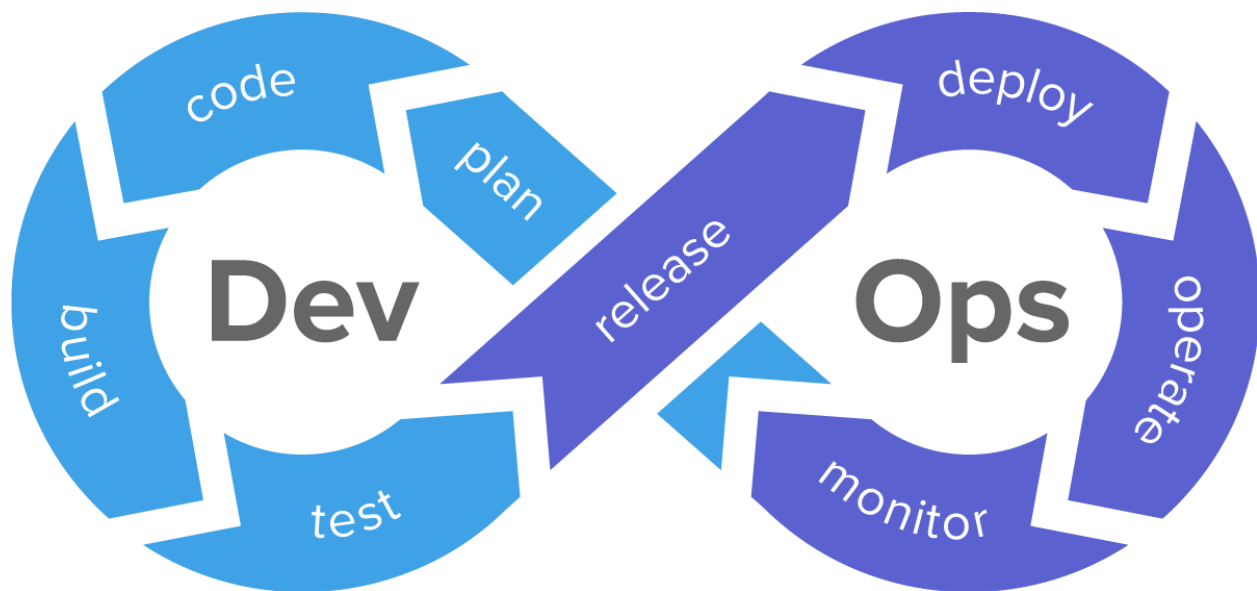


RAPPORT GLOBAL

DevOps



Salsabil DAFRANE
Mahshid HAZHEER
Armand HURULT
Ouriel MAMAN
Guillaume BRUN
Moubarak FADILI
Zineb CHAUCHE
Emilia MEDJEBEUR

SOMMAIRE

| | |
|---|-----------|
| Introduction | 3 |
| TP1 – Premiers pas sur AW | 4 |
| Mise en place de l'infrastructure..... | 5 |
| Les questions..... | 7 |
| TP-2 Construction automatisée d'une AMI avec Ansible et Packer | 9 |
| Ansible..... | 9 |
| Packer..... | 15 |
| TP3 Déployer des ressources sur AWS avec Terraform | 21 |

Introduction

Ce rapport consiste à démontrer les connaissances et compétences acquises lors des travaux pratiques réalisés dans le cadre de la découverte et l'apprentissage de la philosophie DevOps. Lors de ces cours nous avons acquis des compétences variées en travaillant sur le déploiement d'une infrastructure Web, la configuration de réseaux, le déploiement d'applications ainsi que l'automatisation de ces tâches.

Nous avons également découvert de nombreux outils variés tels que : **A**mazon **W**eb **S**ervices (AWS), Ansible, Terraform ou Packer.

TP1 – Premiers pas sur AWS

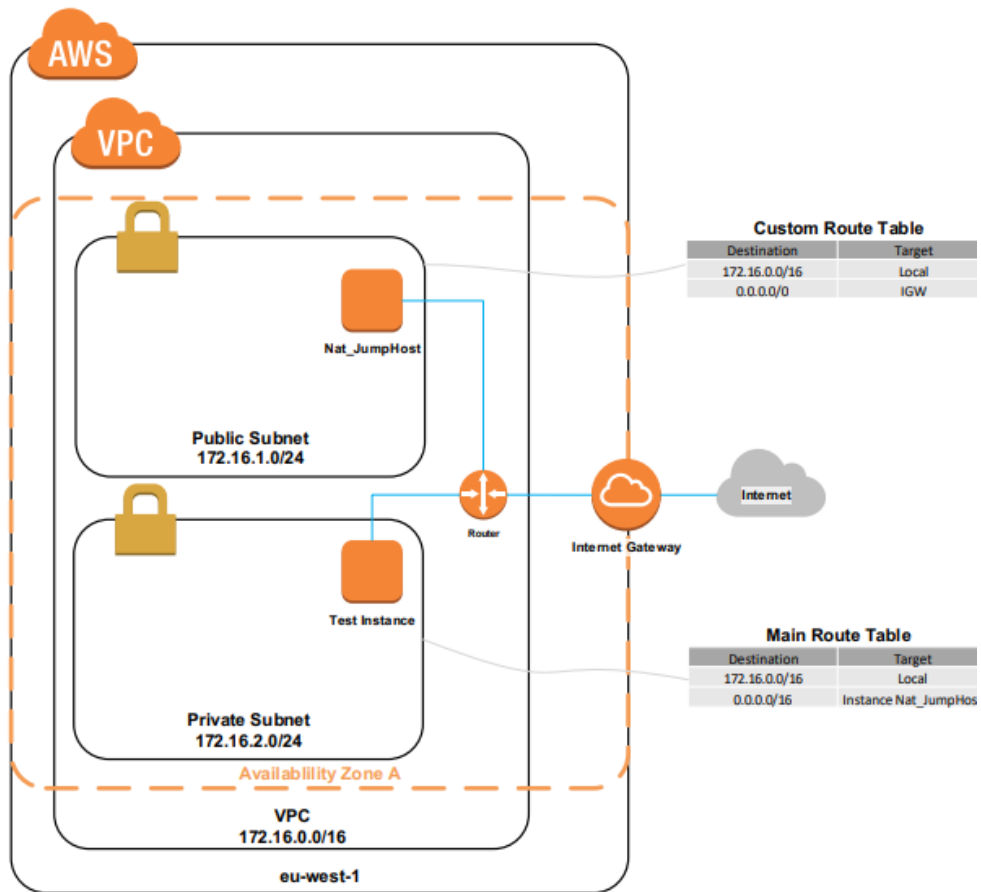
Amazon Web Services, Inc. (AWS) est une filiale d'Amazon qui fournit des plates-formes et des API d'informatique en Cloud à la demande à des particuliers, des entreprises et des gouvernements, sur la base d'un paiement au compteur.

Ces services web d'informatique en cloud fournissent une capacité de traitement informatique et des outils. L'un de ces services est Amazon Elastic Compute Cloud (EC2), qui permet aux utilisateurs d'avoir à leur disposition un cluster virtuel d'ordinateurs, disponible en permanence, via Internet. Les ordinateurs virtuels d'AWS émulent la plupart des attributs d'un ordinateur réel, notamment les unités centrales de traitement (CPU) et les unités de traitement graphique pour le traitement, la mémoire RAM ou local, le stockage sur disque en HDD ou SSD, un choix de systèmes d'exploitation, la mise en réseau et les logiciels d'application préchargés tels que les serveurs web, les bases de données et la gestion de la relation client.

De plus, les services en cloud proposés par AWS sont très populaires et utilisés du fait de la sécurité des machines et du fait des nombreux paramètres possibles. Par exemple, la double authentification (MFA) qui peut être mise en place via de nombreuses applications (DuoMobile, Authy, etc.) est une composante très importante.

Après avoir créé le compte AWS, nous avons effectué des paramétrages de sécurité et mis en place des alertes de paiement. Nous n'allons pas plus détailler cette partie.

Mise en place de l'infrastructure



Afin de mettre en place cette infrastructure, dans un premier temps, il faut configurer l'architecture réseau du serveur Web. Tel qu'illustré dans le schéma précédent, nous avons sur l'infrastructure un sous-réseau public et un sous-réseau privé communiquant via un routeur rattaché à Internet. Sur AWS, ces derniers sont représentés par un Cloud Virtuel Privé (VPC). Cela permet de lancer des VM nommées instances EC2 et mettre en place les sous-réseaux.

| | | | |
|---------------------------------|--|---|---|
| VPC ID vpc-0c16007ab974dfa18 | State Available | DNS hostnames Enabled | DNS resolution Enabled |
| Tenancy Default | DHCP options set dopt-07513dd28dcd2fed9 | Main route table rtb-00d5416f14a75e0bc / dev-rt-main-private | Main network ACL acl-0e28597045e5f4f26 |
| Default VPC No | IPv4 CIDR 10.0.0.0/16 | IPv6 pool - | IPv6 CIDR - |

Afin de faire communiquer les instances présentes dans le sous-réseau avec Internet, un routage est réalisé via l'interface d'AWS.

| Instance: i-0ae0204f9c6d67813 (ToolsClCD) | | |
|---|--|--|
| ▼ Instance summary Info | | |
| Instance ID i-0ae0204f9c6d67813 (ToolsClCD) | Public IPv4 address 34.245.201.119 open address | Private IPv4 addresses 172.31.37.146 |
| IPv6 address - | Instance state Running | Public IPv4 DNS ec2-34-245-201-119.eu-west-1.compute.amazonaws.com open address |
| Hostname type IP name: ip-172-31-37-146.eu-west-1.compute.internal | Private IP DNS name (IPv4 only) ip-172-31-37-146.eu-west-1.compute.internal | |

L'infrastructure mise en place est composée d'un sous-réseau public, un sous-réseau privé et d'une Gateway NAT. Le sous-réseau privé communique seulement avec le sous-réseau public. Tandis que, le sous-réseau public communique avec le sous-réseau privé et Internet. Il est aussi capable de se connecter au réseau privé via SSH. L'instance nommée **NAT_JumpHost** permet au sous-réseau de communiquer avec Internet.

Cette configuration de processus NAT (Network Address Translation) est très importante car cela offre une protection contre les communications malveillantes externes du réseau en modifiant les IP et masquant les plages d'adresses privées qui sont ensuite mappées avec les adresses publiques.

Cette configuration se trouve dans le **groupe de sécurité**. Dans lequel on peut définir des règles de sécurité tout comme un pare-feu afin de les appliquer sur des instances. Chaque groupe de sécurité peut être appliqué à plusieurs instances tant que c'est le même VPC.

Ci-dessous nous avons les règles autorisant les échanges entre toutes les machines en SSH et HTTPS.

EC2 > Security Groups > sg-05c2116736d5f052d - launch-wizard-2 > Edit inbound rules

Edit inbound rules [info](#)

Inbound rules control the incoming traffic that's allowed to reach the instance.

| Security group rule ID | Type | Protocol | Port range | Source | Description - optional | |
|------------------------|-------|----------|------------|--------|------------------------|--------|
| sg-0238acea7634ee367 | SSH | TCP | 22 | Custom | | Delete |
| sg-02dbf00a11d55f0ce | HTTPS | TCP | 443 | Custom | | Delete |

[Add rule](#)

[Cancel](#) [Preview changes](#) [Save rules](#)

Après avoir configuré le réseau, on lance les machines pour tester la connexion entre eux et Internet.

Pour ce faire, nous nous connectons via SSH et on ping Internet 8.8.8.8.

Cela fonctionne ✓

Les questions

→ *Comment contrôler et limiter l'accès à une instance ?*

Pour limiter et contrôler l'accès à une instance, nous avons la passerelle (Gateway NAT) qui protège les sous-réseaux contre les échanges externes malveillants et le service IAM qui contrôle les authentifications et les autorisations d'utiliser les ressources.

→ *Quel élément réseau est nécessaire pour que les instances d'un réseau privé puissent communiquer vers Internet pour récupérer les packages et updates ?*

NAT Gateway permet au sous-réseau privé d'échanger avec Internet pour récupérer les packages et les mises à jour.

→ *Sur AWS quelles sont les propriétés qui caractérisent un « Public Subnet » ?*

Un échange libre avec Internet est la priorité d'un public sur AWS. Il est disponible à l'extérieur et est capable de transmettre des données vers l'extérieur.

→ *Les « Security Groups » sont-ils stateless ou statefull ?*

Les "Security Groups" sont Statefull, puisqu'ils conservent la trace des connexions antérieures. Le statut des connexions est gardé et les connexions sont utilisables plusieurs fois.

→ *Qu'est-ce qu'une « Default Route » Table ?*

La table « Default Route » correspond au cheminement par défaut paramétré sur la table de route que les trafics prendront dans le cas où aucune autre voie n'inclut la destination du trafic.

