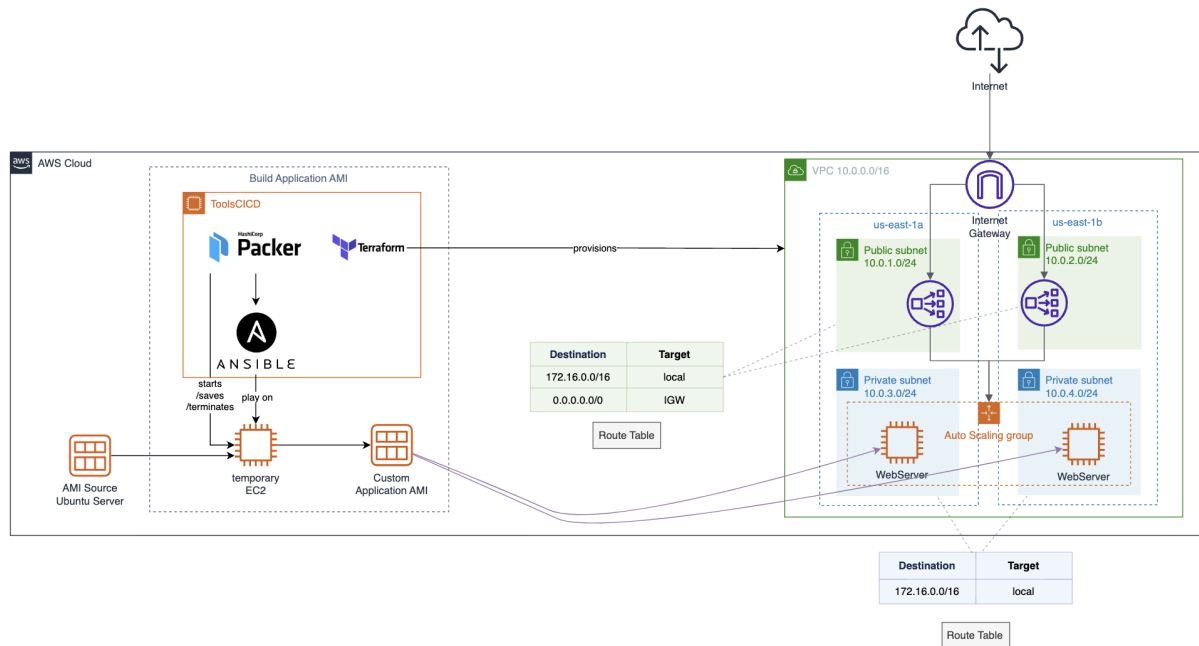


# TP3 - Déployer des ressources sur AWS avec Terraform

## Plan

1. Introduction
2. Introduction à Terraform
  1. Concepts
  2. Terraform Lifecycle
    1. terraform init
    2. terraform plan
    3. terraform apply
    4. terraform destroy
  3. Terraform State
3. Instance ToolsCI/CD
4. Installation de Terraform
5. Premiers pas avec Terraform
  1. Exemple : Récupération de notre AMI WebApp
  2. Le TP1 as Code
  3. Utilisation de l'AMI du TP2
6. Application Web en Haute Disponibilité et réseau privé (optionnel)
  1. Informations de configuration
  2. Ressources à utiliser
  3. Exemples de codes terraform
    1. Faire des boucles sur une liste de valeurs
    2. Exemple d'une launch configuration
    3. Créer un ALB
    4. Obtenir en sortie (output) l'adresse du Load Balancer
  4. Ressources avancées (optionnel)
  5. Troubleshooting
    1. Autoscaling Group
7. Cleanup de vos ressources
8. Questions
  1. Terraform
  2. Autoscaling
  3. AMI (Amazon Images)
  4. Cloud Watch (Contexte Autoscaling)
  5. Load balancer (Autoscaling)

# Introduction



Le but de ce TP est de déployer des infrastructures et ressources AWS via Terraform.

Nous allons voir comment reproduire as code l'infrastructure du TP1.

Comment déployer l'application web du TP2 à l'aide de l'AMI Applicative.

Et finalement, comment automatiser un déploiement complet de l'application en mode hautement disponible par l'utilisation de Load Balancer et d'« Autoscaling Group » AWS.

