Documentation de l'infrastructure cloud AWS – Projet 10

Procédure rédigée par : Vassenet-Guihot Romain

OPENCLASSROOMS

Table des matières

I - Introduction :

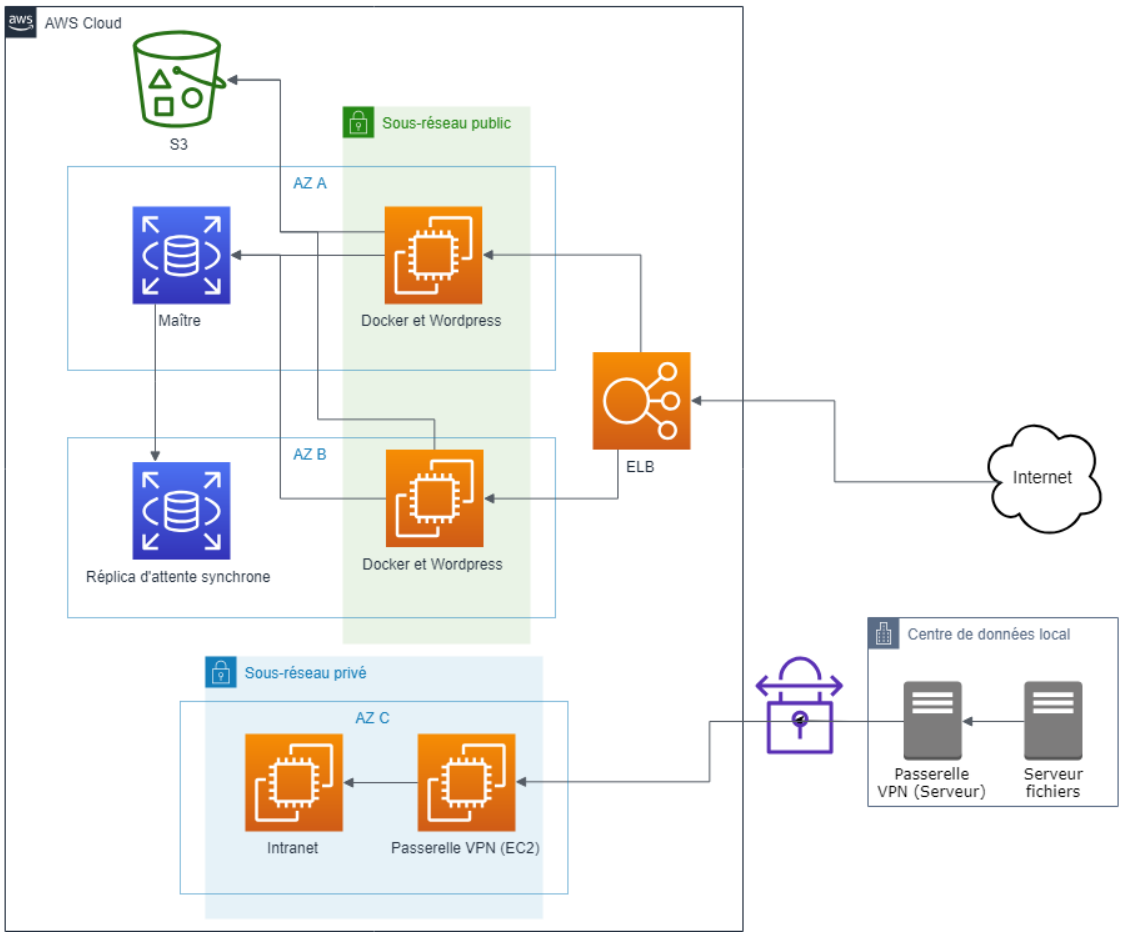
a) – Présentation du scénario

« Vous travaillez dans une petite entreprise qui se développe à l'international. L'entreprise ne dispose que de quelques machines pour la gestion de ses outils internes et pour héberger son site internet. Des incidents matériels sur le serveur de messagerie ont pénalisé l'entreprise le mois dernier. Le directeur veut éviter que la situation ne se reproduise mais réalise que fiabiliser les services en interne coûterait très cher.