→ *Comment faire pour me connecter à une instance dans un sous-réseau privé ?*

Afin de pouvoir se connecter à une instance dans un sous-réseau privé, vous devez autoriser une connexion SSH vers cette instance à partir d'une instance située dans le sous-réseau public.

Il suffit de configurer les groupes de sécurité pour qu'ils acceptent l'accès SSH et de fournir la clé appropriée.

TP2 – Construction automatisée d'une AMI avec Ansible et Packer

Ansible

On commence par déployer 2 instances avec les caractéristiques suivantes :

| Name Tag | Public Subnet / Public IP | Security Group Rules |
|--------------|---------------------------|---|
| ToolsCICD | Yes | SSH – from your Public IP |
| RemoteServer | Yes | SSH – from your Public IP SSH – from ToolsCICD Private IP |

On pense bien à modifier les règles sécurité de groupe pour autoriser la connexion vers celles-ci en SSH et pour autoriser également ToolsCICD à communiquer avec RemoteServer :

The screenshot displays the AWS Security Groups configuration interface. Two security group rules are defined:

- Règle de groupe de sécurité 1:** Type: ssh, Protocole: TCP, Plage de ports: 22, Type de source: Mon IP (185.181.155.201/32).
- Règle de groupe de sécurité 2:** Type: ssh, Protocole: TCP, Plage de ports: 22, Type de source: Personnalisé(e) (172.16.1.81/32).

The summary on the right indicates 1 instance, using the Canonical Ubuntu 16.04 LTS image (ami-0f29c8402f8cce65c), t2.micro instance type, and 1 volume (8 GiB).

Nos 2 machines sont créées et belles et bien prêtes à l'emploi. On y accède par SSH via le Terminal MobaXterm :

```
ubuntu@ip-172-16-1-95:~$  ubuntu@ip-172-16-1-81:~$
```

On commence par installer Ansible sur **ToolsCICD** :

```
ubuntu@ip-172-16-1-81:~$ ansible --version
ansible 2.9.27
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/ubuntu/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.12 (default, Mar 1 2021, 11:38:31) [GCC 5.4.0 20160609]
ubuntu@ip-172-16-1-81:~$
```

Mais avant tout, expliquons ce qu'est Ansible.

Ansible est un outil d'automatisation informatique. Il permet de configurer des systèmes, de déployer des logiciels et d'orchestrer des tâches informatiques plus avancées telles que des déploiements continus ou des mises à jour permanentes sans temps d'arrêt.

Les principaux objectifs d'Ansible sont la simplicité et la facilité d'utilisation. Il met également l'accent sur la sécurité et la fiabilité.

C'est un outil extrêmement pratique puisqu'il permet de déployer des architectures en un rien de temps, surtout lorsque ce sont des éléments identiques, à travers ce qu'on appelle des playbooks. Un playbook est un fichier **.yaml** composé d'une suite d'instructions.

Le but final ici va être de créer un playbook dont la finalité est le déploiement d'un site Internet via le paquet Apache sur la machine **RemoteServer**. Mais découvrons tout d'abord des fonctionnalités plus basiques d'Ansible.

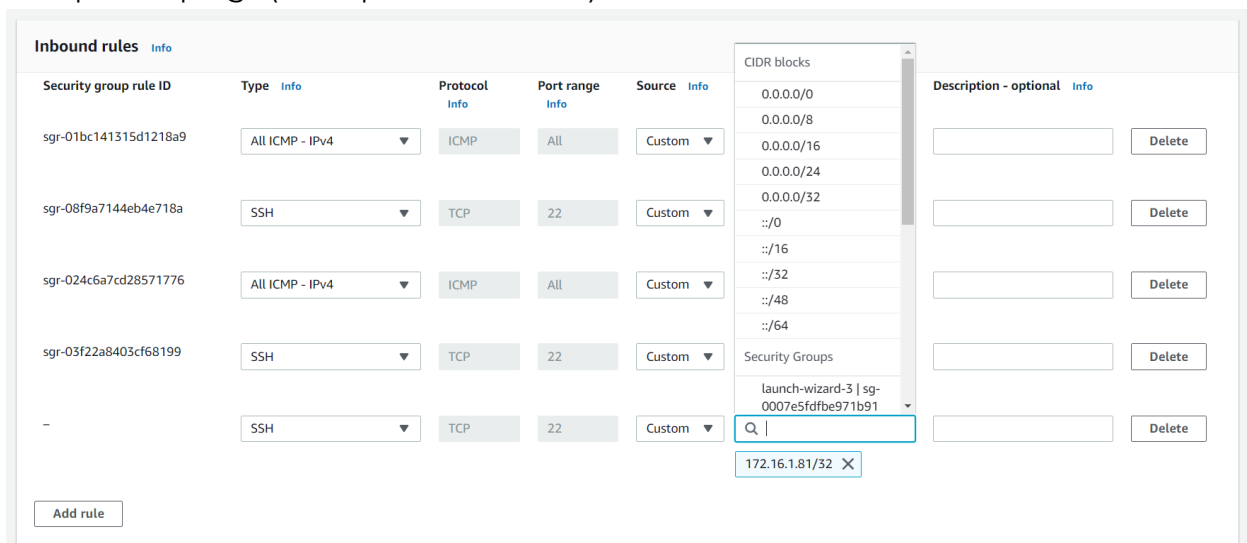
Tout d'abord, nous copions les clés de **RemoteServer** sur **ToolsCICD** pour accéder à celui-ci :

```
ubuntu@ip-172-16-1-81:~$ nano ~/.ssh/Keys.pem
ubuntu@ip-172-16-1-81:~$
ubuntu@ip-172-16-1-81:~$
ubuntu@ip-172-16-1-81:~$ cat ~/.ssh/Keys.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAiUzEI5r4rCLtsD0H9ZYNa9uqR6lRZU7/8EbkLgJWAwMsG3
Wq+spR0cm4YXhDeB iI00h4+0nkR/kuE4Yn0CVpgeFZUAzxYkLZr4f438InBGk+Tk
dNjC8Pu7GJA3yLDAnpEeD9wdekGRrEfPjj10U5HPisxQ1lvpZKn+tNyxToEkWjq
kiMzRM/b9IN7EL6ZZrWetl4wXyZDxAec/CJYkml.88Cx18iD7Z49tUNoj2woqm7P
Si/hYASwd7GSIPeyGrMLJSVj0otrPToDLWPcFS00JV0+stq2G5zIfh5IyCvWt1Mx
wp6jHteJ6PcuAqmTeGj9LAi27vppENChBPEhpQIDAQABAoIBAHS8yn+jgg+PYe3Q
wqayoFK2gH9RnR9aXttdQ1ZJUyLHqWUG5A19GUN5+0vYpSILt50E+v2TuVh30Shn
Sf0bVpSf/g1H1Vp97wYo12RMqrX1VGzmzr/yLT0kk53k8Sk3725i4HaHpkggha+RZ
Yjh14UqKhdTKEIy8/EqA+UYwhoLdZCIBxVFmNhnpY7Rj84bCCuBhZ5byZHifYYVp
o7E/ar0KrnRr2Q1+PGwHh5KnFcZxVWL5d5Vzy90mT7BvSyFkARVWeobNwujv+N
Q0mJDwTFoPeFFV1HGp0vKMPgrgK5e350zQS0j0kd1GZ0CoD8R+cR6L750mF0Fv06
sKpnZ/kCgYEAx1Br50NM+Twz0FVr805bzBm7TR8S1nlgveQoL6ZWdRXzxwNVd7gv
x0futvrZxb5gZnQWhSfyQvDQDLyLILQXwVG3Q2b0XaqZKNTQp25h0Wxzvumzhv i4
uSmdABC6P4AsvMFImPshQ+6D03aqY4y3PLVib958uetY2Vs/VqWbj08CgYEAxFk8
XMxc2XLVmyV5Lmxyaj9hL/K4bR8zZL/MwWoI0FB07rNqrEeDnCR/Hn/g/QQDBTf
82vLv5T0S8LSP4vVzepSLUU2WfotFeFI5LSy2LQ01n7nUr1Rqu9AxVQFK58M5shD
WQ9kNuhzHKYHbo/mhLm9DHj fRo5dnHv+ekRR4qsCgYAGH5DxY5gK9gJtCc9edkCV
oCYtsZ3pXtKXhfj3Dqj4bv6sf2R2ZpdSCRMd9Xva6ZC46oauWA+st7zpHbJ7rsq0
b5vkXmLWldpygcH5SrZNIIdaMs fJ3S4YCs/0583ltKpC1mZ7FIcTdaXL8JZGap66
I2sRv00zHg2sGTwq94PwCwKBgQC1DfxZqQUxJ00AtM+LC001Nzpj iI ixAWQX+FU
oyF+s8TF1x0bjYw8cJzSiSo fQ0G65XZ6LYnQz+DCpUPy3XQD5ANDocBbvHEVQX50
Dpg+myUTmLnTABIA6XJcMjUDBFTbLsdUEF6R5YxzZLBgtg2IxpJP/tjJ0brBV3buQ
TobGEwKBgQCtrXj iGnAP3LU2tAHRQRXCWZ i9qtJrHxjtHzxmW7PvImNti4s2gW1/
9bLbJ4587Iwr/53XKQE/TZamynrD3L+dChKXp0wmlB4w1EoFppkxQhUY4GQCmQC
CDF6cdHFGt4c0Tr7LR9kn0z6XpQfIaPIgA/xtwMsnjVjo9GIEApNMg==
-----END RSA PRIVATE KEY-----
ubuntu@ip-172-16-1-81:~$
```

On commence à mettre en place notre répertoire de travail :

```
ubuntu@ip-172-16-1-81:~$ ls
TP_CICD
ubuntu@ip-172-16-1-81:~$ cd TP_CICD/
ubuntu@ip-172-16-1-81:~/TP_CICD$ cd WebAMIProject/
ubuntu@ip-172-16-1-81:~/TP_CICD/WebAMIProject$
ubuntu@ip-172-16-1-81:~/TP_CICD/WebAMIProject$ pwd
/home/ubuntu/TP_CICD/WebAMIProject
ubuntu@ip-172-16-1-81:~/TP_CICD/WebAMIProject$
```

On va commencer par tenter de pinger **RemoteServer** depuis **ToolsCICD** via Ansible. Mais pour autoriser cela, il faut configurer notre machine **RemoteServer** pour qu'elle accepte les pings (via le protocole ICMP) :



Une fois cela en place, notre machine **RemoteServer** reçoit les pings de **ToolsCICD**, et ce via la commande *ping* mise à disposition par Ansible :

```
ubuntu@ip-172-16-1-81:~/TP_CICD/WebAMIProject$ ansible all -m ping -i ubuntu@172.16.1.95, --private-key ~/.ssh/Keys.pem
The authenticity of host '172.16.1.95 (172.16.1.95)' can't be established.
ECDSA key fingerprint is SHA256:bb0nIoBnx0aIuBRFvW8qjAgce99ZTwUSiSyhE4aieqI.
Are you sure you want to continue connecting (yes/no)? yes
ubuntu@172.16.1.95 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
ubuntu@ip-172-16-1-81:~/TP_CICD/WebAMIProject$
```

BINGO, notre communication fonctionne ✔

On va maintenant tester cela mais via un fichier playbook, dans le but de se rapprocher de notre objectif final :

```
GNU nano 2.5.3 File: play.yml
--
- hosts: all
  tasks:
    - name: test connection
      ping:

ubuntu@ip-172-16-1-81:~/TP_CICD/WebAMIProject$ ansible-playbook -i ubuntu@172.16.1.95, --private-key ~/.ssh/Keys.pem play.yml
PLAY [all] *****
TASK [Gathering Facts] *****
ok: [ubuntu@172.16.1.95]
TASK [test connection] *****
ok: [ubuntu@172.16.1.95]
PLAY RECAP *****
ubuntu@172.16.1.95 : ok=2  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
ubuntu@ip-172-16-1-81:~/TP_CICD/WebAMIProject$
```

Le ping fonctionne à nouveau ! ✓

Désormais, place à notre playbook dont le but sera de :

1. Installer Apache dans sa dernière version
2. Changer le port d'écoute d'Apache et du Virtualhost sur le port 8080
3. Supprimer le default website d'Apache (/var/www/html)
4. Déployer un website « <https://github.com/cloudacademy/static-website-example> »
5. Redémarrer le service Apache
6. Nous tenons à préciser que celui-ci a été fait nous-mêmes, afin que la solution soit donnée par le professeur.

