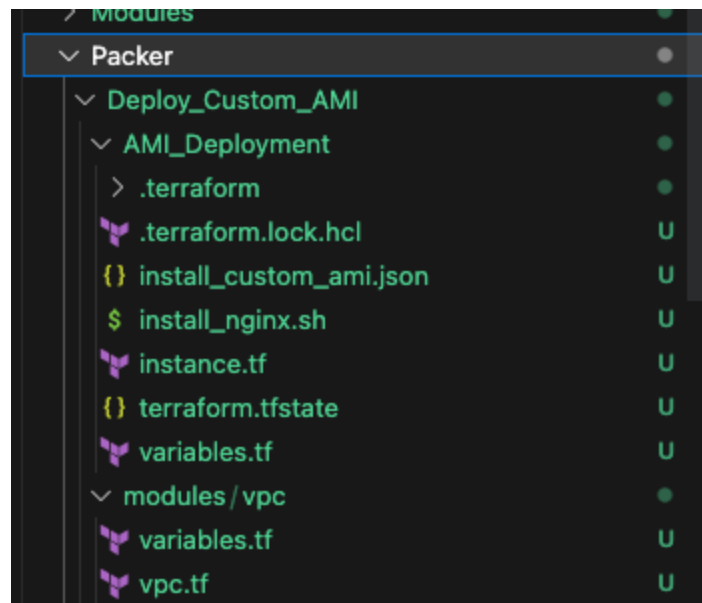


# Terraform + Packer

## 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"
    Environment    = 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 using 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"
    Environment    = 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}"
  Environment = 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": "AKIAVNEQ07RYX7UP4D5R",
    "aws_secret_key": "dqmHdetVqvBtkYOV96EUilf4yuEHi82nj5uiyKdV"
  },
  "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

```
amazon-ebs: output will be in this color.
=> amazon-ebs: Prevalidating any provided VPC information
=> amazon-ebs: Prevalidating AMI Name: scenario2-packer-1694362951
amazon-ebs: Found Image ID: ami-05803413c51f242b7
=> amazon-ebs: Creating temporary keypair: packer_64fdded47-ce20-6f0e-86d3-a40917ee9e70
=> amazon-ebs: Creating temporary security group for this instance: packer_64fdded49-9aa6-346a-951a-06d3843208cd
=> amazon-ebs: Authorizing access to port 22 from [0.0.0.0/0] in the temporary security groups...
=> amazon-ebs: Launching a source AWS instance...
amazon-ebs: Instance ID: i-07aa5d3b39d3f7674
=> amazon-ebs: Waiting for instance (i-07aa5d3b39d3f7674) to become ready...
=> amazon-ebs: Using SSH communicator to connect: 18.117.151.119
=> amazon-ebs: Waiting for SSH to become available...
=> amazon-ebs: Connected to SSH!
=> amazon-ebs: Pausing 10s before the next provisioner...
=> amazon-ebs: Provisioning with shell script: ./install nginx.sh
```

