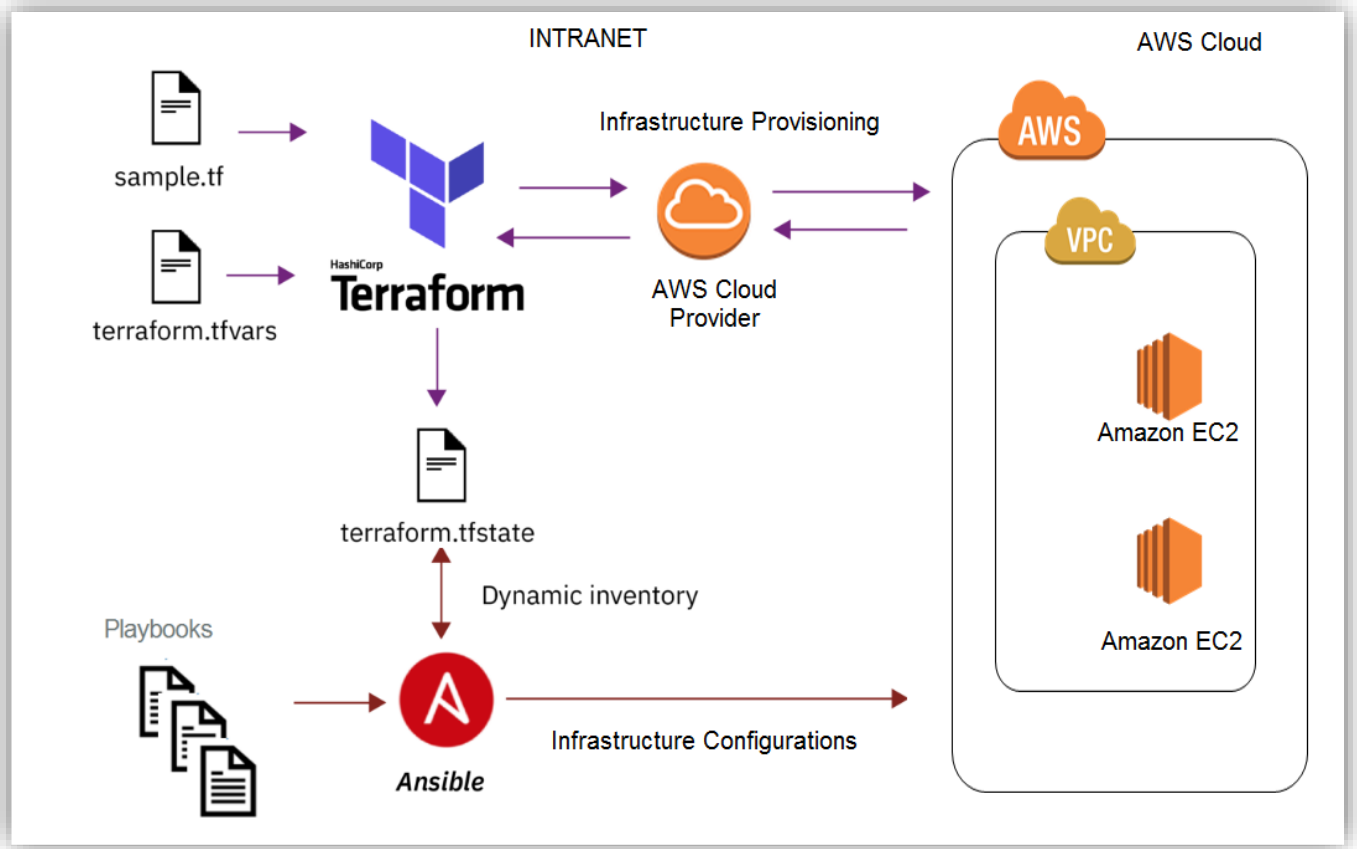
- Des valeurs nommées auxquelles vous pouvez vous référer dans votre configuration.
- Permet de simplifier la configuration Terraform et éviter les répétitions.

```
locals {
      service_name = "forum"
      owner        = "Community Team"
}

locals {
      instance_ids = concat(aws_instance.blue.*.id, aws_instance.green.*.id)
}

locals {
    common_tags = {
        Service = local.service_name
        Owner   = local.owner
    }
}
```

Lab 3 : Déploiements dynamiques avec les variables

Les provisionneurs

- Les provisionneurs peuvent être utilisés pour modéliser des actions spécifiques sur la **machine locale** ou sur une **machine distante** afin de préparer des serveurs ou d'autres objets d'infrastructure pour le service.