Voici le contenu de notre Playbook :

```
---
- hosts: all
  become: yes
  vars:
    document_root: /var/www/html
    git_url: https://github.com/cloudacademy/static-website-example.git
    http_port: 8080
  tasks:
    - name: install apache2
      apt: name=apache2 update_cache=yes state=latest

    - name: install git
      apt: name=git update_cache=yes state=latest

    - name: enabled mod_rewrite
      apache2_module: name=rewrite state=present

    - name: apache2 listen on port 8080
      lineinfile: dest=/etc/apache2/ports.conf regexp="^Listen 80" line="Listen 8080" state=present

    - name: apache2 virtualhost on port 8080
      lineinfile: dest=/etc/apache2/sites-available/000-default.conf regexp="^<VirtualHost \*:80>" line="<VirtualHost
*:8080>" state=present

    - name: Ansible delete file wildcard example
      shell: rm -rf /var/www/html/*

    - name: Git Clone
      git:
        repo: "{{ git_url }}"
        dest: "{{ document_root }}"
        clone: yes
        update: yes
        force: yes
        notify: Reload Apache

    #- name: Disable default Apache site
    #  shell: /usr/sbin/a2dissite 000-default.conf
    #  when: disable_default
    #  notify: Reload Apache

    - name: "UFW - Allow HTTP on port {{ http_port }}"
      ufw:
        rule: allow
        port: "{{ http_port }}"
        proto: tcp

  handlers:
    - name: Reload Apache
      service:
        name: apache2
        state: reloaded

    - name: Restart Apache
      service:
        name: apache2
        state: restarted
```

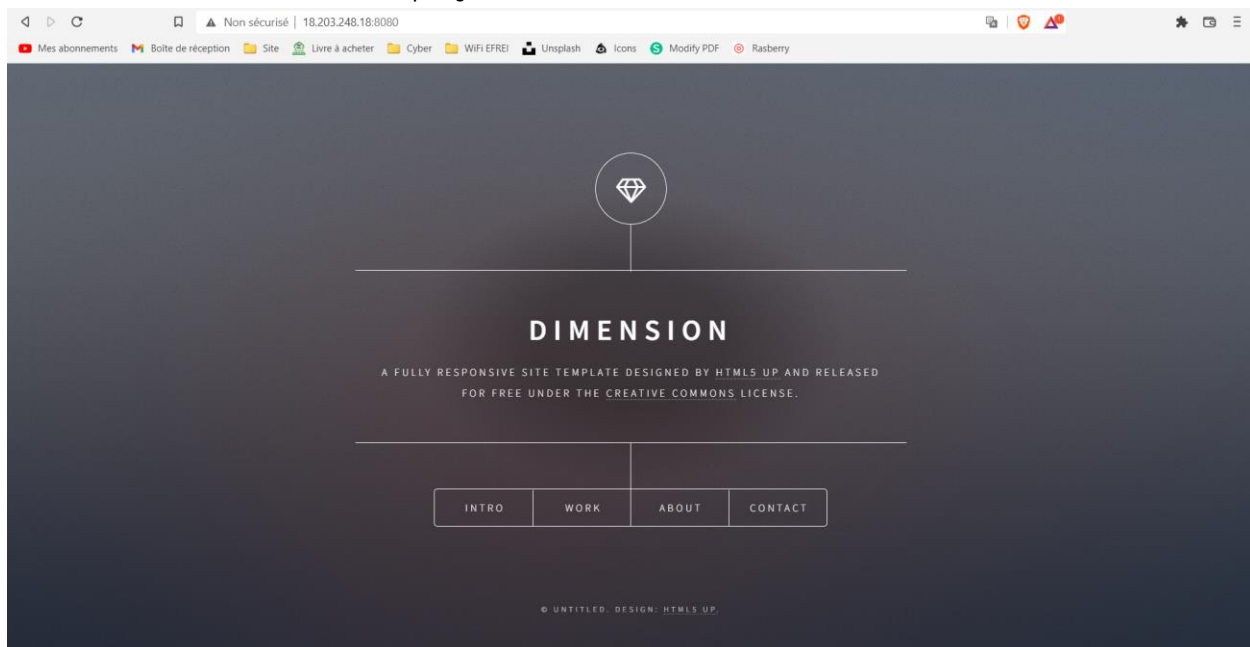
Expliquons son fonctionnement :

1. On commence par définir quelques variables : l'emplacement du contenu du site Web, dans *document_root*, l'URL du repo donc on va récupérer les fichiers HTML pour notre site avec *git_url*, et le port d'exposition du site avec *http_port*.
2. On commence par installer le paquet apache2 ainsi que l'utilitaire git qui nous servira à gitclone.
3. On active un module et on change son port d'écoute pour 8080.
4. On supprime les fichiers de base d'Apache pour le front.
5. On clone le repo GitHub pour récupérer le nouveau contenu de notre site.
6. On reload Apache via son handler prédéfini **Reload Apache**.
7. On lance notre Playbook contenu dans **play.yml** via la commande suivante

```
ubuntu@ip-172-16-1-81:~/TP_CICD/WebAMIPProject$ ansible-playbook --private-key ~/.ssh/Keys.pem play.yml
PLAY [apache] *****
TASK [Gathering Facts] *****
ok: [172.16.1.95]
TASK [install apache2] *****
changed: [172.16.1.95]
PLAY RECAP *****
172.16.1.95 : ok=2  changed=1  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

BINGO ✓

Notre site est bel et bien déployé :



A travers cet exemple d'utilisation d'Ansible, nous avons pu voir à quel point il est simple et rapide de déployer un site Internet sur une nouvelle machine via un repo GitHub.

On pourrait s'imaginer déployer de nombreux sites différents sur diverses machines, en changeant simplement le contenu de la variable *git_url*, ce qui apporte une très grande flexibilité.

Packer

Nous allons désormais voir le déploiement d'AMI Amazon avec Packer. Comme expliqué dans l'énoncé, Packer est un outil open source qui permet de créer des images sur de multiples plateformes par l'intermédiaire de différents **Builders** qui s'adressent à de multiples providers et de **Provisionners** qui permettent de préparer l'image.

Le provider utilisé ici sera donc **AWS**, et le Provisionner **Ansible**.

On commence par créer notre machine **ToolsDevOps** :

The screenshot shows the AWS Management Console interface for creating a new AMI. The top section, 'AMI from catalog', displays a search for 'ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-20190913' with the AMI ID 'ami-03ef731cc103c9f09'. A table lists the AMI's details: Catalog (Community AMIs), Published (2019-09-13T17:29:13.000Z), Architecture (x86_64), Virtualization (hvm), Root device type (ebs), and ENA Enabled (Yes). Below this, the 'Instance type' section shows 't2.micro' selected, which is 'Free tier eligible'. The 'Key pair (login)' section shows 'TP2_CICD' selected as the key pair name, with a 'Create new key pair' button.

| Catalog | Published | Architecture | Virtualization | Root device type | ENA Enabled |
|----------------|--------------------------|--------------|----------------|------------------|-------------|
| Community AMIs | 2019-09-13T17:29:13.000Z | x86_64 | hvm | ebs | Yes |

Instance type

Instance type: **t2.micro** (Free tier eligible)

Family: t2 | 1 vCPU | 1 GiB Memory
On-Demand Linux pricing: 0.0126 USD per Hour
On-Demand Windows pricing: 0.0172 USD per Hour

Key pair (login)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required: **TP2_CICD** [Create new key pair]

No preference [Create new subnet](#)

Auto-assign public IP [Info](#)
 Enable

Firewall (security groups) [Info](#)
 A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☒ Create security group ☐ Select existing security group

Security group name - *required*
 Security_Group_ToolsDevOps_TP2

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and _-./!@#%^&*~`|}{[]+~&{}!\$*

Description - *required* [Info](#)
 launch-wizard-1 created 2022-06-17T10:06:43.031Z

Inbound security groups rules

▼ Security group rule 1 (TCP, 22, 83.114.31.35/32) [Remove](#)

| Type | Protocol | Port range |
|------|----------|------------|
| ssh | TCP | 22 |

Source type [Info](#)
 My IP

Source [Info](#)
 Add CIDR, prefix list or security group
 83.114.31.35/32

Description - *optional* [Info](#)
 e.g. SSH for admin desktop

[Add security group rule](#)

On installe ensuite Packer sur celle-ci :

```
ubuntu@ip-172-31-21-202:~$ packer -v
1.5.1
```

Pour permettre à Packer d'interagir avec les API AWS, il faut lui fournir un accès API. Contrairement à d'autres API qui fonctionnent par clé, ici c'est avec un rôle que l'on va attacher à l'instance. Créons-le :

VPC EC2 IAM

EC2 > Instances > i-0d4518c55bb9712b4 > Modify IAM role

Modify IAM role [Info](#)
 Attach an IAM role to your instance.

Instance ID
 i-0d4518c55bb9712b4 (ToolsDevOps)

IAM role
 Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

EC2RoleForPacker [Create new IAM role](#)

[Cancel](#) [Update IAM role](#)

Instances (1/2) info

| Name | Instance ID | Instance state | Instance type | Status check | Alarm status | Availability Zone | Public IPv4 DNS | Public IPv4 ... | Elastic IP | IPv6 IPs | Me |
|-------------|----------------------|----------------|---------------|--------------|--------------|-------------------|--------------------------|-----------------|------------|----------|----|
| ToolsDevOps | i-0d4518c555b09712b4 | Stopped | t2.micro | - | No alarms | eu-west-1a | - | - | - | - | di |
| ToolsCICD | i-09ab7e2aa6a6be90b | Running | t2.micro | Initializing | No alarms | eu-west-1b | ec2-52-211-88-142.eu-... | 52.211.88.142 | - | - | di |

Instance: i-09ab7e2aa6a6be90b (ToolsCICD)

Details Security Networking Storage Status checks Monitoring Tags

▼ Instance summary info

Instance ID
i-09ab7e2aa6a6be90b (ToolsCICD)

Public IPv4 address
52.211.88.142 | [open address](#)

Instance state
Running

Private IP DNS name (IPv4 only)
ip-172-31-44-25.eu-west-1.compute.internal

Instance type
t2.micro

VPC ID
vpc-05899240d79a8e4c6

Subnet ID
subnet-0c07f5ae8915bf68d

Private IPv4 addresses
172.31.44.25

Public IPv4 DNS
ec2-52-211-88-142.eu-west-1.compute.amazonaws.com | [open address](#)

Elastic IP addresses
-

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | [Learn more](#)

Auto Scaling Group name
-

Auto-assigned IP address
52.211.88.142 [Public IP]

IAM Role
EC2RoleForPacker

▼ Instance details info

Platform
-

AMI ID
-

Monitoring
-

On crée ensuite le fichier **buildAMI.json** en veillant à bien modifier nos variables :

```

{
  "variables": {
    "region": "eu-west-1",
    "ssh_username": "ubuntu",
    "base_ami": "ami-03ef731cc103c9f09",
    "instance_type": "t2.micro",
    "subnet_id": "subnet-0c07f5ae8915bf68d",
    "temporary_security_group_source_cidrs": "52.211.88.142/32",
    "associate_public_ip_address": "true"
  },
  "builders": [
    {
      "type": "amazon-ec2",
      "region": "{{user `region`}}",
      "subnet_id": "{{user `subnet_id`}}",
      "source_ami": "{{user `base_ami`}}",
      "instance_type": "{{user `instance_type`}}",
      "ssh_username": "{{user `ssh_username`}}",
      "ami_name": "AMI-Apache-{{ isotime '\\01022006-150405\\' }}",
      "temporary_security_group_source_cidrs": "{{user `temporary_security_group_source_cidrs`}}",
      "associate_public_ip_address": "{{user `associate_public_ip_address`}}",
      "tags": {
        "Name": "PackerAnsible-Apache"
      }
    }
  ],
  "provisioners": [
    {
      "type": "ansible",
      "playbook_file": "./play.yml"
    }
  ]
}

```

Enfin, on exécute ce fichier **buildAMI.json** :

```
ubuntu@ip-172-31-44-25:~/TF_Cloud/terraform$ terraform apply packer build buildAMI.json
amazon-ecs: output will be in this color:

=> amazon-ecs: Prevalidating any provided VPC information
=> amazon-ecs: Prevalidating AMI Name: ami-06132022-155317
=> amazon-ecs: Found Image ID: ami-06132022-155317
=> amazon-ecs: Creating temporary keypair: packer_02af400f-0131-0131-0131-0131
=> amazon-ecs: Creating temporary security group for this instance: packer_02af400f-0131-0131-0131-0131
=> amazon-ecs: Authorizing access to port 22 from [02.233.55.142/32] in the temporary security group...
=> amazon-ecs: Launching a source AWS instance...
=> amazon-ecs: Adding tags to source instance
=> amazon-ecs: Adding tag: "Name": "Packer Builder"
=> amazon-ecs: Instance ID: i-090206497f1e9d01
=> amazon-ecs: Waiting for instance (i-090206497f1e9d01) to become ready...
=> amazon-ecs: Using ssh communicator to connect: 34.243.181.29
=> amazon-ecs: Waiting for SSH to become available...
=> amazon-ecs: Connected to SSH!
=> amazon-ecs: Provisioning with Ansible...
=> amazon-ecs: Executing Ansible: =====
amazon-ecs:
amazon-ecs: PLAY [all] =====
amazon-ecs:
amazon-ecs: TASK [Gathering Facts] =====
amazon-ecs: ok: [default]
amazon-ecs:
amazon-ecs: TASK [Install Python3] =====
amazon-ecs: ok: [default]
amazon-ecs:
amazon-ecs: TASK [Install Git package] =====
amazon-ecs: changed: [default]
amazon-ecs:
amazon-ecs: TASK [ensure apache is at the latest version] =====
amazon-ecs: changed: [default]
amazon-ecs:
amazon-ecs: TASK [enable mod_rewrite] =====
amazon-ecs: changed: [default]
amazon-ecs:
amazon-ecs: TASK [apache2 listen on port 443] =====
amazon-ecs: changed: [default]
amazon-ecs:
amazon-ecs: TASK [apache2 virtualhost on port 443] =====
amazon-ecs: changed: [default]
amazon-ecs:
amazon-ecs: TASK [remove default website directory] =====
amazon-ecs: changed: [default]
amazon-ecs:
amazon-ecs: TASK [git checkout website] =====
amazon-ecs: changed: [default]
amazon-ecs:
amazon-ecs: TASK [restart apache2] =====
amazon-ecs: changed: [default]
amazon-ecs:
amazon-ecs: PLAY RECAP =====
amazon-ecs: default                : ok=18  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
=> amazon-ecs: Stopping the source instance...
=> amazon-ecs: Waiting for the instance to stop...
=> amazon-ecs: Creating AMI: ami-06132022-155317 from instance i-090206497f1e9d01
=> amazon-ecs: AMI: ami-06132022-155317
=> amazon-ecs: Waiting for AMI to become ready...
=> amazon-ecs: Adding tags to AMI (ami-06132022-155317)...
=> amazon-ecs: Tagging snapshot: snap-012f1f0667a3d8359
=> amazon-ecs: Creating AMI tags
=> amazon-ecs: Adding tag: "Name": "PackerAnsible-Apache"
=> amazon-ecs: Creating snapshot tags
=> amazon-ecs: Terminating the source AWS instance...
=> amazon-ecs: Cleaning up any extra volumes...
=> amazon-ecs: No volumes to clean up, skipping
=> amazon-ecs: Deleting temporary security group...
=> amazon-ecs: Deleting temporary keypair...
Build 'amazon-ecs' finished.

=> Builds finished. The artifacts of successful builds are:
=> amazon-ecs: AMIs were created:
  eu-west-1: ami-06132022-155317
```

Cela a donc pour effet de créer une instance à partir de l'AMI :

The screenshot shows the AWS Management Console interface. On the left, there's a navigation menu with options like VPC, EC2, IAM, Instance types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Scheduled Instances, and Capacity Reservations. The main area displays the 'Amazon Machine Images (AMIs) (1/1) info' page. A table lists the AMIs, with 'PackerAnsible-Apache' (ami-0a931ad687c290541) selected. The right pane shows the details for this AMI, including its name, ID, type (machine), architecture (x86_64), source (444433145928/AMI-Apache-06192022-155317), creation date (Sun Jun 19 2022 17:55:07 GMT+0200 (heure d'été d'Europe centrale)), and various tags.

| Name | AMI ID | AMI name | Source | Owner | Visibility | Status | Creation date |
|----------------------|-----------------------|----------------------------|---|--------------|------------|-----------|---------------------|
| PackerAnsible-Apache | ami-0a931ad687c290541 | AMI-Apache-06192022-155317 | 444433145928/AMI-Apache-06192022-155317 | 444433145928 | Private | Available | 2022/06/19 17:55:07 |

| AMI ID: ami-0a931ad687c290541 (PackerAnsible-Apache) | | | |
|--|--------------------------------|--|-------------------------------|
| Details | Permissions | Storage | Tags |
| AMI ID: ami-0a931ad687c290541 (PackerAnsible-Apache) | Image type: machine | Platform details: Linux/UNIX | Root device type: EBS |
| AMI name: AMI-Apache-06192022-155317 | Owner account ID: 444433145928 | Architecture: x86_64 | Usage operation: RunInstances |
| Root device name: /dev/sda1 | Status: Available | Source: 444433145928/AMI-Apache-06192022-155317 | Virtualization type: hvm |
| Boot mode: - | State reason: - | Creation date: Sun Jun 19 2022 17:55:07 GMT+0200 (heure d'été d'Europe centrale) | Kernel ID: - |
| Block devices: /dev/sda1=snap-012f1f0667a3d8359:truegpg2, /dev/sdb=ephemeral0, /dev/sdc=ephemeral1 | Description: - | Product codes: - | RAM disk ID: - |
| Deprecation time: - | Last launched time: - | | |

On va alors la lancer :

The screenshot shows the Amazon Management Console interface. At the top, the 'Amazon Machine Images (AMIs) (1/1) Info' section is active. A search bar shows 'Owned by me'. Below, a table lists AMIs. The AMI 'ami-0a931ad687c290541' (PackerAnsible-...) is selected, and a context menu is open with the option 'Launch instance from AMI' highlighted. Below the table, the details for the selected AMI are shown, including its ID, name, root device name, and boot mode. The 'Instances (1/4) Info' section is also visible, showing a table of existing instances. The instance 'ToolsCICD - Copy' is selected, and its details are shown below, including its ID, state (Running), and public IP address (34.248.94.8).

Amazon Machine Images (AMIs) (1/1) Info

Owned by me

| Name | AMI ID | AMI name | Source |
|-------------------|-----------------------|----------------------------|------------------------|
| PackerAnsible-... | ami-0a931ad687c290541 | AMI-Apache-06192022-155317 | 444433145928/AMI-Ap... |

AMI ID: ami-0a931ad687c290541 (PackerAnsible-...)

Details | Permissions | Storage | Tags

AMI ID: ami-0a931ad687c290541 (PackerAnsible-Apache)

AMI name: AMI-Apache-06192022-155317

Root device name: /dev/sda1

Boot mode: Available

Owner account ID: 444433145928

Status: Available

State reason:

Instances (1/4) Info

| Name | Instance ID | Instance state | Instance type | Status check | Alarm status | Availability Zone | Public IPv4 DNS | Public IPv4 ... | El... |
|------------------|---------------------|----------------|---------------|-------------------|--------------|-------------------|--------------------------|-----------------|-------|
| ToolsDevOps | i-0d4518c55bb9712b4 | Terminated | t2.micro | - | No alarms | eu-west-1a | - | - | - |
| ToolsCICD | i-09ab7e2aa6a6be90b | Running | t2.micro | 2/2 checks passed | No alarms | eu-west-1b | ec2-52-211-88-142.eu-... | 52.211.88.142 | - |
| Packer Builder | i-09620d497fe19a961 | Terminated | t2.micro | - | No alarms | eu-west-1b | - | - | - |
| ToolsCICD - Copy | i-099e252b432588f3c | Running | t2.micro | - | No alarms | eu-west-1b | ec2-34-248-94-8.eu-we... | 34.248.94.8 | - |

Instance: i-099e252b432588f3c (ToolsCICD - Copy)

Details | Security | Networking | Storage | Status checks | Monitoring | Tags

Instance summary Info

Instance ID: i-099e252b432588f3c (ToolsCICD - Copy)

Public IPv4 address: 34.248.94.8 | open address

Private IPv4 addresses: 172.31.34.97

Instance state: Running

Public IPv4 DNS: ec2-34-248-94-8.eu-west-1.compute.amazonaws.com | open address

Private IP DNS name (IPv4 only): ip-172-31-34-97.eu-west-1.compute.internal

On vérifie qu'on y a bien accès en SSH :

```
PS C:\Users\Armand> ssh -i "C:\Users\Armand\OneDrive\Documents\Pas important\Téléchargement
s\Cours Efrei\Cours\Année 2\DevOps\TP2\TP2_CICD.pem" ubuntu@34.248.94.8
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-1092-aws x86_64)

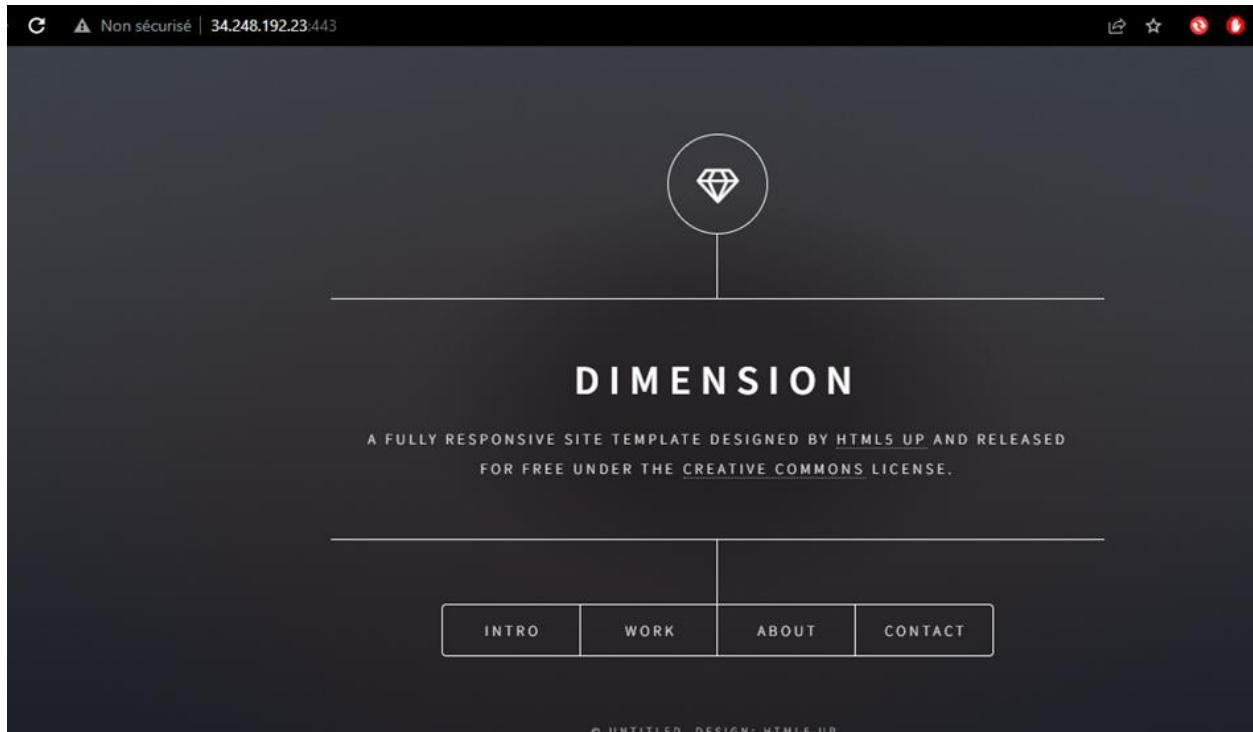
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

160 packages can be updated.
119 updates are security updates.