L'entreprise décide donc de migrer l'ensemble de son SI vers le cloud. Les outils collaboratifs seront migrés vers Google Apps et le site de l'entreprise sera hébergé sur AWS.

L'entreprise garde simplement un serveur de fichiers dans ses locaux pour profiter de la vitesse du réseau local pour le transfert de fichiers volumineux. Ce serveur est également accessible par une liaison VPN depuis un site intranet privé hébergé sur AWS »

Schéma indicatif de l'infrastructure



b) – Instructions d'architecture

- Installez un serveur WordPress (le site de l'entreprise) sur AWS en utilisant :
    - RDS pour le stockage de la base de données
    - S3 pour le stockage des médias (via le plugin amazon-web-services)
    - EC2 et Docker pour le serveur web
    - ELB pour distribuer les requêtes sur les instances EC2
    - CloudFormation pour automatiser la création de l'infrastructure
- Tous les éléments de votre infrastructure publique devront être répartis sur plusieurs zones de disponibilité (multi-AZ). Vous utiliserez le service ELB pour la répartition des requêtes vers les différentes zones de disponibilité (AZ).
- Vous monterez une instance EC2 destinée à héberger l'application intranet sur un sous-réseau privé. Dans le cadre de ce cours, le contenu de l'intranet sera une simple page web HTML.
- En local sur votre machine, vous créerez deux machines virtuelles Linux : une pour le serveur de fichiers et une pour le serveur VPN.
- Vous établirez une liaison VPN entre votre serveur VPN local et le sous-réseau privé AWS via une instance EC2.
- Vous mettrez en place de l'auto-scaling sur les instances EC2 pour augmenter le nombre de machines dès que la charge CPU des serveurs atteint 80% en moyenne sur 5 minutes et vous veillerez à être informé par un mail à chaque fois que l'événement survient.
- Vous évaluerez les coûts de votre infrastructure AWS à partir de différentes hypothèses d'usage que vous formulerez.

II - Construire le Réseau AWS :

a) Choisir sa région :

Il est nécessaire de définir une région dans laquelle notre infrastructure devra être installée et déployée c'est pourquoi il est recommandé de répondre à différents critères comme la région des utilisateurs finaux (temps de réponse), les coûts horaires entre les régions, les lois du droit international (ex : respect des normes en vigueur en fonction du pays pour la prestation de service gratuite ou payante par application.) Dans ce projet nous choisirons la zone « Ohio » pour laquelle le coût horaire des machines est le plus intéressant.