## 💡 Codes sources

Les exemples de code indiqués dans le TP sont disponibles dans ce repository gitlab :

<https://gitlab.com/efrei-devops/tps>

Une fois connectés sur la machine ToolsCICD, vous pouvez simplement exécuter la commande `git clone`

<https://gitlab.com/efrei-devops/tps> pour récupérer les exemples de code sur la machine dans le dossier `tps`.

# Introduction à Terraform

---



<https://www.terraform.io/intro/index.html>

Terraform est un outil d'**Infrastructure As Code** open-source développé par [HashiCorp](#). Il est utilisé pour définir et provisionner l'infrastructure complète à l'aide d'un langage déclaratif facile à maîtriser.

Avantages à l'utilisation de Terraform :

- Fait de l'orchestration, pas seulement de la gestion de la configuration
- Prend en charge plusieurs providers dont les hyperscalers comme AWS, Azure, GCP et bien d'autres : <https://registry.terraform.io/browse/providers>
- Fournir une infrastructure immuable
- Langage déclaratif simple : HCL (HashiCorp Configuration Language)

## Concepts

- **Variables** : Également utilisé comme variable d'entrée, il s'agit d'une paire clé-valeur utilisée par les modules Terraform pour permettre la personnalisation.
- **Provider** : C'est un plugin pour interagir avec les API de service et accéder à ses ressources associées.
- **Module** : C'est un sous-dossier avec des définitions Terraform pour modulariser ou être réutilisés
- **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.
- **Data Sources** : fourni par les providers pour renvoyer des informations sur les objets externes à terraform.
- **Outputs** : 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 terraform 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.
- **Apply** : C'est l'une des étapes où terraform applique les changements d'état réel / actuel de l'infrastructure afin de passer à l'état souhaité.

## Terraform Lifecycle



### **terraform init**

<https://developer.hashicorp.com/terraform/cli/commands/init>

Cette commande permet d'initialiser un projet Terraform. Il faudra l'exécuter à chaque fois que vous créez un projet Terraform ou lorsque vous ajoutez des providers à vos ressources.

### **terraform plan**

<https://developer.hashicorp.com/terraform/cli/commands/plan>

Cette commande permet d'afficher les ressources qui vont être ajoutées/modifiées/supprimées sans appliquer les changements.

### **terraform apply**

<https://developer.hashicorp.com/terraform/cli/commands/apply>

Cette commande permet de déployer les ressources définies dans le projet.

Si vous modifiez vos fichiers Terraform un simple « terraform apply » permet d'appliquer les modifications apportées.

### **terraform destroy**

<https://developer.hashicorp.com/terraform/cli/commands/destroy>

Cette commande permet de détruire les ressources définies.

## Terraform State

<https://developer.hashicorp.com/terraform/language/state>

Ne jamais supprimer le fichier « tfstate » avant d'avoir effectué un « destroy » car ce fichier stocke l'état de la « Stack », sinon il vous sera impossible de déprovisionner programmatiquement les ressources déployées par Terraform.

D'autre part, évitez de supprimer ou modifier manuellement les ressources déployées par Terraform, car si les ressources ne sont plus existantes et que vous réexécutez Terraform il apparaîtra que Terraform ne retrouve plus ses ressources.

### Gestion du state

Les fichiers `state` ne doivent pas être sauvegardés sous Git.

Pour démarrer, il est plus simple de garder le fichier state localement, mais par la suite il faudra utiliser [une meilleure méthode](#) pour par exemple les partager avec votre équipe ou permettre à vos pipelines CI/CD de les récupérer et les mettre à jour.

N'oubliez pas que votre instance EC2 `ToolsCI` est une ressource qui doit pouvoir être supprimée. Garder un state terraform local à cette instance n'est pas une bonne pratique.

