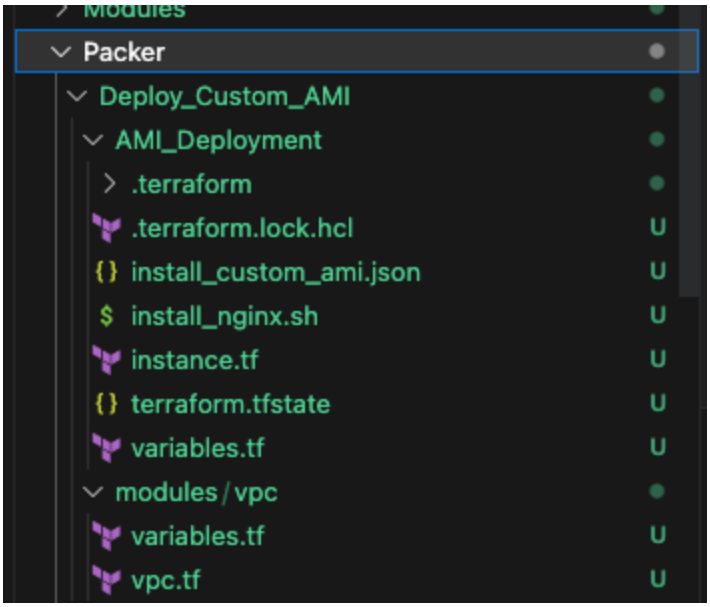
Terraform + Packer

Structure du projet



1- Module VPC

variables.tf

```
#Define Variable for Custom Module VPC

variable "AWS_REGION" {
    type    = string
    default = "us-east-2"
}
```

```
variable "ENVIRONMENT" {
    type    = string
    default = ""
}
```

Ce morceau de code définit deux variables pour un module Terraform personnalisé lié à un Virtual Private Cloud (VPC) sur AWS.

1. `AWS_REGION` : Cette variable définit la région AWS où le VPC sera créé. Le type de cette variable est une chaîne de caractères (`string`) et la valeur par défaut est "us-east-2".
2. `ENVIRONMENT` : Cette variable est utilisée pour spécifier l'environnement dans lequel le VPC sera déployé, comme "production", "staging", etc. Encore une fois, le type est une chaîne de caractères (`string`). Par défaut, cette variable est vide, ce qui signifie qu'elle doit être fournie lors de l'exécution du module, sauf si une valeur par défaut est spécifiée ultérieurement.

En résumé, ces variables permettent de personnaliser le comportement du module VPC en fonction de la région AWS et de l'environnement cible.

VPC.tf

```
#Custom VPC for my Project
module "VPC-01" {
  source = "terraform-aws-modules/vpc/aws"

  name = "vpc-${var.ENVIRONMENT}"
  cidr = "10.0.0.0/16"

  azs             = ["${var.AWS_REGION}a", "${var.AWS_REGION}b", "${var.AWS_REGION}c"]
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets  = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]

  enable_nat_gateway = false
  enable_vpn_gateway = false

  tags = {
    Terraform   = "true"
    Environment = var.ENVIRONMENT
  }
}

#Output Specific to Custom VPC
```

```
output "VPC-ID" {
  description = "VPC ID"
  value       = module.VPC-01.vpc_id
}

output "PRIVATE_SUBNETS" {
  description = "List of IDs of private subnets"
  value       = module.VPC-01.private_subnets
}

output "PUBLIC_SUBNETS" {
  description = "List of IDs of public subnets"
  value       = module.VPC-01.public_subnets
}
```

Ce code Terraform crée un Virtual Private Cloud (VPC) personnalisé pour votre projet en utilisant un module VPC d'AWS. Voici une explication des principaux composants :

Module VPC

1. `source` : Spécifie la source du module, en l'occurrence un module VPC AWS prédéfini.
2. `name` : Nom du VPC, défini en fonction de la variable `ENVIRONMENT`.
3. `cidr` : Bloc d'adresses CIDR pour le VPC.
4. `azs` : Zones de disponibilité où les sous-réseaux seront créés, en utilisant la variable `AWS_REGION`.
5. `private_subnets` **et** `public_subnets` : Blocs d'adresses CIDR pour les sous-réseaux privés et publics.
6. `enable_nat_gateway` **et** `enable_vpn_gateway` : Indiquent si une passerelle NAT ou VPN doit être créée. Ici, les deux sont désactivées.
7. `tags` : Étiquettes pour identifier les ressources.