t2.nano	1	Variable	0,5 Gio	EBS uniquement	0,0058 USD par heure

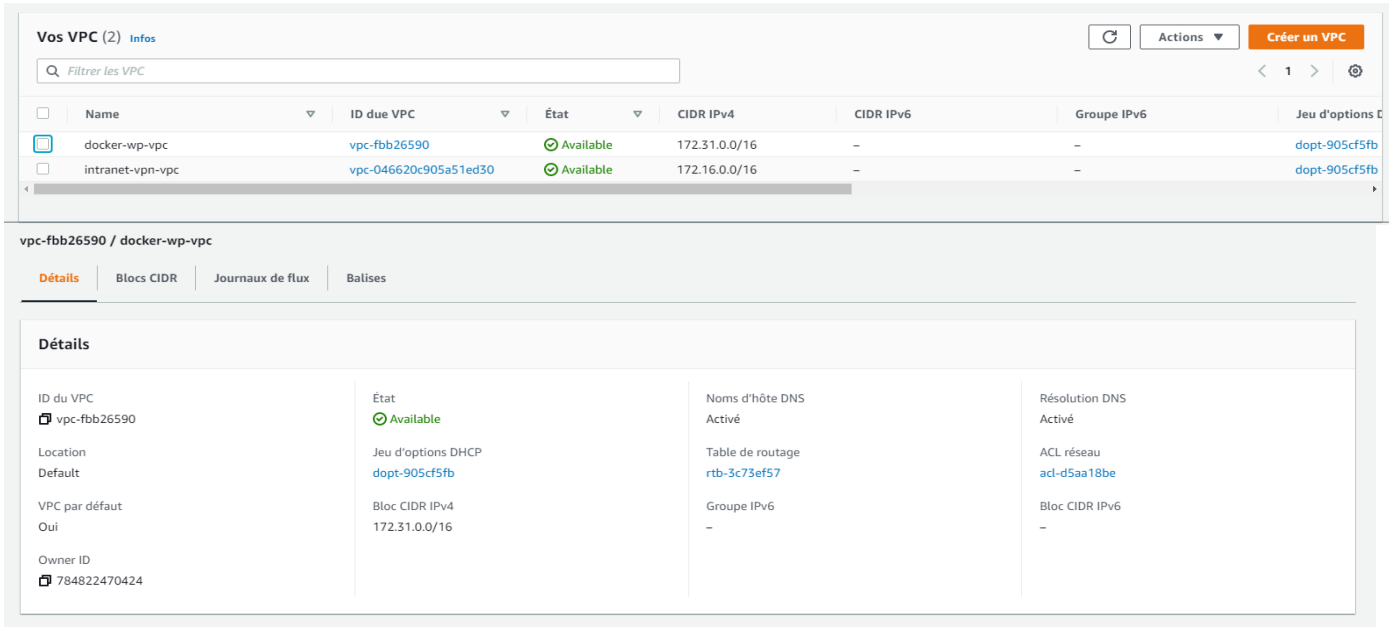
Pour en savoir plus sur les tarifications amazon :
https://aws.amazon.com/fr/emr/pricing/
https://aws.amazon.com/fr/ec2/pricing/on-demand/