## Instance ToolsCICD

---

Si vous avez détruit l'instance `ToolsCICD` du TP2, vous devez la recréer :

Name Tag	Type	AMI	Subnet	Security Group Rules
ToolsCICD	t2.micro	Ubuntu Server	TP_DevOps_Public	SSH – from your Public IP

- Connectez-vous à l'instance `ToolsCICD` en SSH
- Installez ansible et packer :

```
sudo apt update && \
sudo apt install software-properties-common --yes && \
sudo apt-add-repository --update ppa:ansible/ansible --yes && \
sudo apt install ansible --yes && \
ansible --version && \
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add - && \
sudo apt-add-repository -y "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs)
main" && \
sudo apt-get update && sudo apt-get install -y packer && \
packer -v
```

- Déposez votre fichier playbook `play.yml`
- Déposez le fichier packer `buildAMI.pkr.hcl`
- initialisez les plugins packer : `packer init buildAMI.pkr.hcl`
- buildez l'AMI avec packer : `packer build buildAMI.pkr.hcl`

## Installation de Terraform

---

<https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli>

Sur l'instance `ToolsCICD`, pour installer terraform, exécutez :

```
sudo apt-get update && sudo apt-get install -y gnupg software-properties-common && \
wget -O- https://apt.releases.hashicorp.com/gpg | gpg --dearmor | sudo tee
/usr/share/keyrings/hashicorp-archive-keyring.gpg && \
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/hashicorp.list && \
sudo apt update
sudo apt-get install terraform
```

Vérification de l'installation :

```
terraform -v
```

## Premiers pas avec Terraform

---

Nous allons principalement utiliser le provider terraform `aws` :

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>

La plupart des ressources disponibles présente toujours des exemples pour rapidement les utiliser dans votre code.

### Exemple : Récupération de notre AMI WebApp

Sur l'instance ToolsCICD :

- créez un dossier `tp3`
- allez dans ce dossier et créez un fichier `main.tf` avec cette première définition terraform  
<https://gitlab.com/efrei-devops/tps/-/blob/main/tp3/main.tf>

A présent, exécutez les commandes terraform :

```
terraform init
terraform plan
```

Ce plan ne proposera pas de créer de ressources (`without changing any real infrastructure`) car nous n'avons déclaré qu'une data source (lecture d'informations) et un output (sortie terraform). Il vous affichera l'ID et le nom de votre AMI.

💡 Créez des alias dans votre shell pour taper plus vite les commande terraform :

```
alias tf="terraform"
alias tfi="tf init"
alias tfp="tf plan"
alias tfa="tf apply"
alias tfd="tf destroy"
```

### Le TP1 as Code

Le code terraform équivalent au TP1 est disponible ici :

[https://gitlab.com/efrei-devops/tps/-/blob/main/tp3/tp1\\_as\\_code/main.tf](https://gitlab.com/efrei-devops/tps/-/blob/main/tp3/tp1_as_code/main.tf)

Sauvegardez ce code dans le fichier `main.tf` (utilisez un autre dossier, par exemple `~/tp3/tp1_as_code`) et exécutez les commandes `terraform init` et `terraform plan` :

1. terraform va vous demander de saisir une adresse IP. Saisissez votre adresse IP (récupérable par exemple sur <https://checkip.amazonaws.com/>)
2. puis terraform va vous montrer ce qu'il va créer :

```
...
Plan: 12 to add, 0 to change, 0 to destroy.
...
```

Si vous exécutez `terraform apply`, terraform vous redemandera votre IP et va vous demander validation (tapez `yes`) puis **provisionnera** les ressources.

⚠️ Notez que les ressources vont avoir les mêmes noms "logiques" que certaines déjà existantes du TP1 (VPC, Routes, ...). Attention à ne pas les confondre dans la console.

Vous pouvez alors reproduire les tests du TP1 :

- se connecter en SSH sur l'ip publique de l'instance **NAT\_JumpHost**
- sur **NAT\_JumpHost**, copier la clé privée (format PEM, droits 600) et se connecter en SSH sur l'ip privée de l'instance **PrivateHost**
- depuis **PrivateHost**, effectuez les tests **ping 8.8.8.8** et **curl https://www.google.fr** pour vérifier les accès vers Internet à travers l'instance NAT

A présent exécutez **terraform destroy** ou **tfd** dans le même dossier où vous avez lancé **terraform apply**, terraform vous redemandera votre IP et validation (tapez yes) puis il **supprimera** les ressources.