Sorties (Outputs)

1. `VPC-ID` : L'ID du VPC créé.
2. `PRIVATE_SUBNETS` : Liste des IDs des sous-réseaux privés.
3. `PUBLIC_SUBNETS` : Liste des IDs des sous-réseaux publics.

Ces sorties vous permettent d'accéder facilement à des informations spécifiques sur le VPC après sa création, comme son ID et les IDs de ses sous-réseaux.

2- Groupe de sécurité (Dans instance.tf)

```
#Security Group for Instances
resource "aws_security_group" "SEC-GRP-SSH" {
  vpc_id      = module.VPC-01-DEV.VPC-ID
  name        = "allow-ssh-${var.ENV}"
  description = "security group that allows ssh traffic"

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

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

  tags = {
    Name         = "allow-ssh"
    Environmnent = var.ENV
  }
}
```

Ce bloc de code Terraform définit un groupe de sécurité AWS pour des instances EC2. Voici les détails :

Groupe de sécurité pour les instances EC2

- `vpc_id` : Attribue ce groupe de sécurité au VPC créé par le module `VPC-01-DEV`.
- `name` : Nomme le groupe de sécurité en y ajoutant la valeur de la variable `ENV` (environnement).
- `description` : Fournit une description pour ce groupe de sécurité, indiquant qu'il permet le trafic SSH.

Règles de trafic sortant (egress)

- `from_port` **et** `to_port` : La plage de ports pour le trafic sortant est définie de 0 à 0, ce qui signifie tous les ports.
- `protocol` : Le protocole "-1" signifie que tous les protocoles sont autorisés.
- `cidr_blocks` : Autorise le trafic sortant vers toutes les adresses IP (0.0.0.0/0).

Règles de trafic entrant (ingress)

- `from_port` **et** `to_port` : La plage de ports pour le trafic entrant est fixée à 22, ce qui est le port standard pour SSH.
- `protocol` : Le protocole "tcp" est utilisé pour le trafic SSH.
- `cidr_blocks` : Autorise le trafic entrant depuis toutes les adresses IP (0.0.0.0/0).

Étiquettes (tags)

- Les étiquettes sont utilisées pour identifier facilement ce groupe de sécurité. L'environnement (`ENV`) est également inclus comme étiquette.

Notez que laisser le CIDR à 0.0.0.0/0 pour le trafic entrant sur le port SSH est généralement considéré comme une mauvaise pratique en termes de sécurité, à moins que des mesures de sécurité supplémentaires ne soient en place.

3- Instance EC2

variables.tf

```
# Variable for Create Instance Module
variable "KEY_PUB_PATH" {
  description = "Public key path"
  default = "~/.ssh/levelup_key.pub"
}

variable "ENV" {
    type    = string
    default = "dev"
}

variable "AWS_REGION" {
```

```
default = "us-east-2"
}a

variable "INSTANCE_TYPE" {
  default = "t2.micro"
}


variable "AMI_ID" {
  type = string
  default = ""
}
```

Ce fragment de code Terraform définit des variables qui crée une instance EC2 sur AWS. Voici une explication des différentes variables :

1. `KEY_PUB_PATH` : Il s'agit du chemin vers la clé publique SSH qui sera utilisée pour accéder à l'instance. Par défaut, ce chemin est `~/.ssh/levelup_key.pub`.
2. `ENV` : Cette variable représente l'environnement dans lequel l'instance sera déployée, comme "dev" pour développement ou "prod" pour production. Par défaut, la valeur est "dev".
3. `AWS_REGION` : Définit la région AWS où l'instance sera créée. La valeur par défaut est "us-east-2".
4. `INSTANCE_TYPE` : Spécifie le type d'instance EC2 à créer, par exemple "t2.micro". La valeur par défaut est "t2.micro".
5. `AMI_ID` : L'ID de l'Amazon Machine Image (AMI) qui sera utilisée pour créer l'instance. Cette variable est de type `string` et sa valeur par défaut est vide, ce qui signifie qu'il faut la fournir lors de l'exécution du module.

Note : Il semble y avoir une petite erreur de syntaxe avec un "a" erroné après la définition de la variable `AWS_REGION`.

