

# TP 2 - Ope Tofu 🍔 et la création d'infrastructure avec du code



## Mise en place

Allez sur la plateforme AWS Academy et accédez au cours AWS Academy Learner Lab. Puis cliquez sur **Modules > Learner Lab**. Lancez votre environnement en cliquant sur **Start Lab**. Une fois le cercle passé au vert, cliquez sur **AWS Details** et **AWS CLI**. Ces clés permettent un accès programmatique à votre compte.

Ouvrez un terminal et exécutez la commande `aws configure`. Un prompt va vous demander votre AWS Access Key ID, validez à vide, et faites pareil pour la Secret AccessKey. Cherchez le dossier caché `.aws` (dans votre dossier personnel sous Linux et Mac, dans votre dossier user sur windows) et collez le contenu de **AWS CLI** dans le fichier **credentials**.

Créez un dossier `cloud-computing` avec la commande `mkdir "cloud-computing"`. Déplacez-vous dans le dossier avec la commande `cd "cloud computing"`. Clonez le dépôt git du TP avec un `git clone https://github.com/HealerMikado/Ensai-CloudComputingLab2.git`.

## Mon premier script avec OpenTofu

### Une instance de base

Ouvrez le fichier `infra.tf`. Il contient l'architecture minimale du code nécessaire pour que vous puissiez réaliser le TP.

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }

  required_version = ">= 1.5.0"
}

provider "aws" {
  region = "us-east-1"
}

# Pour récupérer l'IP de la VM
output "instance_public_ip" {
  description = "Adresse IP publique de l'instance EC2"
  value       = aws_instance.ubuntu_vm.public_ip
}
```

Maintenant, vous allez définir votre première **ressource**. Elle sera de type **aws\_instance** et vous l'appelerez **ubuntu\_vm**. Le doc complète de cette ressource se trouve ici : <https://search.opentofu.org/provider/hashicorp/aws/latest/docs/resources/instance>

Ensuite, via des paramètres, vous allez définir un peu plus en détail votre instance. Rappelez-vous, pour une instance EC2, il vous faut définir son OS (appelé AMI chez AWS) et le type d'instance.

Ajoutez à votre instance son *AMI* avec le paramètre **ami** qui prendra comme valeur **ami-0ecb62995f68bb549** (c'est l'identifiant de l'AMI Ubuntu dans la région **us-east-1**), et pour le type d'instance, vous prendrez une **t3.micro**. Exécutez votre architecture avec la commande **tofu init** pour initialiser OpenTofu et **tofy apply** pour créer votre infrastructure (les deux sont à faire dans un terminal). Connectez-vous au tableau de bord EC2 et vérifiez que votre instance est bien démarrée. Néanmoins, si vous essayez de vous connecter en SSH à votre instance, vous allez voir que c'est impossible. En effet, lors de la définition de l'instance, nous n'avons pas défini la clé SSH à utiliser et le *security group* de l'instance. Tout cela fait que, pour le moment, l'instance n'est pas accessible.

## Configuration de la partie réseau

Vu que ce n'est pas intéressant à trouver seul, voici le code pour définir le *security group* de l'instance :

```
resource "aws_security_group" "web_sg" {
  name      = "web-sg"
  description = "Autorise SSH (22) et HTTP (80) en entrée, tout en sortie"
  vpc_id    = data.aws_vpc.default.id

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

  ingress {
    description = "HTTP"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    description = "Tout trafic sortant"
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "web-sg"
  }
}
```