 Ask ChatGPT

Please give me the terraform code for an AWS VPC named "TP\_DevOps" on CIDR "172.16.0.0/16" with a public subnet named "TP\_DevOps\_Public" on CIDR "172.16.1.0/24" and a private subnet named "TP\_DevOps\_Private" on CIDR "172.16.2.0/24", a NAT Instance named "Nat\_JumpHost" using Linux Ubuntu Server and the already existing IAM Profile named LabInstanceProfile, a private instance named "PrivateHost" using Linux Ubuntu Server and the already existing IAM Profile named LabInstanceProfile accessing Internet through the NAT Instance. Use variables for CIDRs of VPC and subnets

## Utilisation de l'AMI du TP2

A présent vous devez adapter le code terraform du TP1 :

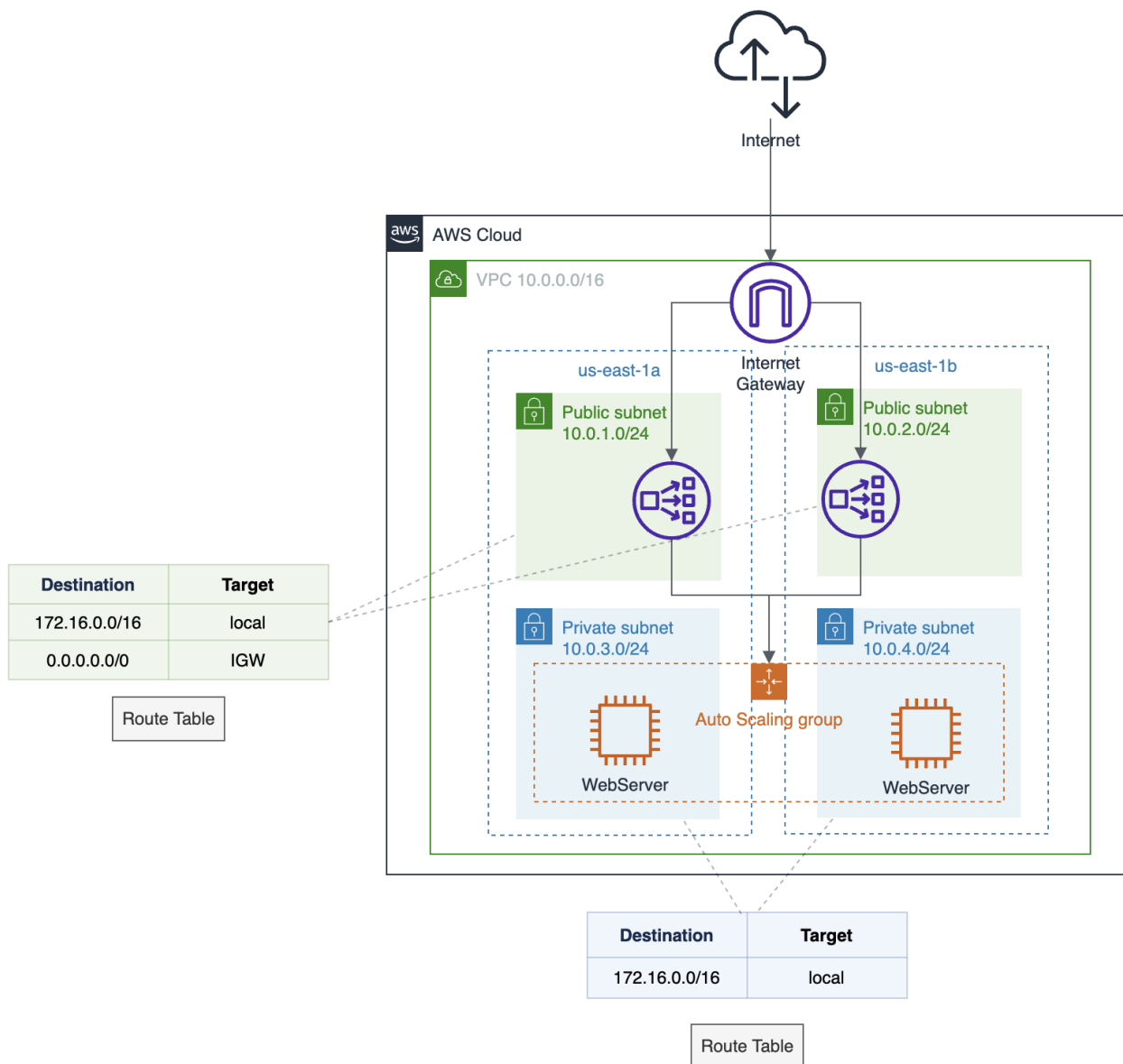
- pour créer une seule instance EC2 :
  - nommée **WebServer**
  - dans le réseau public
  - utilisant l'AMI WebApp que vous avez créée au TP2. (cf [exemple de datasource](#) pour récupérer l'ID de votre AMI)
  - dont le security group **public** autorise les accès (**inbound/ingress**) sur le port 8080 depuis votre adresse IP