En somme, ces variables permettent de personnaliser le comportement de l'instance en fonction de différents paramètres comme la région AWS, le type d'instance, et l'AMI à utiliser.

instance.tf

```
# Create Instance uisng Custom VPC

module "VPC-01-DEV" {
  source = "../modules/vpc"

  ENVIRONMENT = var.ENV
  AWS_REGION = var.AWS_REGION
}

provider "aws" {

  region = var.AWS_REGION

}

#Resource key pair
resource "aws_key_pair" "levelup_key" {
  key_name      = "levelup_key"
  public_key    = file(var.KEY_PUB_PATH)
}

#Security Group for Instances
resource "aws_security_group" "SEC-GRP-SSH" {
  vpc_id      = module.VPC-01-DEV.VPC-ID
  name        = "allow-ssh-${var.ENV}"
  description = "security group that allows ssh traffic"

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

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

  tags = {
    Name         = "allow-ssh"
    Environmnent = var.ENV
  }
}

# Create Instance Group
resource "aws_instance" "EC2-01" {
  ami           = var.AMI_ID
  instance_type = var.INSTANCE_TYPE
```

```
  # the VPC subnet
  subnet_id = element(module.VPC-01-DEV.PUBLIC_SUBNETS, 0)
  availability_zone = "${var.AWS_REGION}a"

  # the security group
  vpc_security_group_ids = ["${aws_security_group.SEC-GRP-SSH.id}"]

  # the public SSH key
  key_name = aws_key_pair.levelup_key.key_name

  # attribuer une adresse IP publique à l'instance
  associate_public_ip_address = true

  tags = {
    Name         = "instance-${var.ENV}"
    Environmnent = var.ENV
  }
}
```

Module VPC

1. `source` : Indique que le module VPC personnalisé se trouve dans un répertoire local "../modules/vpc".
2. `ENVIRONMENT` **et** `AWS_REGION` : Ces variables sont passées au module VPC pour définir l'environnement et la région AWS.

Fournisseur AWS

- `region` : Définit la région AWS pour le fournisseur, en utilisant la variable `AWS_REGION`.

Ressource de la paire de clés

- `aws_key_pair`: Crée une paire de clés pour SSH avec le nom "levelup_key" et utilise la clé publique définie dans `var.KEY_PUB_PATH`.

Groupe de sécurité

- `aws_security_group`: Crée un groupe de sécurité qui autorise le trafic SSH. Ce groupe est associé au VPC créé par le module personnalisé.

Création d'une instance EC2

1. `ami` **et** `instance_type` : Utilise les variables `AMI_ID` et `INSTANCE_TYPE` pour définir l'AMI et le type de l'instance.
2. `subnet_id` : Utilise le premier sous-réseau public du VPC personnalisé pour cette instance.
3. `availability_zone` : Définit la zone de disponibilité où l'instance sera lancée.
4. `vpc_security_group_ids` : Attribue au groupe de sécurité précédemment créé à cette instance.
5. `associate_public_ip_address` **:** À `true` pour s'assurer que l'instance reçoit une adresse IP publique lors de sa création
6. `key_name` : Utilise la paire de clés créée précédemment pour cette instance.
7. `tags` : Ajoute des étiquettes pour identifier facilement l'instance.

Dans l'ensemble, ce script Terraform crée une instance EC2 avec des configurations spécifiques, y compris la paire de clés SSH, le groupe de sécurité et d'autres paramètres, tout en utilisant un VPC personnalisé.

4- Configurations Packer

install_custom_ami.json

```
{
    "variables": {
        "aws_access_key": "AKIAVNEQO7RYX7UP4D5R",
        "aws_secret_key": "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"
    },
    "builders": [
        {
            "type": "amazon-ebs",
            "access_key": "{{user `aws_access_key`}}",
            "secret_key": "{{user `aws_secret_key`}}",
            "region": "us-east-2",
            "source_ami_filter": {
                "filters": {
                    "virtualization-type": "hvm",
                    "name": "ubuntu/images/*ubuntu-xenial-16.04-amd64-server-*",
                    "root-device-type": "ebs"
                },
                "owners": [
```

```
                    "099720109477"
                ],
                "most_recent": true
            },
            "instance_type": "t2.micro",
            "ssh_username": "ubuntu",
            "ami_name": "scenario2-packer-{{timestamp}}",
            "tags": {
                "Name": "Nom_De_Votre_AMI",
                "Environment": "Production",
                // ... (autres balises)
            }
        }
    ],
    "provisioners": [
        {
            "type": "shell",
            "scripts": ["./install_nginx.sh"],
            "execute_command": "{{ .Vars }} sudo -E sh '{{ .Path }}'",
            "pause_before": "10s"
        }
    ]
}
```

Ce fichier JSON est une configuration pour Packer, un outil qui permet de créer des images de machine automatiquement. Voici les détails de chaque section :

Variables

- `aws_access_key` **et** `aws_secret_key` : Ces clés sont utilisées pour authentifier votre compte AWS. (Attention : Ne partagez jamais vos clés d'accès en clair dans votre code ou tout autre support public).

Builders

Cette section décrit comment et où l'image sera construite.