```

    }
}
```

Ce *Security Group* n'accepte que les connexions HTTP et SSH en entrée et permet tout le trafic en sortie. Pour associer ce *Security Group* à votre instance, vous allez devoir ajouter un paramètre `vpc_security_group_ids` lors de la création de l'objet. Attention, ce paramètre attend une liste d'id de *Security Groups*. pour récupérer l'id fait simplement `aws_security_group.web_sg.id`. Et pour avoir une liste, mettez le tout entre `[]`. Pour définir la clé, ajoutez le paramètre `key_name` avec comme valeur le nom de la clé (`vockey`). Vous pouvez maintenant relancer votre instance avec un nouveau `tofu apply`. Cela va résilier l'instance précédente et en créer une nouvelle.

## Configuration des user data

Pour le moment, nous n'avons pas défini les *user data* de l'instance. Pour les ajouter, il faut simplement ajouter le paramètre `user_data_base64` avec comme valeur la fonction `base64encode()` en lui passant comme argument `file("${path.module}/user_data.sh")`. Cela va encoder en base64 le contenu du fichier `user_data.sh` qui se trouve dans le dossier du TP.

## Configuration des disques (bonus)

Actuellement, l'instance créée n'a qu'un disque de 8 Go. C'est suffisant, mais il est possible de changer cela via OpenTofu. Par exemple, ajoutez ce bout de code à votre instance.