 Vous pouvez laisser les ressources réseau **private** dans le code terraform même si inutiles. Mais ne conservez bien qu'une seule ressource **aws\_instance**.

Exécutez les commandes terraform (init et apply), puis vérifiez que vous avez bien accès à l'application web via votre navigateur puis supprimez les ressources avec terraform.

## Application Web en Haute Disponibilité et réseau privé (optionnel)

A partir des codes terraform précédents, vous allez déployer une infrastructure hautement disponible déployée sur 2 zones de disponibilité (AZ).



Cette infrastructure comporte :

- 2 subnets publics et 2 subnets privés
  - les routes tables n'ont pas besoin d'être dupliquées
- un répartiteur de charge (Load Balancer) présent sur les 2 zones
- un Autoscaling group capable de démarrer des instances EC2 **WebServer** utilisant votre AMI dans les 2 zones
  - un Autoscaling group nécessite une launch configuration. C'est à dire les propriétés des instances EC2 à lancer.

### Informations de configuration

VPC :

Name Tag	CIDR
WebVpc	10.0.0.0/16



Subnets :

Name Tag	AZ	CIDR
WebPublic-1	us-east-1a	10.0.1.0/24
WebPublic-2	us-east-1b	10.0.2.0/24
WebPrivate-1	us-east-1a	10.0.3.0/24
WebPrivate-2	us-east-1b	10.0.4.0/24

### Ressources à utiliser

- toutes celles déjà vues précédemment
- - Autoscaling group et Launch configuration :
    - [https://www.terraform.io/docs/providers/aws/r/launch\\_configuration.html](https://www.terraform.io/docs/providers/aws/r/launch_configuration.html)
    - [https://www.terraform.io/docs/providers/aws/r/autoscaling\\_group.html](https://www.terraform.io/docs/providers/aws/r/autoscaling_group.html)
  - Load Balancer :
    - Classic (ELB) : <https://www.terraform.io/docs/providers/aws/r/elb.html>
    - ou NLB/ALB : <https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/lb>

### Exemples de codes terraform

#### Faire des boucles sur une liste de valeurs

```
variable "availability_zones" {
  default = ["us-east-1a", "us-east-1b"]
}

variable "public_subnet_cidrs" {
  default = ["10.0.1.0/24", "10.0.2.0/24"]
}

...

# Create the public subnets
resource "aws_subnet" "public_subnets" {
  count                = length(var.availability_zones)
  vpc_id              = aws_vpc.web.id
  cidr_block          = var.public_subnet_cidrs[count.index]
  availability_zone    = var.availability_zones[count.index]
  map_public_ip_on_launch = true
  tags = {
    Name = "WebPublic-${count.index}"
  }
}

...

resource "aws_route_table_association" "public" {
  count          = length(aws_subnet.public_subnets)
  subnet_id     = aws_subnet.public_subnets[count.index].id
  route_table_id = aws_route_table.public_route_table.id
}
```

**i** Cette utilisation de boucles permettrait de faire varier votre infrastructure sur plus ou moins 2 zones de disponibilité par simple paramétrage.