- `type`: Le type de builder est "amazon-ebs", ce qui signifie que Packer créera une AMI (Amazon Machine Image) basée sur EBS (Elastic Block Store).
- `access_key` **et** `secret_key` : Utilisation des clés d'accès AWS définies dans la section des variables.
- `region`: La région AWS où l'AMI sera créée est "us-east-2".
- `source_ami_filter`: Des filtres sont utilisés pour trouver l'AMI source. Ici, il cherche une image Ubuntu 16.04 avec un type de virtualisation HVM et un type de

périphérique racine EBS.

- `instance_type`: Le type d'instance est "t2.micro".
- `ssh_username`: Le nom d'utilisateur SSH pour accéder à l'instance est "ubuntu".
- `ami_name`: Le nom de l'AMI créée contiendra le timestamp pour la rendre unique.
- `tags` est une balise de nom avec `Name` qui a la valeur "Nom_De_Votre_AMI" et une balise `Environment` avec la valeur "Production" seront ajoutées à l'AMI créée.

Provisioners

Cette section spécifie les scripts qui seront exécutés sur la machine.

- `type`: Utilisation d'un shell pour exécuter des commandes.
- `scripts`: Exécution du script `install_nginx.sh` pour installer Nginx.
- `execute_command`: La commande pour exécuter le script.
- `pause_before`: Pause de 10 secondes avant l'exécution du provisioner.

En résumé, ce fichier Packer crée une AMI basée sur Ubuntu 16.04 dans la région "us-east-2" en utilisant une instance "t2.micro". Après la création de l'instance, le script `install_nginx.sh` est exécuté pour installer Nginx sur cette machine.

install_nginx.sh

```
#!/bin/bash
sudo apt-get update
sudo apt-get install -y nginx docker.io
```

Ce fichier est un script shell Bash destiné à être exécuté sur une machine Ubuntu qui correspond à l'instance EC2 créé. Voici ce qu'il fait :

1. `sudo apt-get update`: Met à jour la liste des paquets disponibles à partir des dépôts pour s'assurer que vous disposez des dernières versions. `sudo` est utilisé pour exécuter la commande en tant qu'utilisateur root.
2. `sudo apt-get install -y nginx docker.io`: Installe les paquets `nginx` et `docker.io`. Le `y` permet d'automatiser l'installation en répondant "oui" aux invites.

- `nginx`: C'est un serveur web très populaire.
- `docker.io`: C'est le paquet pour installer Docker, un système de conteneurisation.

Donc, ce script est conçu pour mettre à jour les paquets du système et installer ensuite Nginx et Docker sur une machine Ubuntu.

5- Commandes à exécuter

Ce sont les commandes à exécuter pour générer l'image AMI avec Packer et ensuite créer l'instance EC2 avec Terraform.

Avec Packer

1. **Initialisation** : Ouvrez un terminal et naviguez jusqu'au dossier où se trouve votre fichier `install_custom_ami.json`.
2. **Validation** : Exécutez `packer validate install_custom_ami.json` pour s'assurer que la configuration est valide.
3. **Création de l'AMI** : Exécutez `packer build install_custom_ami.json`. À la fin de cette opération, un ID d'AMI sera affiché, similaire à `ami-00a30bdd2403cd70e`.

Avec Terraform

1. **Initialisation** : Ouvrez un autre terminal et accédez au dossier contenant vos fichiers Terraform (`main.tf`, `variables.tf`, etc.).
2. **Mise à jour de l'AMI_ID** : Vous pouvez soit modifier le fichier Terraform où la variable `AMI_ID` est définie, soit utiliser cette variable comme un argument lors de l'exécution de `terraform apply`.
3. **Initialisation de Terraform** : Exécutez `terraform init` pour initialiser le projet Terraform.
4. **Validation de Terraform** : Exécutez `terraform validate` pour vérifier la validité des fichiers.
5. **Application de Terraform avec Variable** : Utilisez la commande suivante pour appliquer la configuration Terraform en spécifiant l'ID de l'AMI que vous avez créé