New release '18.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Jun 19 16:05:08 2022 from 83.114.31.35
ubuntu@ip-172-31-34-97:~$
```

Et on tente enfin d'accéder à notre site Internet déployé grâce au même Playbook que dans la partie précédente :



BINGO ✓

Ainsi, Packer permet d'automatiser le déploiement des instances. Couplé à Ansible, on commence à entrevoir les possibilités qu'il offre : un simple script qui déploierait à lui seule une machine et la configurerait.

TP3 – Déployer des ressources sur AWS avec Terraform

Introduction

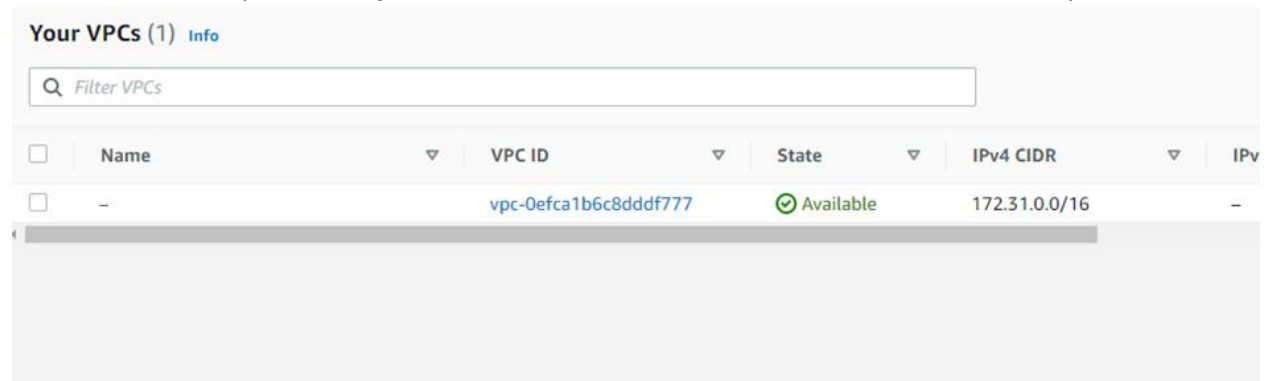
Dans cette partie nous utiliserons Terraform afin de déployer de manière automatisée l'application w

Web à l'aide de l'AMI applicative préconstruite dans la partie précédente.

Terraform est un outil, dit d'**Infrastructure-as-a-Code** qui permet de construire et gérer une infrastructure de manière automatisée.

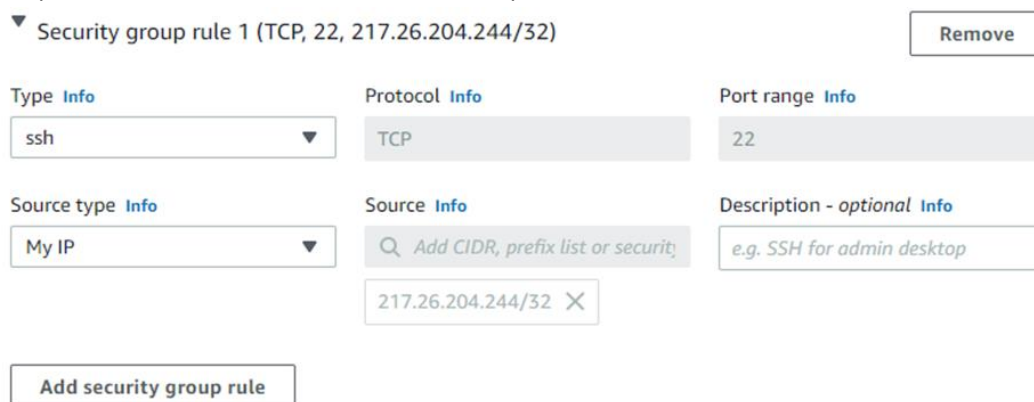
Nous allons reprendre la création d'une AMI à l'aide d'Ansible et Packer, réalisée précédemment. Ici nous allons détailler la prise en main de Terraform qui va nous permettre de créer infrastructures, règles et machines de notre système.

On commence par nettoyer toutes nos VPC en laissant seulement celle par défaut :



Puis, on déploie une instance **ToolsCICD** dans le « Default VPC ».

On pense bien à autoriser le SSH depuis notre IP :





On crée également un rôle EC2 avec les accès Administrateurs :

▼ Advanced details [Info](#)

Purchasing option [Info](#)
☐ Request Spot Instances
Request Spot Instances at the Spot price, capped at the On-Demand price

IAM instance profile [Info](#)

EC2RoleForPacker
arn:aws:iam::234285159957:instance-profile/EC2RoleForPacker

 [Create new IAI](#) 

Hostname type [Info](#)

Ensuite, on installe Terraform, Ansible et Packer pour préparer le terrain :

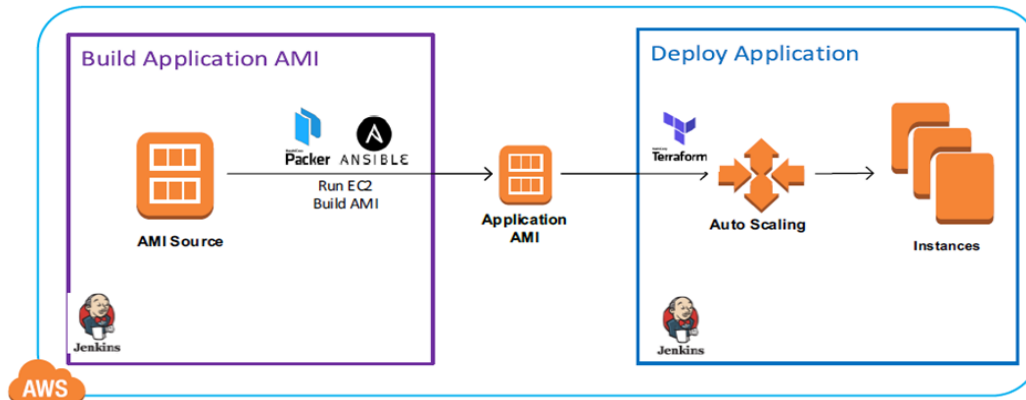
```
ubuntu@ip-172-31-47-171:~$ terraform -v
Terraform v0.12.20

Your version of Terraform is out of date! The latest version
is 1.2.3. You can update by downloading from https://www.terraform.io/downloads.html
ubuntu@ip-172-31-47-171:~$ ansible --version
ansible [core 2.12.7]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/ubuntu/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.10.4 (main, Apr 2 2022, 09:04:19) [GCC 11.2.0]
  jinja version = 3.0.3
  libyaml = True
ubuntu@ip-172-31-47-171:~$ packer --version
1.5.1
ubuntu@ip-172-31-47-171:~$
```

Maintenant que notre environnement est prêt à l'emploi, voici les étapes que nous allons suivre :

1. On build l'AMI
2. On déploie l'infrastructure avec le fichier infra.tf
3. On déploie les instances dans l'infrastructure avec le fichier website.tf

Le but est d'obtenir cela :



Build de l'AMI

On commence par build l'AMI dans le fichier **buildAMI.json**, en modifiant bien l'ID du subnet et l'adresse IP dans la partie *variables*:

```

GNU nano 6.2 buildAMI.json *
{
  "variables": {
    "region": "eu-west-1",
    "ssh_username": "ubuntu",
    "base_ami": "ami-03ef731cc103c9f09",
    "instance_type": "t2.micro",
    "subnet_id": "subnet-0748c5b2e6e2866bc",
    "temporary_security_group_source_cidrs": "34.242.93.25/32",
    "associate_public_ip_address": "true"
  },
  "builders": [
    {
      "type": "amazon-ec2",
      "region": "{{user `region`}}",
      "subnet_id": "{{user `subnet_id`}}",
      "source_ami": "{{user `base_ami`}}",
      "instance_type": "{{user `instance_type`}}",
      "ssh_username": "{{user `ssh_username`}}",
      "ami_name": "AMI-Apache-{{ isotime '\\01022006-150405\\' }}",
      "temporary_security_group_source_cidrs": "{{user `temporary_security_group_source_cidrs`}}",
      "associate_public_ip_address": "{{user `associate_public_ip_address`}}",
      "tags": {
        "Name": "PackerAnsible-Apache"
      }
    }
  ],
  "provisioners": [
    {
      "type": "ansible",
      "playbook_file": "./play.yml"
    }
  ]
}

```

Puis on crée le playbook dans **play.yml** :

```
GNU nano 6.2 play.yml *
--
- hosts: all
  tasks:
    - name: Install Python3
      become: yes
      apt: name=python3 update_cache=yes state=latest

    - name: Install Git package
      become: yes
      apt: name=git update_cache=yes state=latest

    - name: ensure apache is at the latest version
      become: yes
      apt: name=apache2 update_cache=yes state=latest

    - name: enabled mod_rewrite
      become: yes
      apache2_module: name=rewrite state=present

    - name: apache2 listen on port 443
      become: yes
      lineinfile: dest=/etc/apache2/ports.conf regexp="^Listen 80" line="Listen 443" state=present

    - name: apache2 virtualhost on port 443
      become: yes
      lineinfile: dest=/etc/apache2/sites-available/000-default.conf regexp="^<VirtualHost *:80>" line="<VirtualHost *:443>" state=present

    - name: remove default website directory
      become: yes
      file:
        path: /var/www/html
        state: absent

    - name: Git checkout website
      become: yes
```

On lance ensuite le build à l'aide de Packer :

```
ubuntu@ip-172-31-36-73:~/TP_CICD$ packer build buildAMI.json
amazon-ebss: output will be in this color.

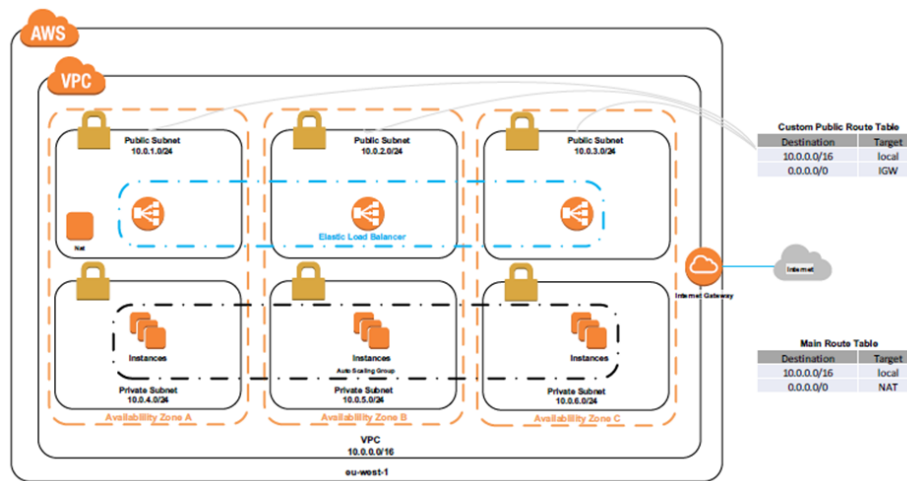
==> amazon-ebss: Prevalidating any provided VPC information
==> amazon-ebss: Prevalidating AMI Name: AMI-Apache-06242022-142708
amazon-ebss: Found Image ID: ami-03ef731cc103c9f09
==> amazon-ebss: Creating temporary keypair: packer_62b5c9bc-6929-d34c-f5ab-5263552e11d7
==> amazon-ebss: Creating temporary security group for this instance: packer_62b5c9bc-c36f-3852-4c9f-3713aee021d9
==> amazon-ebss: Authorizing access to port 22 from [54.154.111.99/32] in the temporary security groups...
==> amazon-ebss: Launching a source AWS instance...
==> amazon-ebss: Adding tags to source instance
amazon-ebss: Adding tag: "Name": "Packer Builder"
amazon-ebss: Instance ID: i-01274c686562a90f6
==> amazon-ebss: Waiting for instance (i-01274c686562a90f6) to become ready...
==> amazon-ebss: Using ssh communicator to connect: 3.250.124.11
==> amazon-ebss: Waiting for SSH to become available...
==> amazon-ebss: Connected to SSH!
==> amazon-ebss: Provisioning with Ansible...
==> amazon-ebss: Executing Ansible: *****i*****l*****p*****l*****g*****b*****o*****k*****_*****
*****v*****g*****r*****s*****p*****g*****c*****k*****g*****r*****b*****u*****i*****l*****d*****a*****n*****
*****m*****e*****g*****m*****g*****z*****o*****n*****-*****e*****b*****s*****p*****g*****c*****k*****e*****r*****
*****l*****d*****e*****p*****e*****=*****a*****m*****z*****o*****n*****-*****e*****b*****s*****p*****g*****c*****k*****e*****r*****
*****d*****e*****i*****e*****s*****o*****i*****y*****=*****y*****g*****e*****-*****e*****i*****e*****-*****e*****p*****
*****/*****p*****g*****k*****e*****r*****c*****p*****r*****o*****v*****i*****c*****e*****i*****o*****g*****e*****r*****-*****g*****n*****e*****c*****e*****l*****b*****
*****/*****e*****5*****0*****1*****7*****2*****5*****5*****7*****1***** *****/*****h*****o*****g*****e*****g*****/*****u*****b*****g*****n*****t*****u*****/*****
*****/*****p***** *****c*****i*****c*****d*****/*****p*****l*****g*****y*****.*****y*****e*****g*****l*****e***** *****e***** *****g*****n*****e*****c*****e*****b*****
*****/*****e***** *****s*****s*****h***** *****p*****r*****i*****v*****g*****t*****e***** *****k*****g*****y***** *****f*****i*****l*****e*****-*****/*****
*****/*****m*****p*****/*****g*****n*****s*****i*****b*****l*****e*****-*****k*****g*****y*****s*****5*****3*****6*****2*****4*****5*****2*****
amazon-ebss:
amazon-ebss: PLAY [all] *****
amazon-ebss:
amazon-ebss: TASK [Gathering Facts] *****
amazon-ebss: ok: [default]
amazon-ebss:
amazon-ebss: TASK [Install Python3] *****
amazon-ebss: ok: [default]
amazon-ebss:
amazon-ebss: TASK [Install Git package] *****
amazon-ebss: changed: [default]
amazon-ebss:

==> Builds finished. The artifacts of successful builds are:
--> amazon-ebss: AMIs were created:
eu-west-1: ami-03cac726ed51ed583
```

Notre AMI apparaît bien dans notre panel **Instances** d'Amazon AWS :

| <input type="checkbox"/> | Name ▾ | AMI ID ▾ | AMI name ▾ | Source ▾ | Owner ▾ | Visibility |
|--------------------------|-------------------|-----------------------|----------------------------|-------------------------------------|--------------|------------|
| <input type="checkbox"/> | PackerAnsible-... | ami-03cac726ed51ed583 | AMI-Apache-06242022-142708 | 234285159957/AMI-Apache-06242022... | 234285159957 | Private |

Déploiement de l'infrastructure



Tout d'abord, il est important de noter que nous n'utilisons pas le NAT car les machines n'ont pas besoin de faire de requêtes vers Internet.

On commence par initialiser du fichier Terraform et du provider AWS (région Irlande) :

```
1 provider "aws" {
2   region = "eu-west-1"
3 }
4
5 variable "env" {
6   type    = string
7   default = "dev"
8 }
9
```

Puis on crée le VPC et la gateway Internet reliée au VPC :

```
10 # VPC
11 resource "aws_vpc" "vpc" {
12   cidr_block     = "10.0.0.0/16"
13   enable_dns_support = "true"
14   enable_dns_hostnames = "true"
15   enable_classiclink = "false"
16   instance_tenancy = "default"
17
18   tags = {
19     Name = "${var.env}-vpc"
20   }
21 }
22
23 # IGW
24 resource "aws_internet_gateway" "igw" {
25   vpc_id = aws_vpc.vpc.id
26
27   tags = {
28     Name = "${var.env}-igw"
29   }
30 }
```

Puis on crée les 3 subnets public avec zone de disponibilité spécifique :

```
32 # Subnets
33 ## Public
34 ### AZ1
35 resource "aws_subnet" "subnet-public-1" {
36     vpc_id            = aws_vpc.vpc.id
37     cidr_block        = "10.0.1.0/24"
38     map_public_ip_on_launch = "true"
39     availability_zone  = "eu-west-1a"
40     tags = {
41         Name = "${var.env}-subnet-public-1"
42     }
43 }
44
45 ### AZ2
46 resource "aws_subnet" "subnet-public-2" {
47     vpc_id            = aws_vpc.vpc.id
48     cidr_block        = "10.0.2.0/24"
49     map_public_ip_on_launch = "true"
50     availability_zone  = "eu-west-1b"
51     tags = {
52         Name = "${var.env}-subnet-public-2"
53     }
54 }
55
56 ### AZ3
57 resource "aws_subnet" "subnet-public-3" {
58     vpc_id            = aws_vpc.vpc.id
59     cidr_block        = "10.0.3.0/24"
60     map_public_ip_on_launch = "true"
61     availability_zone  = "eu-west-1c"
62     tags = {
63         Name = "${var.env}-subnet-public-3"
64     }
65 }
66
```

Ensuite on crée les 3 subnets privées avec zones de disponibilités spécifiques :

```
67  ## Private
68  ### AZ1
69  resource "aws_subnet" "subnet-private-1" {
70      vpc_id            = aws_vpc.vpc.id
71      cidr_block        = "10.0.4.0/24"
72      map_public_ip_on_launch = "false"
73      availability_zone  = "eu-west-1a"
74      tags = {
75          Name = "${var.env}-subnet-private-1"
76      }
77  }
78
79  ### AZ2
80  resource "aws_subnet" "subnet-private-2" {
81      vpc_id            = aws_vpc.vpc.id
82      cidr_block        = "10.0.5.0/24"
83      map_public_ip_on_launch = "false"
84      availability_zone  = "eu-west-1b"
85      tags = {
86          Name = "${var.env}-subnet-private-2"
87      }
88  }
89
90  ### AZ3
91  resource "aws_subnet" "subnet-private-3" {
92      vpc_id            = aws_vpc.vpc.id
93      cidr_block        = "10.0.6.0/24"
94      map_public_ip_on_launch = "false"
95      availability_zone  = "eu-west-1c"
96      tags = {
97          Name = "${var.env}-subnet-private-3"
98      }
99  }
```

Puis on crée le Security Group et les règles de l'instance NAT. On permet ainsi aux machines des sous-réseaux Private-1/2/3 d'accéder à tous les ports et pour tous les protocoles sur l'instance NAT.

Nous n'allons pas l'utiliser ici car les machines n'ont pas besoin de faire de requêtes sur Internet, donc on commente la partie NAT :

```
121
122 # Nat SG
123 √ resource "aws_security_group" "allow_nat" {
124     name      = "allow_web"
125     vpc_id    = aws_vpc.vpc.id
126     description = "Allow inbound traffic"
127 }
128
129 ## SG Rule egress
130 √ resource "aws_security_group_rule" "web_egress_allow_all" {
131     type          = "egress"
132     to_port       = 0
133     protocol      = "-1"
134     from_port     = 0
135     cidr_blocks   = ["0.0.0.0/0"]
136     security_group_id = aws_security_group.allow_nat.id
137 }
138
139 ## SG Rule ingress
140 √ resource "aws_security_group_rule" "ingress_allow_private" {
141     type          = "ingress"
142     from_port     = 0
143     to_port       = 0
144     protocol      = -1
145     cidr_blocks   = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]
146     security_group_id = aws_security_group.allow_nat.id
147 }
148
149 */
```

Puis on crée l'instance NAT à l'aide d'une AMI Ubuntu du catalogue d'AMI :

```
101  /*
102  # Nat Instance
103  resource "aws_instance" "nat" {
104      ami                = "ami-0f630a3f40b1eb0b8"
105      instance_type      = "t2.micro"
106      subnet_id          = aws_subnet.subnet-public-1.id
107      vpc_security_group_ids = [aws_security_group.allow_nat.id]
108      source_dest_check   = "false"
109
110      user_data = <<-EOF
111      |         | #!/bin/bash
112      |         | sysctl -w net.ipv4.ip_forward=1 /sbin/
113      |         | iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
114      |         | EOF
115
116      tags = {
117          Name = "${var.env}-NatInstance"
118      }
119  }
120
```

Puis on crée la default route table utilisée dans le VPC pour rediriger les paquets vers le NAT :

```
151  # Route Table
152  ## Private
153  ### Use Main Route Table
154  resource "aws_default_route_table" "main-private" {
155      default_route_table_id = aws_vpc.vpc.default_route_table_id
156  }
157  /*
158  route {
159      cidr_block = "0.0.0.0/0"
160      instance_id = aws_instance.nat.id
161  }
162  */
163  tags = {
164      Name = "${var.env}-rt-main-private"
165  }
166
```

Puis on crée une route table qui va permettre aux machines dans les sous-réseaux publics d'accéder à Internet via la Gateway :

```
167  ## Public (accès à internet)
168  resource "aws_route_table" "public" {
169      vpc_id = aws_vpc.vpc.id
170
171      route {
172          cidr_block = "0.0.0.0/0"
173          gateway_id = aws_internet_gateway.igw.id
174      }
175
176      tags = {
177          Name = "${var.env}-rt-public"
178      }
179  }
```

Enfin, on associe la route table précédente aux subnets publiques à l'aide des *route_table_associations*::

```
181  # Public Route Table Association
182  resource "aws_route_table_association" "public-1" {
183      subnet_id      = aws_subnet.subnet-public-1.id
184      route_table_id = aws_route_table.public.id
185  }
186
187  resource "aws_route_table_association" "public-2" {
188      subnet_id      = aws_subnet.subnet-public-2.id
189      route_table_id = aws_route_table.public.id
190  }
191
192  resource "aws_route_table_association" "public-3" {
193      subnet_id      = aws_subnet.subnet-public-3.id
194      route_table_id = aws_route_table.public.id
195  }
```

Une fois notre fichier **intra.tf** créé, on le met dans un répertoire puis on lance dans ce dossier la commande suivante :

terraform init

puis

terraform apply

```
ubuntu@ip-172-31-36-188:~/TP_CICD/Deploy_Infra$ terraform init
Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "aws" (hashicorp/aws) 3.37.0...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "... constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.aws: version = ">= 3.37"

Warning: registry.terraform.io: This version of Terraform has an outdated GPG key and is unable to verify new provider releases. Please upgrade Terraform to at least 0.12.31 to receive new
provider updates. For details see: https://discuss.hashicorp.com/t/hcsec-2021-12-codecov-security-event-and-hashicorp-gpg-key-exposure/23512

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
ubuntu@ip-172-31-36-188:~/TP_CICD/Deploy_Infra$ terraform apply

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_default_route_table.main-private will be created
```

puis

terraform plan

```
ubuntu@ip-172-31-36-73:~/TP_CICD/Deploy_Infra$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

-----

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_default_route_table.main-private will be created
+ resource "aws_default_route_table" "main-private" {
+   arn                = (known after apply)
+   default_route_table_id = (known after apply)
+   id                 = (known after apply)
+   owner_id           = (known after apply)
+   route               = [
+     {
+       cidr_block          = "0.0.0.0/0"
+       destination_prefix_list_id = ""
+       egress_only_gateway_id = ""
+       gateway_id          = ""
+       instance_id         = (known after apply)
+       ipv6_cidr_block      = ""
+       nat_gateway_id      = ""
+       network_interface_id = ""
+       transit_gateway_id   = ""
+       vpc_endpoint_id     = ""
+       vpc_peering_connection_id = ""
+     }
+   ],
+   tags                = {
+     "Name" = "dev-rt-main-private"
+   }
+   vpc_id              = (known after apply)
}
```



```

    + "Name" = "dev-vpc"
  }
}

Plan: 13 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

Enter a value: yes

aws_vpc.vpc: Creating...
aws_vpc.vpc: Still creating... [10s elapsed]
aws_vpc.vpc: Creation complete after 12s [id=vpc-075b522080b5ac195]
aws_subnet.subnet-private-2: Creating...
aws_default_route_table.main-private: Creating...
aws_internet_gateway.igw: Creating...
aws_subnet.subnet-private-1: Creating...
aws_subnet.subnet-public-3: Creating...
aws_subnet.subnet-public-2: Creating...
aws_subnet.subnet-public-1: Creating...
aws_subnet.subnet-private-3: Creating...
aws_default_route_table.main-private: Creation complete after 0s [id=rtb-0840194d31b423430]
aws_subnet.subnet-private-1: Creation complete after 0s [id=subnet-0815420ed62faea4b]
aws_subnet.subnet-private-2: Creation complete after 0s [id=subnet-041f2007609b15549]
aws_internet_gateway.igw: Creation complete after 0s [id=igw-012d5e20f04064822]
aws_route_table.public: Creating...
aws_subnet.subnet-private-3: Creation complete after 0s [id=subnet-0221eb8b2c490fc70]
aws_route_table.public: Creation complete after 1s [id=rtb-0a3822bdc4621271c]
aws_subnet.subnet-public-3: Still creating... [10s elapsed]
aws_subnet.subnet-public-2: Still creating... [10s elapsed]
aws_subnet.subnet-public-1: Still creating... [10s elapsed]
aws_subnet.subnet-public-3: Creation complete after 10s [id=subnet-054024e5ce12e0948]
aws_subnet.subnet-public-1: Creation complete after 10s [id=subnet-022df209b1a4ab757]
aws_route_table_association.public-3: Creating...
aws_route_table_association.public-1: Creating...
aws_subnet.subnet-public-2: Creation complete after 10s [id=subnet-022a25586d739a3bf]
aws_route_table_association.public-2: Creating...
aws_route_table_association.public-3: Creation complete after 1s [id=rtbassoc-0f1fa2cf1dfbc7760]
aws_route_table_association.public-1: Creation complete after 1s [id=rtbassoc-0e109f93ca423b8f2]
aws_route_table_association.public-2: Creation complete after 1s [id=rtbassoc-0b6cc94e60d6de13c]