```

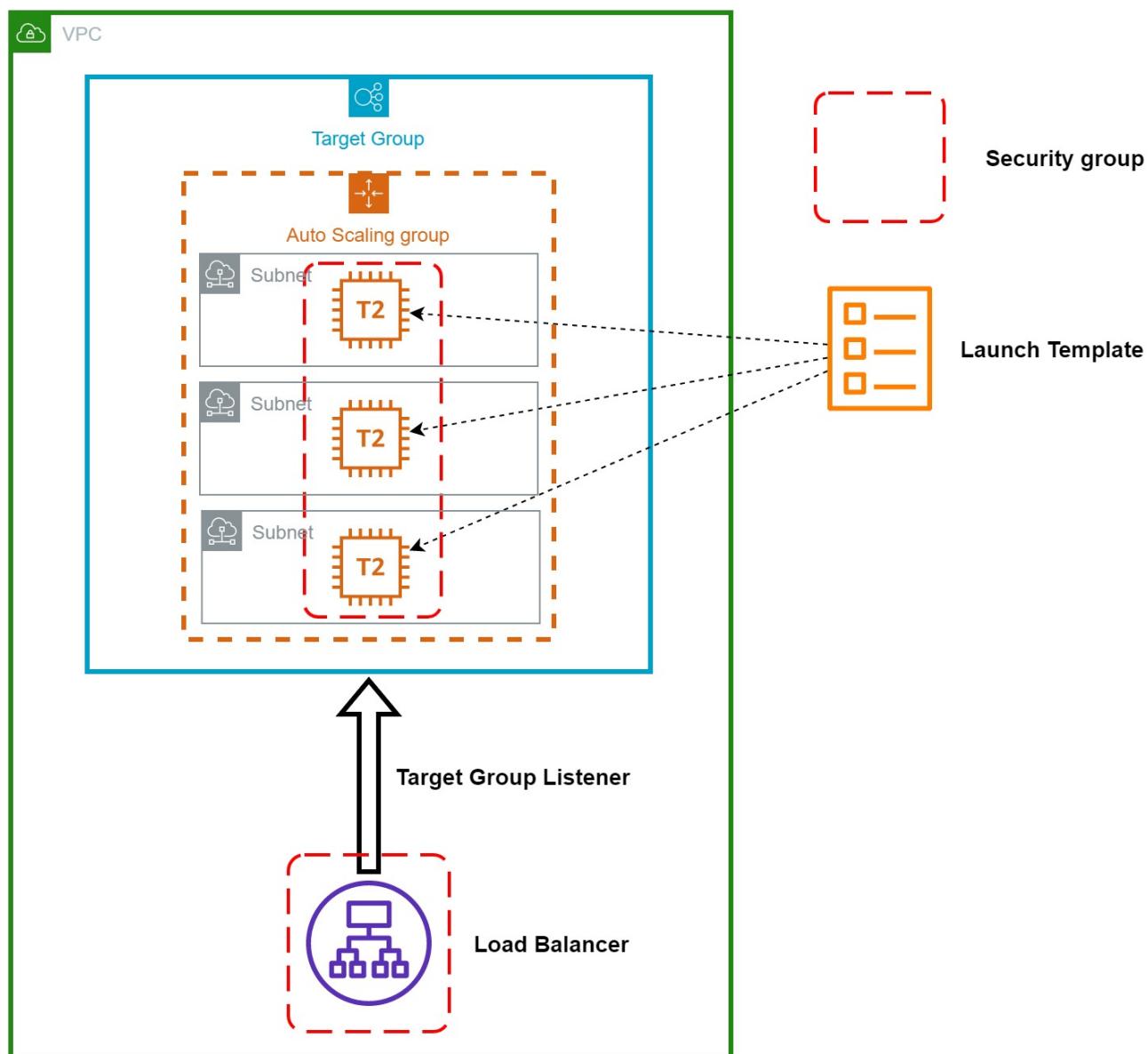
# Disque principal (root)
root_block_device {
    volume_size = 20
    volume_type = "gp3"
}
ebs_block_device {
    device_name = "/dev/sdf"    # nom logique dans EC2
    volume_size = 100
    volume_type = "gp3"
    delete_on_termination = true
}
```

Le premier disque de l'instance aura ainsi un volume de 20 Go, et un second disque sera attaché avec un volume de 100 Go. Les deux disques seront supprimés en même temps que l'instance. Vous pouvez voir les deux disques en vous connectant à l'instance en SSH et en exécutant la commande `df -h` (*disk free*).

## Mise en place d'un Auto Scaling Group et d'un Load Balancer

Ci-dessous, vous trouverez l'architecture finale que vous allez mettre en place pour ce TP. Elle est un peu plus détaillée que lors du précédent TP pour faire apparaître chaque élément que vous allez devoir définir. Se détacher de l'interface graphique pour utiliser un outil IaC fait réaliser à quel point la console AWS masque de nombreux détails. Tout implémenter n'est pas difficile, mais est laborieux quand on n'est pas guidé. Toutes les étapes sont découpées pour être unitaires et simples. Elles consistent toutes à définir un

objet Python avec la bonne classe et les bons paramètres. Ce n'est pas simple de trouver cela seul, alors je vous donne tout. Il suffit de suivre le TP à votre rythme.



Vous trouverez le code de cet exercice dans le dossier `ex_2 haute dispo`.

## Launch Template

La première étape va être de définir le *template* des instances de l'*Auto Scaling Group*. Pour cela, vous allez utiliser la ressource `aws_launch_template`. Comme un *template* est quasiment la même chose qu'une instance, l'objet `aws_launch_template` va fortement ressembler à une instance, seuls les noms des paramètres vont changer (oui, il n'y a pas de cohérence sur les noms). Ainsi, votre objet `aws_launch_template` va avoir comme paramètres :

- `image_id` qui va définir son image AMI,
- `instance_type` qui va définir le type d'instance,
- `user_data` qui va définir les user data. Attention, même si ce n'est pas précisé, elles doivent bien être encodées en base 64,
- `vpc_security_group_ids` au lieu de `security_groups` pour la liste des *security groups*,
- `key_name` pour la clé SSH à utiliser.

## Auto Scaling Group

Maintenant que le *template* est défini, c'est le moment de l'utiliser avec un *Auto Scaling Group*. Souvenez-vous, un *Auto Scaling Group* va maintenir un nombre d'instances compris entre le min et le max défini. La classe qui représente un ASG est simplement `aws_autoscaling_group`. Elle prend en paramètres :

- `min_size`, `max_size` et `desired_capacity` pour la limite inférieure, supérieure, et la valeur initiale,
- `launch_template` qui permet de spécifier le *template* à utiliser. Vous pouvez passer un dictionnaire contenant uniquement les clé `id` avec comme valeur l'id du *launch template* que vous obtiendrez avec l'attribut `id` du *launch template* (`aws_launch_template.non_de_votre_lauche_template.id`) et la clé `version` que vous mettrez à `"$Latest"`,
- `vpc_zone_identifier` pour spécifier les sous-réseaux à utiliser pour l'Auto Scaling Group. Laissez la valeur fournie.

Il ne vous reste plus qu'à lancer votre code. Il va créer les sous-réseaux nécessaires, un Launch Template et un ASG selon vos spécifications. Attendez quelques instants puis allez sur le tableau de bord EC2, vous devriez voir apparaître 3 instances.