Les provisionneurs - Type

Il existe de nombreux provisionneurs disponibles

- Le provisionneur **local-exec** : appelle un exécutable local après la création d'une ressource. Cela appelle un processus et non sur la ressource que nous créons.

- Le provisionneur **remote-exec** : permet d' appeler un script sur une ressource distante une fois qu'elle est créée. Nous pouvons fournir une liste de chaînes de commandes qui sont exécutées dans l'ordre où elles sont fournies.

- Le provisionneur **file** : est utilisé pour copier des fichiers ou des répertoires de la machine exécutant Terraform vers la ressource nouvellement créée. Il prend également en charge les connexions de type SSH et WinRM.

Le provisionneur local-exec

- Le provisionneur local-exec invoque un exécutable local après la création d'une ressource.
- Cela appelle un processus sur la machine exécutant Terraform, pas sur la ressource.

```
resource "aws_instance" "web" {
    # ...
    provisioner "local-exec" {
        command = "echo ${self.private_ip} >> private_ips.txt"
    }
}
```

- Les arguments suivants sont pris en charge :
  - ***command-*** (Obligatoire) Il s'agit de la commande à exécuter.
  - ***working_dir***- (Facultatif) spécifie le répertoire de travail où commandsera exécuté.
  - ***interpreter-*** (Facultatif)il une liste d'arguments de l'interpréteur utilisés pour exécuter la commande.
  - ***environment-*** (Facultatif) bloc de paires clé-valeur représentant l'environnement de la commande exécutée.

Le provisionneur remote-exec

- Le provisionneur remote-exec invoque un script sur une ressource distante après sa création. Cela peut être utilisé pour exécuter un outil de gestion de configuration, amorcer un cluster, etc.
- Les arguments suivants sont pris en charge :
  - **inline-** une liste de chaînes de commande. Elles sont exécutés dans l'ordre où elle sont fournis.
  - **script-** Il s'agit d'un chemin (relatif ou absolu) vers un script local qui sera copié sur la ressource distante puis exécuté.
  - **scripts-** Il s'agit d'une liste de chemins (relatifs ou absolus) vers des scripts locaux qui seront copiés sur la ressource distante puis exécutés. Ils sont exécutés dans l'ordre où ils sont fournis. Cela ne peut pas être fourni avec inlineou script.

```
resource "aws_instance" "web" {
  # ...

  provisioner "remote-exec" {
    inline = [
      "puppet apply",
      "consul join ${aws_instance.web.private_ip}",
    ]
  }
}
```

Le provisionneur file

- Utilisé pour copier des fichiers ou des répertoires de la machine exécutant Terraform vers la ressource nouvellement créée.
  - Les arguments suivants sont pris en charge :
    - **source**- Il s'agit du fichier ou dossier source. Cet attribut ne peut pas être spécifié avec content.
    - **content**- C'est le contenu à copier sur la destination. Si la destination est un fichier, le contenu sera écrit sur ce fichier, dans le cas d'un répertoire, un fichier nommé tf-file-content est créé.
    - **destination**- (Obligatoire) Il s'agit du chemin de destination. Il doit être spécifié en tant que chemin absolu.

```
resource "aws_instance" "web" {
  # ...

    # Copies the myapp.conf file to /etc/myapp.conf
    provisioner "file" {
        source      = "conf/myapp.conf"
        destination = "/etc/myapp.conf"
    }

    # Copies the string in content into /tmp/file.log
    provisioner "file" {
        content     = "ami used: ${self.ami}"
        destination = "/tmp/file.log"
    }
}
```

Lab 4 : Les provisioners