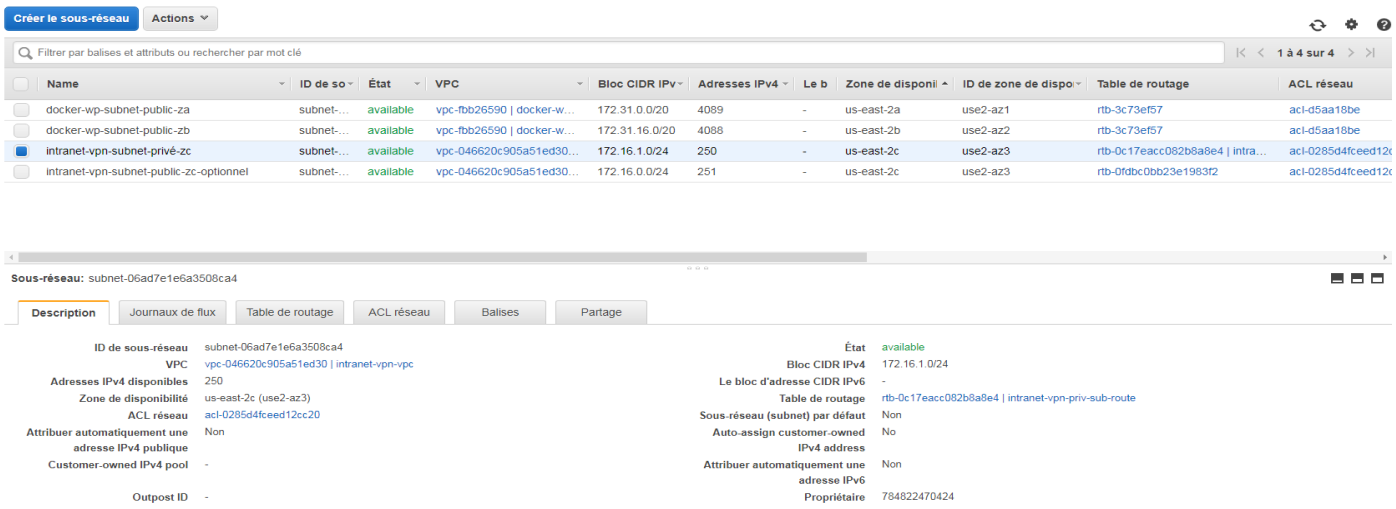
b) VPC & Sous-Réseaux

Un VPC (Virtual Private Cloud) est un réseau privé virtuel. Il s'agit d'une section réseau isolée, dans laquelle nous pourrons mettre en place les machines. Un VPC réside dans une région AWS.

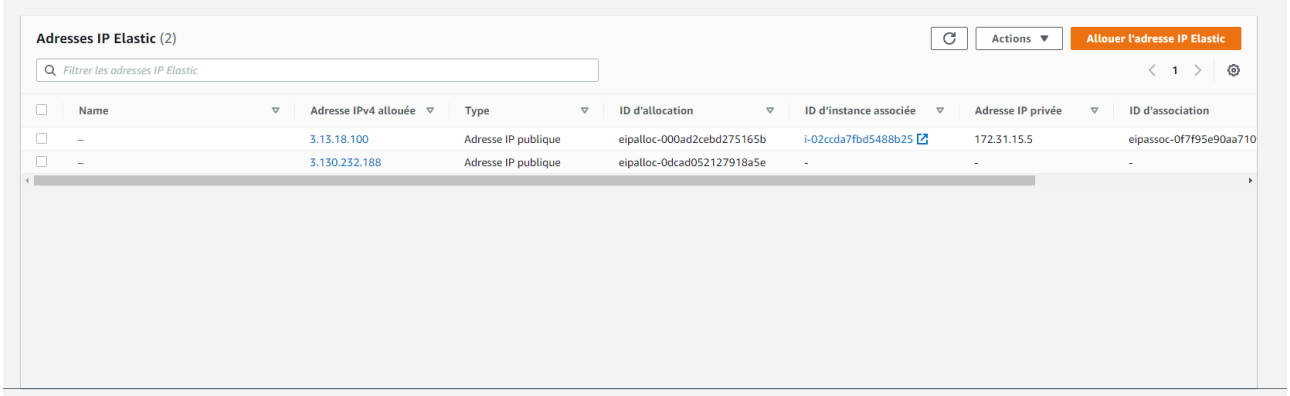
Un VPC couvre toutes les zones de disponibilité de la région. Au sein d'un VPC, nous pourrons déployer un ou plusieurs **sous-réseaux** dans chaque zone de disponibilité.

Un sous-réseau est une portion d'un VPC, et par conséquent correspond à un sous-ensemble du bloc CIDR du VPC. C'est pourquoi il est fortement recommandé d'adresser au VPC un masque de sous réseau plus large que des masques des sous réseaux.
Par exemple un VPC en /16 et ses sous réseaux en /24.





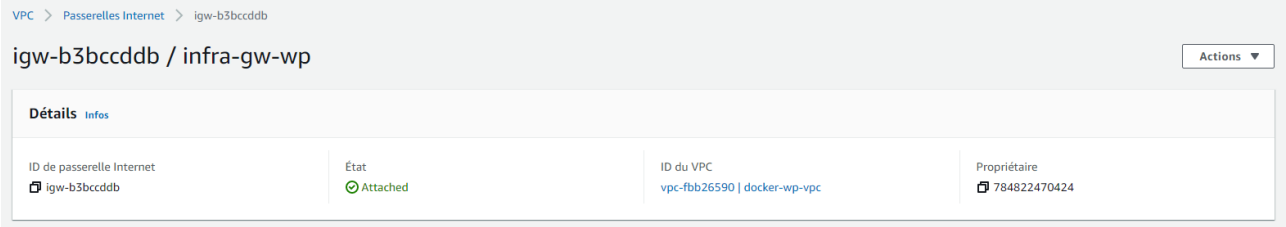
Il est également possible de louer une adresse IP publique statique appelée « elastic ip » que nous pourrons si nécessaire attacher à une instance ou à une interface réseau.