## Elastic Load Balancer

Dernière pièce à définir, le *Load Balancer* va avoir la charge de répartir les requêtes entre les instances. La création via l'interface a caché pas mal de choses et, au lieu de créer un simple objet, il faut en créer 3 :

- le *Load Balancer* en tant que tel,
- le *Target Group* qui va permettre de considérer l'ASG comme une cible possible pour le *Load Balancer*,
- et un *Load Balancer Listener* pour relier les deux.

## Load Balancer

Définir le *Load Balancer* est assez simple. Cela se fait avec la ressource `aws_lb`. Elle prend en paramètre :

- son type avec le paramètre `load_balancer_type`. Dans le cas présent, cela sera "application",
- les sous-réseaux avec lesquels il communique avec le paramètre `subnets`. Laissez la valeur `subnets` déjà définie,
- et les groupes de sécurité qui lui sont associés avec le paramètre `security_groups`. Le `security group` déjà défini convient très bien. Attention, ce paramètre attend une liste.

## Target Group

Le *Target Group* est également facile. Utilisez la classe `LbTargetGroup` et passez la stack et un id. Il vous faut ensuite définir les paramètres :

- `port` en spécifiant le port 80 et `protocol` en spécifiant `HTTP` car nous voulons que le TG soit accessible uniquement en HTTP sur le port 80,
- `vpc_id` avec l'id du VPC déjà défini. Cela est nécessaire car cela permet à AWS de savoir que les machines du *Target Group* seront dans le réseau.

Il faut maintenant associer votre *Target Group* à votre ASG. Cela passe par l'ajout d'un attribut `target_group_arns` dans l'ASG. Cet attribut attend la liste des ARN (Amazon Resource Names) des Target

Groups. Votre *Target Group* expose son ARN via l'attribut **arn**.

## Load Balancer Listener

Il ne nous reste plus qu'à dire au *Load Balancer* de forwarder les requêtes HTTP vers notre *Target Group*. Il faut utiliser l'objet **aws\_lb\_listener** pour ça. Il prend, en plus des paramètres habituels :

- **load\_balancer\_arn** qui est l'arn du Load Balancer. Pour récupérer l'arn de votre Load Balancer, utilisez l'attribut **arn**,
- **port** qui va prendre la valeur 80 car nous allons forwarder les requêtes faites sur le port 80,
- **protocol** qui va prendre la valeur HTTP car nous allons forwarder les requêtes HTTP,
- **default\_action** où nous allons dire ce que nous voulons faire, ici forwarder les requêtes vers notre *Target Group*. Vous allez mettre

```
default_action {  
    type          = "forward"  
    target_group_arn = aws_lb_target_group.votre_target_groupe.arn  
}
```

Vous pouvez maintenant relancer votre code en vous plaçant dans le dossier de l'exercice, un **tofu init** et un **tofu apply**, allez sur la page du load balancer, obtenir son adresse dns et accéder au endpoint **/instance**. Rafraîchissez la page et l'ID affichez devrait changer régulièrement.

## Conclusion

Vous venez lors de ce TP de créer via du code toute une infrastructure informatique. Même si cela n'est pas simple à faire, le code que vous avez écrit peut être maintenant réutiliser à volonté et versionné via git. Il est ainsi partageable, et vous pouvez voir son évolution. Il peut également être utilisé dans un pipeline de CI/CD pour que l'architecture soit déployée automatiquement.

Même si les solutions IaC ont des avantages, je ne vous les recommande pas pour découvrir un service. Explorer l'interface dans un premier temps pour voir les options disponibles permet de mieux comprendre le service. Automatiser la création de services via du code par la suite si c'est nécessaire.