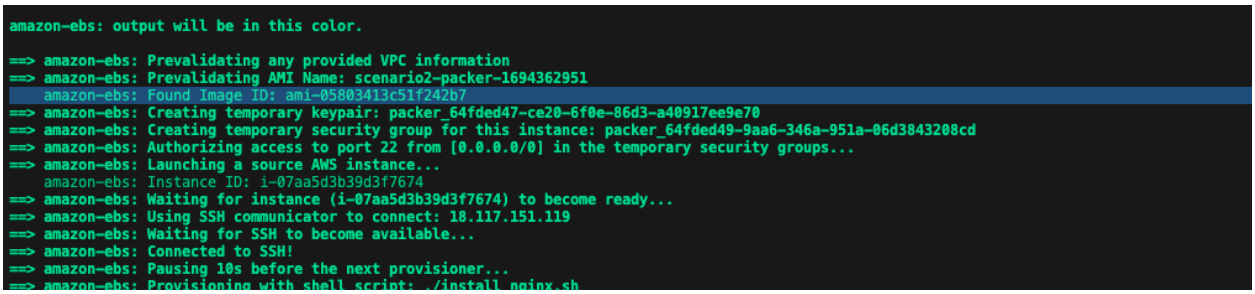
avec Packer :

```
terraform apply -var "AMI_ID=ami-00a30bdd2403cd70e"
```

En suivant ces étapes, vous utiliserez l'AMI généré par Packer pour lancer une nouvelle instance EC2 avec votre code Terraform.