#### Exemple d'une launch configuration



```
# ASG Launch Configuration
resource "aws_launch_configuration" "web" {
  image_id           = data.aws_ami.web_ami.id
  instance_type      = var.instance_type
  key_name           = "vockey"
  iam_instance_profile = "LabInstanceProfile"
  security_groups    = [aws_security_group.private.id]
  lifecycle {
    create_before_destroy = true
  }
}
```

## Créer un ALB

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/lb>

3 ressources sont nécessaires : 1 aws\_lb, 1 aws\_listener, 1 aws\_target\_group :

```
# Load Balancer
resource "aws_lb" "web" {
  name                = "web"
  internal            = false
  load_balancer_type  = "application"
  subnets            = [for subnet in aws_subnet.public_subnets : subnet.id]
  enable_deletion_protection = false
}

resource "aws_lb_target_group" "web" {
  name     = "web"
  port     = 8080
  protocol = "HTTP"
  vpc_id   = aws_vpc.web.id
  health_check {
    enabled            = true
    healthy_threshold = 3
    interval           = 15
    path              = "/"
    port              = 8080
    timeout           = 5
    unhealthy_threshold = 3
  }
}

resource "aws_lb_listener" "front_end" {
  load_balancer_arn = aws_lb.web.arn
  port              = "8080"
  protocol          = "HTTP"
  default_action {
    type = "forward"
    target_group_arn = aws_lb_target_group.web.arn
  }
}
```

**i** Il sera nécessaire de déclarer le target group dans l'auto scaling group :

```
resource "aws_autoscaling_group" "web" {
  ...
  target_group_arns = [aws_lb_target_group.web.arn]
  ...
}
```

Obtenir en sortie (output) l'adresse du Load Balancer

```
# DNS name of the ELB (wait for a few minutes before it becoming accessible on first deployment)
output "lb_dns_name" {
  description = "The DNS name of the ELB"
  value       = aws_lb.web.dns_name
}
```

## Ressources avancées (optionel).

- Pour permettre à l'autoscaling group d'effectuer les actions de « Scale In » et « Scale Out », il faut spécifier des alarmes Cloud Watch et des Autoscaling Policies
  - [https://www.terraform.io/docs/providers/aws/r/cloudwatch\\_metric\\_alarm.html](https://www.terraform.io/docs/providers/aws/r/cloudwatch_metric_alarm.html)
  - [https://www.terraform.io/docs/providers/aws/r/autoscaling\\_policy.html](https://www.terraform.io/docs/providers/aws/r/autoscaling_policy.html)
- Récupération dynamique des zones de disponibilité :
   
[https://www.terraform.io/docs/providers/aws/d/availability\\_zones.html](https://www.terraform.io/docs/providers/aws/d/availability_zones.html)

## Troubleshooting

### **Autoscaling Group**

Lors de la création de l'auto scaling group, vous obtenez une erreur du type :

```
|
| Error: waiting for Auto Scaling Group (terraform-20230411102745810400000004) capacity satisfied: 2
| errors occurred:
|   * Scaling activity (53a61dbf-6df0-c4c7-92e3-8f2d378dd1ec): Failed: Access denied when
| attempting to assume role arn:aws:iam::767874509532:role/aws-service-
| role/autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling. Validating load balancer configuration
| failed.
|   * Scaling activity (15d61dbf-6dec-0d01-2b5b-ef0904e23fc1): Failed: Access denied when
| attempting to assume role arn:aws:iam::767874509532:role/aws-service-
| role/autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling. Validating load balancer configuration
| failed.
```

=> Relancez terraform car un rôle est créé automatiquement par AWS et n'est pas prêt lors de la toute première utilisation d'un AutoScaling Group dans un compte AWS.

## Cleanup de vos ressources

---

1. exécutez, si vous ne l'avez pas déjà fait, la commande **terraform destroy** dans vos répertoires de travail terraform (~/tp1, ~/tp3)
2. Nettoyez les AMI que vous créez (en 2 steps: Deregister AMI + Delete Snapshot). Si vous n'avez pas encore terminé votre TP, conservez tout de même votre AMI la plus récente jusqu'à ce que vous terminiez le TP. Autrement vous devrez répéter les opérations ("Instance ToolsCICD") pour builder l'AMI.

## Questions

---

### Terraform

- Conceptuelle : Quels sont les avantages de l'utilisation de Terraform pour le déploiement d'infrastructures par rapport à des méthodes manuelles ou à d'autres outils d'automatisation ?
- Application : Après avoir déployé une application web avec Terraform, comment vérifieriez-vous son bon fonctionnement ?
- Analyse : Si Terraform échoue lors de l'application d'un plan, quelles étapes suivriez-vous pour diagnostiquer et résoudre le problème ?