### Avec Terraform

```
module.VPC-01-DEV.module.VPC-01.aws_route_table_association.private[0]: Creating...
module.VPC-01-DEV.module.VPC-01.aws_default_security_group.this[0]: Creation complete after 3s [id=sg-05a3a079d7d936a88]
module.VPC-01-DEV.module.VPC-01.aws_route_table_association.public[2]: Creation complete after 1s [id=rtbassoc-0161cd824adfbfdee]
module.VPC-01-DEV.module.VPC-01.aws_route_table_association.public[0]: Creation complete after 1s [id=rtbassoc-0528d42e255f568f7]
module.VPC-01-DEV.module.VPC-01.aws_route_table_association.public[1]: Creation complete after 1s [id=rtbassoc-0eae55a575d077aa8]
module.VPC-01-DEV.module.VPC-01.aws_route_table_association.private[1]: Creation complete after 1s [id=rtbassoc-05c84376500e19eab]
module.VPC-01-DEV.module.VPC-01.aws_route_table_association.private[2]: Creation complete after 1s [id=rtbassoc-0c9ea4d5cb0c17145]
module.VPC-01-DEV.module.VPC-01.aws_route_table_association.private[0]: Creation complete after 1s [id=rtbassoc-0c659ad0f3ae94cc8]
module.VPC-01-DEV.module.VPC-01.aws_route_public_internet_gateway[0]: Creation complete after 1s [id=r-rtb-0a2927e518d7fd1561080289494]
aws_security_group.SEC-GRP-SSH: Creation complete after 3s [id=sg-0beffa2f15c1d9e6f]
aws_instance.EC2-01: Creating...
aws_instance.EC2-01: Still creating... [10s elapsed]
aws_instance.EC2-01: Still creating... [20s elapsed]
aws_instance.EC2-01: Still creating... [30s elapsed]
aws_instance.EC2-01: Still creating... [40s elapsed]
aws_instance.EC2-01: Creation complete after 44s [id=i-072df9e148dad9d21]

Apply complete! Resources: 25 added, 0 changed, 0 destroyed.
```

## Image AMI

The screenshot shows the Amazon Machine Images (AMIs) console. At the top, there's a header with 'Amazon Machine Images (AMIs) (1/1)' and an 'Info' link. Below this is a search bar with the placeholder 'Find AMI by attribute or tag'. A table lists the AMIs, with columns for Name, AMI ID, AMI name, Source, and Owner. The first AMI is highlighted: 'ami-00a30bdd2403cd70e' with name 'scenario2-packer-1694362951' and source '371817643121/scenario2-packer-1694...'. Below the table, the details for the selected AMI are shown. The details are organized into tabs: Details, Permissions, Storage, and Tags. The 'Details' tab is active, showing a grid of information about the AMI.

Name	AMI ID	AMI name	Source	Owner
scenario2-packer-1694362951	ami-00a30bdd2403cd70e	scenario2-packer-1694362951	371817643121/scenario2-packer-1694...	371817643121

AMI ID: ami-00a30bdd2403cd70e			
Details	Permissions	Storage	Tags
AMI ID ami-00a30bdd2403cd70e	Image type machine	Platform details Linux/UNIX	Root device type EBS
AMI name scenario2-packer-1694362951	Owner account ID 371817643121	Architecture x86_64	Usage operation RunInstances
Root device name /dev/sda1	Status Available	Source 371817643121/scenario2-packer-1694362951	Virtualization type hvm
Boot mode -	State reason -	Creation date Sun Sep 10 2023 18:24:35 GMT+0200	Kernel ID -

## Instance EC2

**Instances (1/2)** [Info](#)

Find Instance by attribute or tag (case-sensitive)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
-	i-07aa5d3b39d3f7674	Terminated	t2.micro	-	No alarms	us-east-2a
Instance-dev	i-072df9e148dad9d21	Running	t2.micro	2/2 checks passed	No alarms	us-east-2a

**Instance: i-072df9e148dad9d21 (instance-dev)**

[Details](#) [Security](#) [Networking](#) [Storage](#) [Status checks](#) [Monitoring](#) [Tags](#)

**Instance summary** [Info](#)

Instance ID i-072df9e148dad9d21 (instance-dev)	Public IPv4 address -	Private IPv4 addresses 10.0.101.4
IPv6 address -	Instance state Running	Public IPv4 DNS -
Hostname type IP name: ip-10-0-101-4.us-east-2.compute.internal	Private IP DNS name (IPv4 only) ip-10-0-101-4.us-east-2.compute.internal	
Answer private resource DNS name -	Instance type t2.micro	Elastic IP addresses It is taking a bit longer than usual to fetch your data
Auto-assigned IP address It is taking a bit longer than usual to fetch your data	VPC ID vpc-0773f60c5491f8b99 (vpc-dev)	AWS Compute Optimizer finding It is taking a bit longer than usual to fetch your data
IAM Role	Subnet ID	Auto Scaling Group name