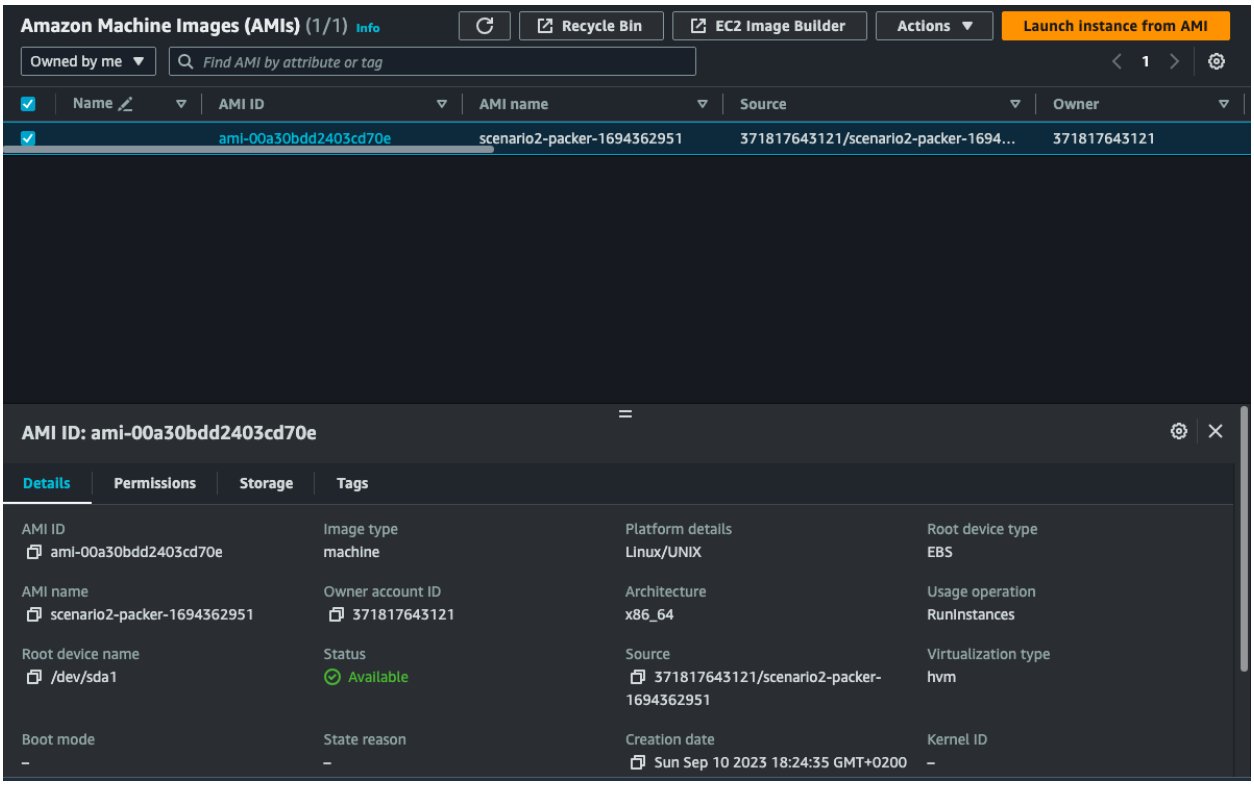
6- Resultat

Avec Packer

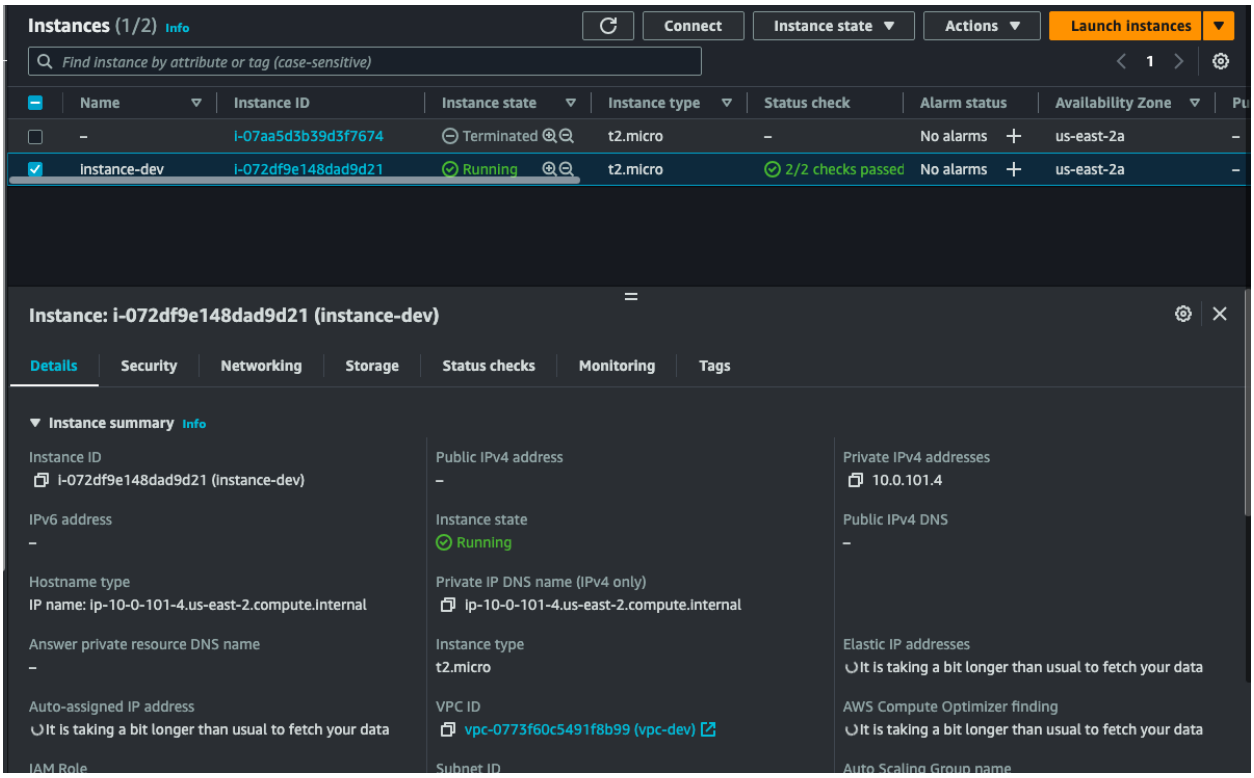


Avec Terraform

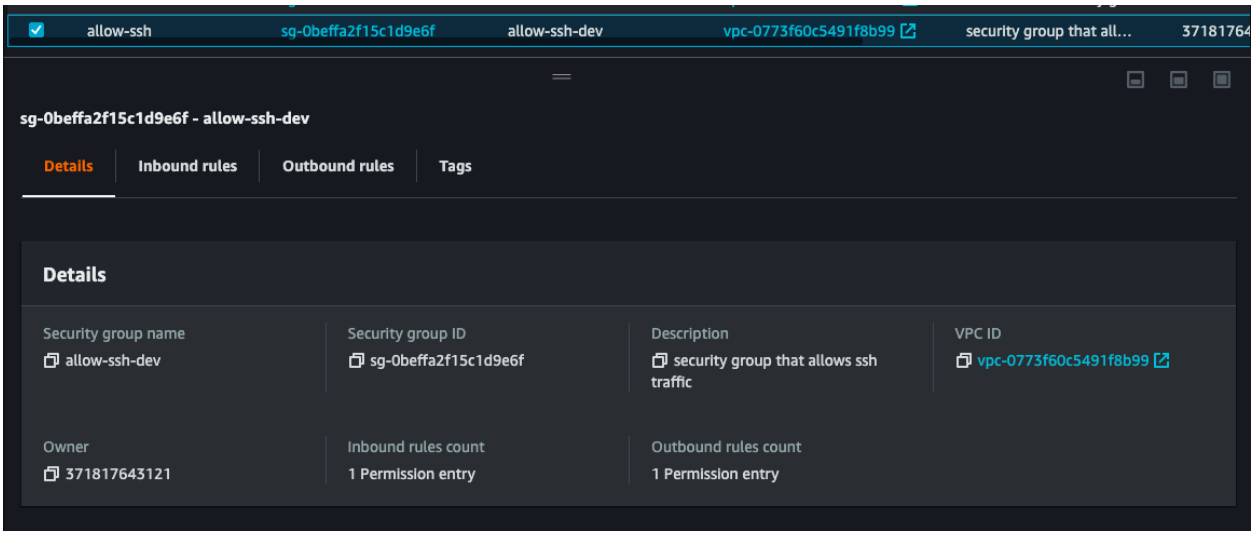
Image AMI



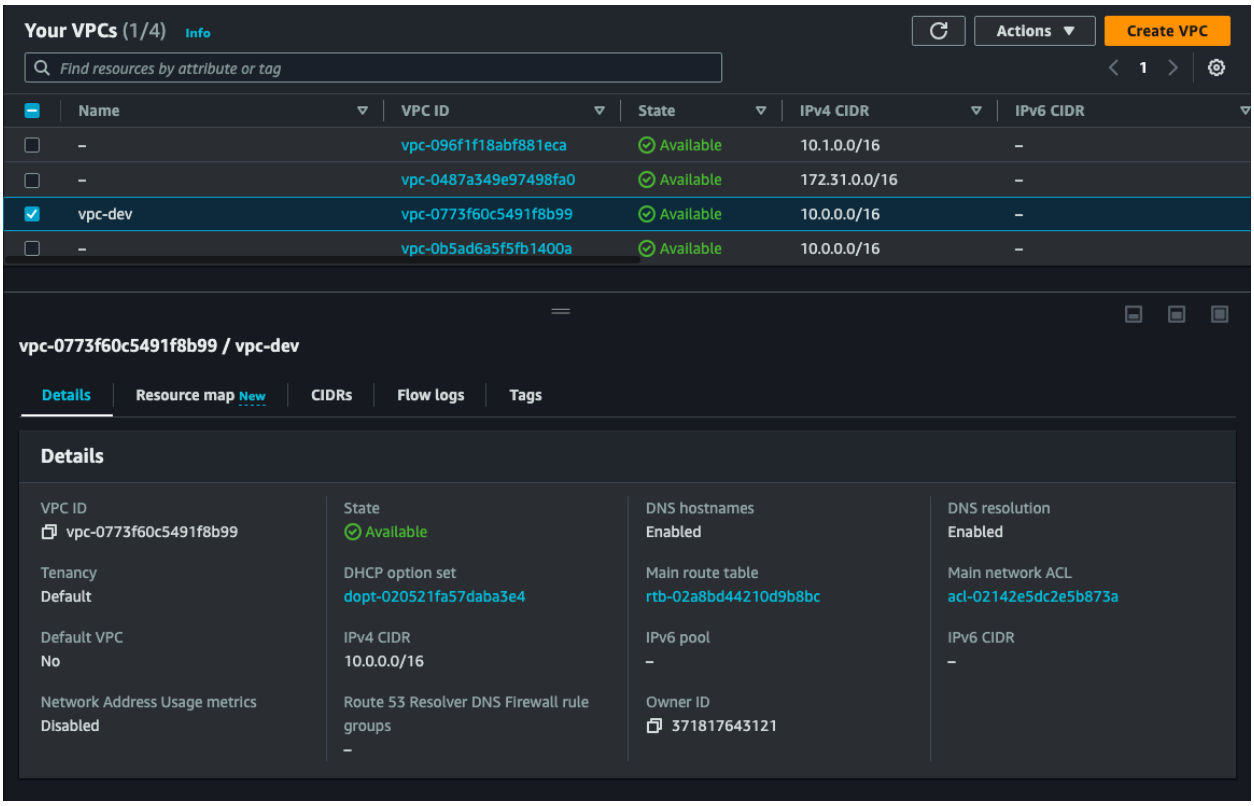
Instance EC2



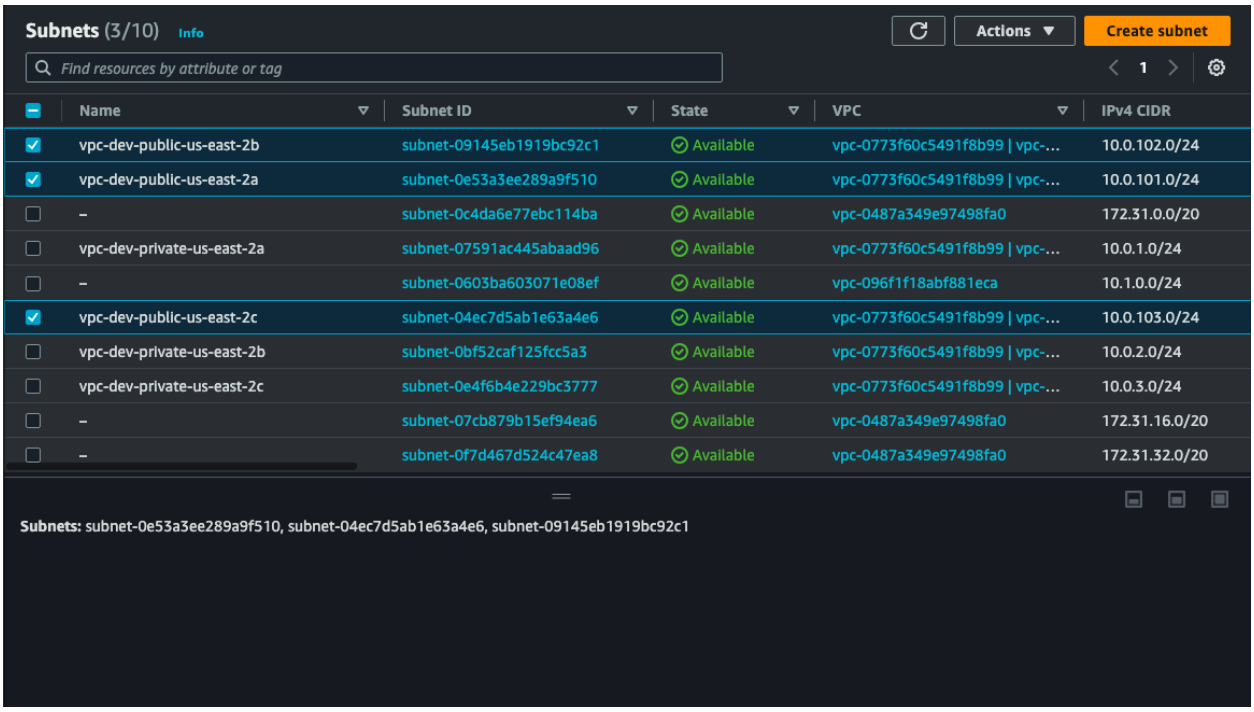
Groupe de sécurité



VPC



Sous-réseaux publics



7- Accès à l’instance EC2 via SSH

```
ssh <ip public de l'instance EC2> -l ubuntu -i levelup_key
```

La commande `ssh <ip public de l'instance EC2> -l ubuntu -i levelup_key` permet de se connecter à une instance EC2 via SSH. Voici la signification des différentes parties de la commande :

- `ssh`: C'est la commande utilisée pour initier une connexion SSH (Secure Shell).
- `<ip public de l'instance EC2>`: C'est l'adresse IP publique de l'instance EC2 à laquelle vous souhaitez vous connecter. Vous devrez remplacer cette partie par l'adresse IP réelle.