- Critique : Discutez des limites de Terraform en matière de gestion de l'infrastructure. Y a-t-il des scénarios où il ne serait pas l'outil idéal ?
- Extension : Comment intégreriez-vous Terraform avec d'autres outils de DevOps pour un pipeline de déploiement continu ?
- Sécurité : Quelles sont les meilleures pratiques pour assurer la sécurité lors du déploiement d'infrastructures avec Terraform sur AWS ?
- Optimisation : Comment optimiseriez-vous les coûts lors du déploiement d'une infrastructure avec Terraform sur AWS ?
- Évolution : Comment Terraform peut-il être utilisé pour gérer les changements dans une infrastructure AWS existante ?
- Cas d'usage : Proposez un scénario où l'utilisation de Terraform serait particulièrement bénéfique dans un contexte de production.
- Réflexion : Quelles sont les conséquences potentielles de la suppression manuelle de ressources AWS qui ont été déployées via Terraform ?

### Autoscaling

- Compréhension : Comment l'autoscaling fonctionne-t-il avec AWS et quel est son rôle dans la gestion des ressources ?
- Configuration : Quelles sont les étapes pour configurer une politique d'autoscaling avec Terraform sur AWS ?
- Scénarios : Dans quels scénarios l'utilisation de l'autoscaling serait-elle particulièrement utile ? Donnez des exemples concrets.
- Dépannage : Si une instance ne se lance pas comme prévu dans un groupe d'autoscaling configuré via Terraform, quelles seraient vos premières étapes de dépannage ?
- Optimisation : Comment optimiseriez-vous les coûts tout en garantissant la performance avec l'autoscaling sur AWS ?
- Sécurité : Quels sont les enjeux de sécurité à considérer lors de la mise en place de l'autoscaling avec Terraform sur AWS ?
- Évaluation : Comment évalueriez-vous l'efficacité d'une stratégie d'autoscaling mise en place via Terraform sur AWS ?
- Mise à jour : Quelles sont les meilleures pratiques pour mettre à jour les AMI dans un groupe d'autoscaling sans interrompre le service ?

### AMI (Amazon Images)

- Optimisation : Comment construire une AMI pour optimiser les performances et la réactivité dans un environnement d'autoscaling ?
- Sécurité : Quelles mesures de sécurité spécifiques doivent être prises lors de l'utilisation d'AMI personnalisées dans un groupe d'autoscaling ?
- Dépannage : Comment identifier et résoudre les problèmes liés à l'utilisation d'une AMI spécifique dans un groupe d'autoscaling ?

### Cloud Watch (Contexte Autoscaling)

- Fonctionnement : Comment les alarmes CloudWatch interagissent-elles avec les groupes d'autoscaling dans AWS ?
- Configuration : Quelles étapes suivez-vous pour configurer une alarme CloudWatch qui déclenche l'autoscaling sur AWS avec Terraform ?
- Sensibilité : Comment déterminer les seuils appropriés pour les alarmes CloudWatch dans un contexte d'autoscaling ?
- Analyse : Quels types de métriques CloudWatch sont les plus utiles pour gérer efficacement l'autoscaling ?
- Optimisation : Comment l'utilisation des alarmes CloudWatch peut-elle améliorer l'efficacité de l'autoscaling en termes de coûts et de performance ?

### Load balancer (Autoscaling).

- Load balancer : Pouvez-vous expliquer les différences clés entre un Classic Load Balancer, un Network Load Balancer et un Application Load Balancer sur AWS ?
- Rôle fondamental : Quel est le rôle principal d'un load balancer dans un environnement avec autoscaling sur AWS ?
- Distribution de trafic : Comment un load balancer gère-t-il la distribution du trafic entrant dans un groupe d'autoscaling ?
- Scalabilité : Comment le load balancing contribue-t-il à la scalabilité d'une application déployée sur AWS avec l'autoscaling ?
- Haute disponibilité : En quoi un load balancer est-il essentiel pour assurer la haute disponibilité dans un système avec l'autoscaling ?