## Groupe de sécurité

[allow-ssh](#) [sg-0beffa2f15c1d9e6f](#) [allow-ssh-dev](#) [vpc-0773f60c5491f8b99](#) [security group that all...](#) 37181764

**sg-0beffa2f15c1d9e6f - allow-ssh-dev**

[Details](#) [Inbound rules](#) [Outbound rules](#) [Tags](#)

**Details**

Security group name allow-ssh-dev	Security group ID sg-0beffa2f15c1d9e6f	Description security group that allows ssh traffic	VPC ID vpc-0773f60c5491f8b99
Owner 371817643121	Inbound rules count 1 Permission entry	Outbound rules count 1 Permission entry	

## VPC

**Your VPCs (1/4)** [Info](#)

Find resources by attribute or tag

Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR
-	vpc-096f1f18abf881eca	Available	10.1.0.0/16	-
-	vpc-0487a349e97498fa0	Available	172.31.0.0/16	-
<input checked="" type="checkbox"/> vpc-dev	vpc-0773f60c5491f8b99	Available	10.0.0.0/16	-
-	vpc-0b5ad6a5f5fb1400a	Available	10.0.0.0/16	-

**vpc-0773f60c5491f8b99 / vpc-dev**

[Details](#) [Resource map](#) [CIDRs](#) [Flow logs](#) [Tags](#)

**Details**

VPC ID vpc-0773f60c5491f8b99	State Available	DNS hostnames Enabled	DNS resolution Enabled
Tenancy Default	DHCP option set dopt-020521fa57daba3e4	Main route table rtb-02a8bd44210d9b8bc	Main network ACL acl-02142e5dc2e5b873a
Default VPC No	IPv4 CIDR 10.0.0.0/16	IPv6 pool -	IPv6 CIDR -
Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups -	Owner ID 371817643121	

## Sous-réseaux publics

**Subnets (3/10)** [Info](#)

Find resources by attribute or tag

Name	Subnet ID	State	VPC	IPv4 CIDR
<input checked="" type="checkbox"/> vpc-dev-public-us-east-2b	subnet-09145eb1919bc92c1	Available	vpc-0773f60c5491f8b99   vpc-...	10.0.102.0/24
<input checked="" type="checkbox"/> vpc-dev-public-us-east-2a	subnet-0e53a3ee289a9f510	Available	vpc-0773f60c5491f8b99   vpc-...	10.0.101.0/24
-	subnet-0c4da6e77ebc114ba	Available	vpc-0487a349e97498fa0	172.31.0.0/20
vpc-dev-private-us-east-2a	subnet-07591ac445abaad96	Available	vpc-0773f60c5491f8b99   vpc-...	10.0.1.0/24
-	subnet-0603ba603071e08ef	Available	vpc-096f1f18abf881eca	10.1.0.0/24
<input checked="" type="checkbox"/> vpc-dev-public-us-east-2c	subnet-04ec7d5ab1e63a4e6	Available	vpc-0773f60c5491f8b99   vpc-...	10.0.103.0/24
vpc-dev-private-us-east-2b	subnet-0bf52caf125fcc5a3	Available	vpc-0773f60c5491f8b99   vpc-...	10.0.2.0/24
vpc-dev-private-us-east-2c	subnet-0e4f6b4e229bc3777	Available	vpc-0773f60c5491f8b99   vpc-...	10.0.3.0/24
-	subnet-07cb879b15ef94ea6	Available	vpc-0487a349e97498fa0	172.31.16.0/20
-	subnet-0f7d467d524c47ea8	Available	vpc-0487a349e97498fa0	172.31.32.0/20

**Subnets:** subnet-0e53a3ee289a9f510, subnet-04ec7d5ab1e63a4e6, subnet-09145eb1919bc92c1



## 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'accueil 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
```