- `l ubuntu`: Cette option spécifie que vous voulez vous connecter en tant qu'utilisateur `ubuntu` sur l'instance EC2.
- `i levelup_key`: Cette option indique que vous voulez utiliser la clé SSH située dans le fichier `levelup_key` pour l'authentification.

Assurez-vous que le fichier de clé `levelup_key` est dans le bon répertoire et qu'il a les bonnes permissions (généralement `chmod 600 levelup_key`) avant de lancer cette commande.

N'oubliez pas non plus de vérifier que le groupe de sécurité associé à votre instance EC2 permet les connexions SSH sur le port 22.

Verifier la présence e nginx dans l’instance

```
apt list --installed | grep nginx
```

La commande `apt list --installed | grep nginx` permet de vérifier si le paquet `nginx` est installé sur un système Ubuntu ou Debian. Voici les détails :

- `apt list --installed` : Cette commande liste tous les paquets qui sont actuellement installés sur le système.
- `|` : Le symbole de "pipe" prend la sortie de la commande précédente (`apt list --installed`) et la passe comme entrée à la commande suivante (`grep nginx`).
- `grep nginx` : Cette commande recherche le texte "nginx" dans la liste des paquets installés.

Si `nginx` est installé, cette commande retournera une ou plusieurs lignes contenant le nom du paquet, ce qui indique que le paquet est effectivement installé sur le système.

Verifier aussi docker.io

```
apt list --installed | grep docker
```

Afficher la page d’acceuil de nginx (Étant dans l’instance EC2)

```
curl localhost
```

Extraire le conteneur docker de nginx

```
docker pull nginx:latest
```

Ne pas oublier de supprimer les composants créés

```
terraform destroy
```