Apply complete! Resources: 13 added, 0 changed, 0 destroyed.

```

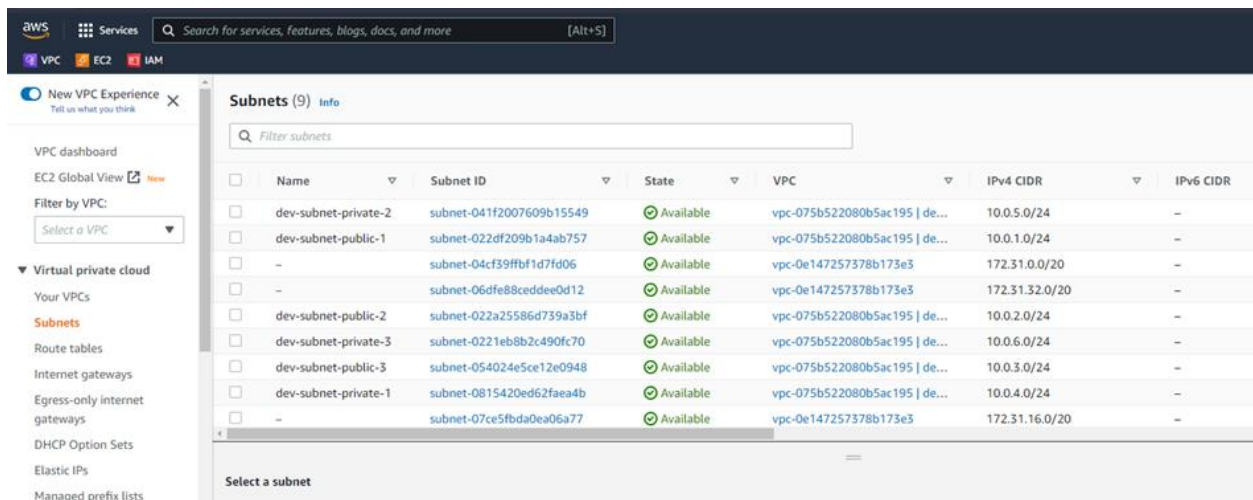
Vérifions que tous les éléments ont bien été déployés :

→ VPC :

The screenshot shows the AWS Management Console interface for VPCs. On the left, there's a sidebar with navigation options like 'VPC dashboard', 'EC2 Global View', and 'Filter by VPC'. The main area is titled 'Your VPCs (2)' and contains a table with the following data:

| | Name | VPC ID | State | IPv4 CIDR |
|--------------------------|---------|-----------------------|-----------|---------------|
| <input type="checkbox"/> | dev-vpc | vpc-075b522080b5ac195 | Available | 10.0.0.0/16 |
| <input type="checkbox"/> | - | vpc-0e147257378b173e3 | Available | 172.31.0.0/16 |

→ Subnets:



The screenshot shows the AWS Management Console interface. The top navigation bar includes the AWS logo, 'Services', and a search bar. The left sidebar shows the navigation menu with 'Subnets' highlighted. The main content area displays a list of subnets under the heading 'Subnets (9) Info'. The list includes columns for Name, Subnet ID, State, VPC, IPv4 CIDR, and IPv6 CIDR. All subnets are in an 'Available' state.

| Name | Subnet ID | State | VPC | IPv4 CIDR | IPv6 CIDR |
|----------------------|--------------------------|-----------|-------------------------------|----------------|-----------|
| dev-subnet-private-2 | subnet-041f2007609b15549 | Available | vpc-075b522080b5ac195 de... | 10.0.5.0/24 | - |
| dev-subnet-public-1 | subnet-022df209b1a4ab757 | Available | vpc-075b522080b5ac195 de... | 10.0.1.0/24 | - |
| - | subnet-04cf39ffbf1d7fd06 | Available | vpc-0e147257378b173e3 | 172.31.0.0/20 | - |
| - | subnet-06dfe88ceddee0d12 | Available | vpc-0e147257378b173e3 | 172.31.32.0/20 | - |
| dev-subnet-public-2 | subnet-022a25586d739a3bf | Available | vpc-075b522080b5ac195 de... | 10.0.2.0/24 | - |
| dev-subnet-private-3 | subnet-0221eb8b2c490fc70 | Available | vpc-075b522080b5ac195 de... | 10.0.6.0/24 | - |
| dev-subnet-public-3 | subnet-054024e5ce12e0948 | Available | vpc-075b522080b5ac195 de... | 10.0.3.0/24 | - |
| dev-subnet-private-1 | subnet-0815420ed62faea4b | Available | vpc-075b522080b5ac195 de... | 10.0.4.0/24 | - |
| - | subnet-07ce5fbd0ea06a77 | Available | vpc-0e147257378b173e3 | 172.31.16.0/20 | - |

On va désormais déployer les instances dans l'infrastructure avec le fichier **website.tf**.

On commence par initialiser le fichier Terraform et le provider AWS (région Irlande) :

```
1 provider "aws" {
2   region = "eu-west-1"
3 }
4
5 # Variables normalement dans un autre fichier (variables.tf) mais pour faire simple.... ca marche aussi !!!
6 variable "env" {
7   type    = string
8   default = "dev"
9 }
```

On recherche la dernière AMI créée avec le Name TAG **PackerAnsible-Apache** et on récupère les ressources réseau :

```
11 #####
12 # On recherche la dernière AMI créée avec le Name TAG PackerAnsible-Apache
13 data "aws_ami" "selected" {
14     owners = ["self"]
15     filter {
16         name = "state"
17         values = ["available"]
18     }
19     filter {
20         name = "tag:Name"
21         values = ["PackerAnsible-Apache"]
22     }
23     most_recent = true
24 }
25
26
27 # On recupere Les ressources reseau
28 ## VPC
29 data "aws_vpc" "selected" {
30     tags = {
31         Name = "${var.env}-vpc"
32     }
33 }
```

On précise les sous-réseaux :

```
35 ## Subnets
36 data "aws_subnet" "subnet-public-1" {
37     tags = {
38         Name = "${var.env}-subnet-public-1"
39     }
40 }
41
42 data "aws_subnet" "subnet-public-2" {
43     tags = {
44         Name = "${var.env}-subnet-public-2"
45     }
46 }
47
48 data "aws_subnet" "subnet-public-3" {
49     tags = {
50         Name = "${var.env}-subnet-public-3"
51     }
52 }
53
54 data "aws_subnet" "subnet-private-1" {
55     tags = {
56         Name = "${var.env}-subnet-private-1"
57     }
58 }
59
60 data "aws_subnet" "subnet-private-2" {
61     tags = {
62         Name = "${var.env}-subnet-private-2"
63     }
64 }
65
66 data "aws_subnet" "subnet-private-3" {
67     tags = {
68         Name = "${var.env}-subnet-private-3"
69     }
70 }
```

On définit les zones de disponibilité et les groupes de sécurité :

```
72  ## AZ zones de disponibilités dans la région
73  data "aws_availability_zones" "all" {}
74
75  #####
76  # Security Groups
77  ## ASG
78  resource "aws_security_group" "web-sg-asg" {
79      name     = "${var.env}-sg-asg"
80      vpc_id = data.aws_vpc.selected.id
81      egress {
82          from_port = 0
83          to_port   = 0
84          protocol  = "-1"
85          cidr_blocks = ["0.0.0.0/0"]
86      }
87      ingress {
88          from_port = 443
89          protocol  = "tcp"
90          to_port   = 443
91          security_groups = [aws_security_group.web-sg-elb.id] # on autorise en
92      }
93      lifecycle {
94          create_before_destroy = true
95      }
96  }
97  ## ELB
98  resource "aws_security_group" "web-sg-elb" {
99      name     = "${var.env}-sg-elb"
100     vpc_id = data.aws_vpc.selected.id
101     egress {
102         from_port = 0
103         to_port   = 0
104         protocol  = "-1"
105         cidr_blocks = ["0.0.0.0/0"]
106     }
107     ingress {
108         from_port = 443
109         protocol  = "tcp"
110         to_port   = 443
111         cidr_blocks = ["0.0.0.0/0"] # Normalement Ouvert sur le web sauf dans
112     }
113     lifecycle {
114         create_before_destroy = true
115     }
116 }
```

Puis on définit la configuration au lancement ainsi que le groupe d'auto-scaling :

```
117 #####
118 # ASG Launch Configuration
119 resource "aws_launch_configuration" "web-lc" {
120     image_id      = data.aws_ami.selected.id
121     instance_type = "t2.micro"
122     # key_name = "" # Si vous voulez utiliser une KeyPair pour vous connecter aux instances
123     security_groups = [aws_security_group.web-sg-asg.id]
124     lifecycle {
125         create_before_destroy = true
126     }
127 }
128
129 # ASG
130 resource "aws_autoscaling_group" "web-asg" {
131     launch_configuration = aws_launch_configuration.web-lc.id
132     #availability_zones   = data.aws_availability_zones.all.names
133     vpc_zone_identifier   = [data.aws_subnet.subnet-private-1.id, data.aws_subnet.subnet-private-2.id, data.aws_subnet.subnet-private-3.id]
134     load_balancers        = [aws_elb.web-elb.name]
135     health_check_type     = "ELB"
136
137     max_size = 4
138     min_size = 2
139
140     tag {
141         key       = "name"
142         value     = "${var.env}-asg"
143         propagate_at_launch = true
144     }
145     lifecycle {
146         create_before_destroy = true
147     }
148 }
```

On met en place l'Elastic Load Balancer

```
150 # ELB
151 resource "aws_elb" "web-elb" {
152     name            = "${var.env}-elb"
153     subnets        = [data.aws_subnet.subnet-public-1.id, data.aws_subnet.subnet-public-2.id, data.aws_subnet.subnet-public-3.id]
154     security_groups = [aws_security_group.web-sg-elb.id]
155
156     listener {
157         instance_port     = 443
158         instance_protocol = "http"
159         lb_port           = 443
160         lb_protocol       = "http"
161     }
162
163     health_check {
164         healthy_threshold   = 2
165         interval            = 30
166         target               = "HTTP:443/"
167         timeout             = 3
168         unhealthy_threshold = 2
169     }
170     lifecycle {
171         create_before_destroy = true
172     }
173 }
```

On définit les règles d'auto-scaling de notre groupe ainsi que les alarmes

```
175 #####
176
177 # Autoscaling Scale Policies
178 ## scale up
179 ### ASG Policy
180 resource "aws_autoscaling_policy" "web-cpu-policy-scaleup" {
181     name = "web-cpu-policy"
182     autoscaling_group_name = aws_autoscaling_group.web-asg.name
183     adjustment_type = "ChangeInCapacity"
184     scaling_adjustment = "1"
185     cooldown = "300"
186     policy_type = "SimpleScaling"
187 }
188
189 ## Cloudwatch Alarm
190 resource "aws_cloudwatch_metric_alarm" "web-cpu-alarm-scaleup" {
191     alarm_name = "web-cpu-alarm-ScaleUp"
192     comparison_operator = "GreaterThanOrEqualToThreshold"
193     evaluation_periods = "2"
194     metric_name = "CPUUtilization"
195     namespace = "AWS/EC2"
196     period = "120"
197     statistic = "Average"
198     threshold = "70"
199     dimensions = {
200         "AutoScalingGroupName" = aws_autoscaling_group.web-asg.name
201     }
202     actions_enabled = true
203     alarm_actions = [aws_autoscaling_policy.web-cpu-policy-scaleup.arn]
204 }
205
206 ## scale down
207 ### ASG Policy
208 resource "aws_autoscaling_policy" "web-cpu-policy-scaledown" {
209     name = "web-cpu-policy-scaledown"
210     autoscaling_group_name = aws_autoscaling_group.web-asg.name
211     adjustment_type = "ChangeInCapacity"
212     scaling_adjustment = "-1"
213     cooldown = "300"
214     policy_type = "SimpleScaling"
215 }
216
217 ### CloudWatch Alarm
218 resource "aws_cloudwatch_metric_alarm" "web-cpu-alarm-scaledown" {
219     alarm_name = "web-cpu-alarm-scaledown"
220     comparison_operator = "LessThanOrEqualToThreshold"
221     evaluation_periods = "2"
222     metric_name = "CPUUtilization"
223     namespace = "AWS/EC2"
224     period = "120"
```

```
233 # Outputs normalement dans un autre fichier(Outputs.tf) mais pour faire simple...
234 ## On renvoie le nom DNS de l'ELB pour s'y connecter (Compter quelques minutes avant disponibilité au premier déploiement)
235 output "elb_dns_name" {
236     description = "The DNS name of the ELB"
237     value = aws_elb.web-elb.dns_name
238 }
```

On met enfin le fichier **website.tf** dans un dossier puis on lance dans ce dossier les commandes suivantes :

terraform init

```
ubuntu@ip-172-31-36-73:~/TP_CICD/Deploy_WebSite$ terraform init
Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "aws" (hashicorp/aws) 3.37.0...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.aws: version = ">= 3.37"

Warning: registry.terraform.io: This version of Terraform has an outdated GPG key and is unable to verify new provider releases. Please upgrade Terraform to at least 0.12.31 to receive new provider updates. For details see: https://discuss.hashicorp.com/t/hcsec-2021-12-codecov-security-event-and-hashicorp-gpg-key-exposure/23512

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
ubuntu@ip-172-31-36-73:~/TP_CICD/Deploy_WebSite$
```

puis

terraform plan

```
ubuntu@ip-172-31-36-73:~/TP_CICD/Deploy_WebSite$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

data.aws_subnet.subnet-public-3: Refreshing state...
data.aws_vpc.selected: Refreshing state...
data.aws_availability_zones.all: Refreshing state...
data.aws_subnet.subnet-private-2: Refreshing state...
data.aws_subnet.subnet-public-1: Refreshing state...
data.aws_subnet.subnet-private-3: Refreshing state...
data.aws_subnet.subnet-public-2: Refreshing state...
data.aws_ami.selected: Refreshing state...
data.aws_subnet.subnet-private-1: Refreshing state...

-----

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

# aws_autoscaling_group.web-asg will be created
+ resource "aws_autoscaling_group" "web-asg" {
  + arn                = (known after apply)
  + availability_zones = (known after apply)
  + default_cooldown   = (known after apply)
  + desired_capacity   = (known after apply)
  + force_delete       = false
  + force_delete_warm_pool = false
  + health_check_grace_period = 300
  + health_check_type   = "ELB"
  + id                 = (known after apply)
  + launch_configuration = (known after apply)
  + load_balancers      = [
    + "dev-elb",
  ]
  + max_size           = 4
}
```

Cela permet de s'assurer qu'il n'y ait pas d'erreur.

terraform apply

```
ubuntu@ip-172-31-38-188:~/TP_CICD/Deploy_WebSite$ terraform apply
data.aws_subnet.subnet-public-3: Refreshing state...
data.aws_subnet.subnet-private-3: Refreshing state...
data.aws_ami.selected: Refreshing state...
data.aws_vpc.selected: Refreshing state...
data.aws_subnet.subnet-public-1: Refreshing state...
data.aws_subnet.subnet-public-2: Refreshing state...
data.aws_subnet.subnet-private-1: Refreshing state...
data.aws_availability_zones.all: Refreshing state...
data.aws_subnet.subnet-private-2: Refreshing state...

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

aws_autoscaling_policy.web-cpu-policy-scaleup: Creating...
aws_autoscaling_policy.web-cpu-policy-scaledown: Creation complete after 0s [id=web-cpu-policy-scaledown]
aws_autoscaling_policy.web-cpu-policy-scaleup: Creation complete after 0s [id=web-cpu-policy-scaleup]
aws_cloudwatch_metric_alarm.web-cpu-alarm-scaledown: Creating...
aws_cloudwatch_metric_alarm.web-cpu-alarm-scaleup: Creating...
aws_cloudwatch_metric_alarm.web-cpu-alarm-scaledown: Creation complete after 0s [id=web-cpu-alarm-scaledown]
aws_cloudwatch_metric_alarm.web-cpu-alarm-scaleup: Creation complete after 1s [id=web-cpu-alarm-scaleup]

Apply complete! Resources: 9 added, 0 changed, 0 destroyed.

Outputs:
elb_dns_name = dev-elb-1677684786.eu-west-1.elb.amazonaws.com
ubuntu@ip-172-31-38-188:~/TP_CICD/Deploy_WebSite$ |
```

Vérifions que les 2 instances ont bien été déployées

| Instances (3) Info | | | | | | | | | | |
|-------------------------------------|-----------|---------------------|----------------|---------------|-------------------|--------------|-------------------|--------------------------|-----------------|------------|
| <input type="text" value="Search"/> | | | | | | | | | | |
| <input type="checkbox"/> | Name | Instance ID | Instance state | Instance type | Status check | Alarm status | Availability Zone | Public IPv4 DNS | Public IPv4 ... | Elastic IP |
| <input type="checkbox"/> | - | i-04393567074e60u97 | Running | t2.micro | 2/2 checks passed | No alarms | eu-west-1c | - | - | - |
| <input type="checkbox"/> | ToolsCICD | i-0428af0dd9dfc2f81 | Running | t2.micro | 2/2 checks passed | No alarms | eu-west-1b | ec2-34-252-156-169.eu... | 34.252.156.169 | - |
| <input type="checkbox"/> | - | i-0f84e77cb2953e047 | Running | t2.micro | 2/2 checks passed | No alarms | eu-west-1b | - | - | - |

| Public IPv4 ... | Elastic IP | IPv6 IPs | Monitoring | Security group name | Key name | Launch time |
|-----------------|------------|----------|------------|---------------------|----------|------------------------|
| - | - | - | enabled | dev-sg-asg | - | 2022/06/24 17:05 GMT+2 |
| 34.252.156.169 | - | - | disabled | launch-wizard-1 | TP3_CICD | 2022/06/24 15:10 GMT+2 |
| - | - | - | enabled | dev-sg-asg | - | 2022/06/24 17:05 GMT+2 |

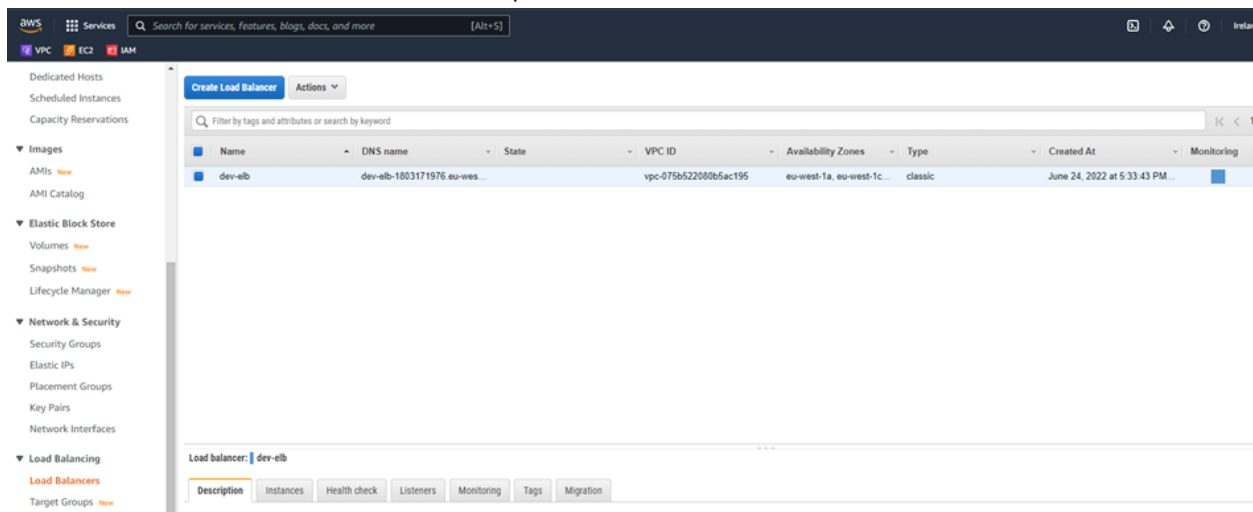
BINGO ✓

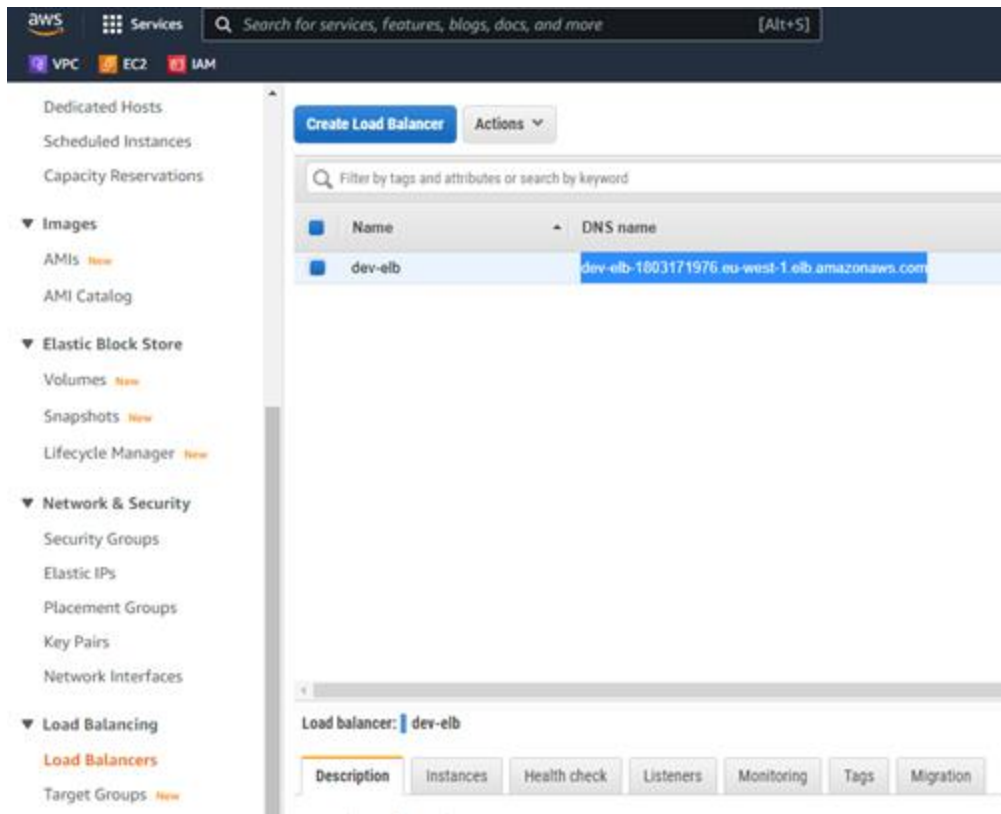
Cependant, nous remarquons que les 2 instances n'ont pas d'IP publique car elles sont derrière un Load Balancer.

L'intérêt du Load Balancer est de distribuer automatiquement le trafic entrant entre plusieurs cibles disponibles tels que les instances EC2, les containers Docker ou les adresses IP. Cela permet de répartir la charge vers les zones saines et d'éviter notamment des incidents tels que du DDoS.

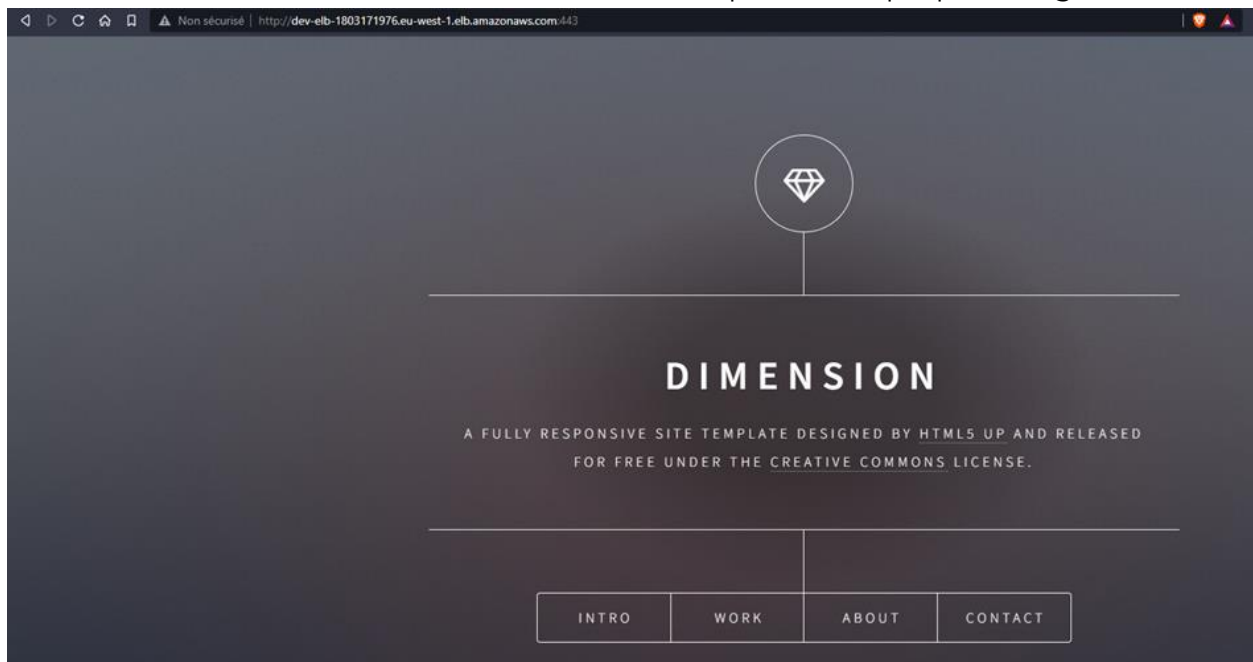
Si l'une des instances n'est plus disponible, le Load Balancer redirigera le service vers l'autre instance.

Pour se connecter à l'instance faisant tourner le site web, on doit se connecter via le nom DNS du Load Balancer sur le port 443 :





Nous avons désormais bien accès au site Web depuis notre propre navigateur :



Conclusion

En conclusion, ces divers travaux pratiques nous ont permis de découvrir des outils centraux dans le domaine du DevOps.

Nous avons pu notamment découvrir Amazon AWS qui est un provider très largement utilisé et dont les utilisations outrepassent le DevOps.

Des outils comme Ansible, Packer, Terraform, nous ont permis d'apprendre et mettre en pratique le principe d'Infrastructure-as-a-Code, notion clé du DevOps, permettant de déployer des infra ISO en un rien de temps.