Afin de communiquer avec l'extérieur, AWS met en place des passerelles de sortie ; il en existe deux sortes :

- les **passerelles Internet**, permettant de router du trafic réseau dans et hors de votre VPC. Les passerelles Internet supportent l'IPv4 et l'IPv6 ;
- les **passerelles de sortie** uniquement, permettant de router du trafic réseau uniquement en sortie de votre VPC. Cela peut être une alternative à la création d'une instance NAT ou d'une passerelle NAT. Les passerelles de sortie uniquement ne supportent que l'IPv6.

Après avoir créé une passerelle internet, nous devons l'attacher à notre VPC.



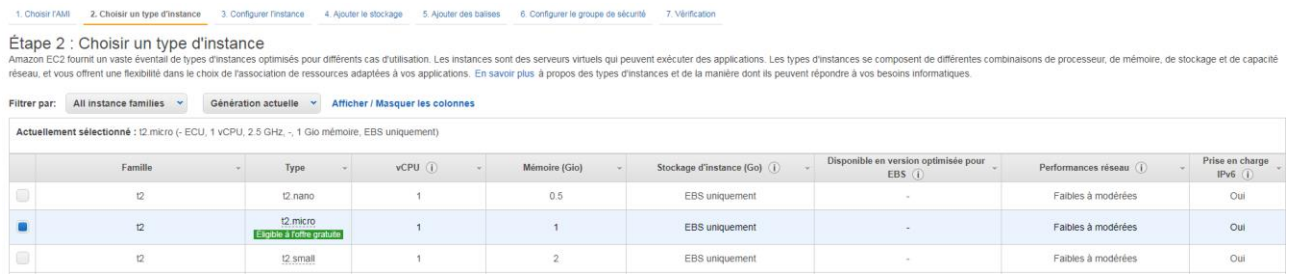
III - Systèmes EC2, RDS, S3

a) Installation & Sécurité

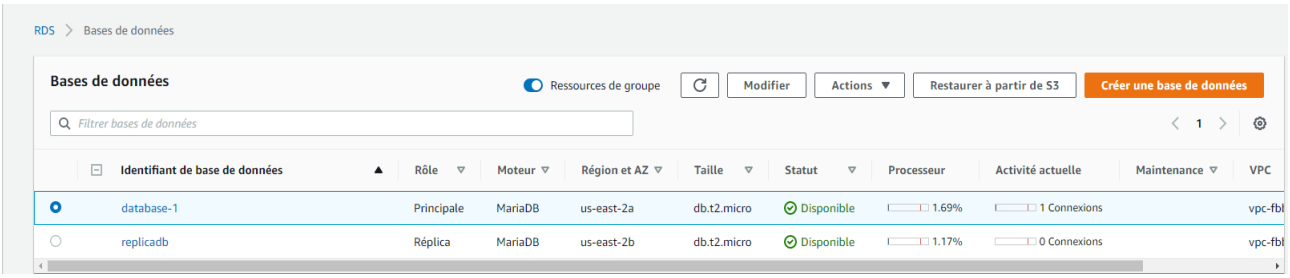
Les machines ou instances de distribution (AMI) Linux ou Windows s'installe et se paramètre grâce au service « EC2 »

Un ensemble d'options peuvent être configurés afin de répondre à différents besoins (Type d'instance : mémoire, cpu, stockage ; groupe de sécurité, zone de disponibilité)

Dans ce projet nous utiliserons l'offre gratuite disponible (t2.micro)

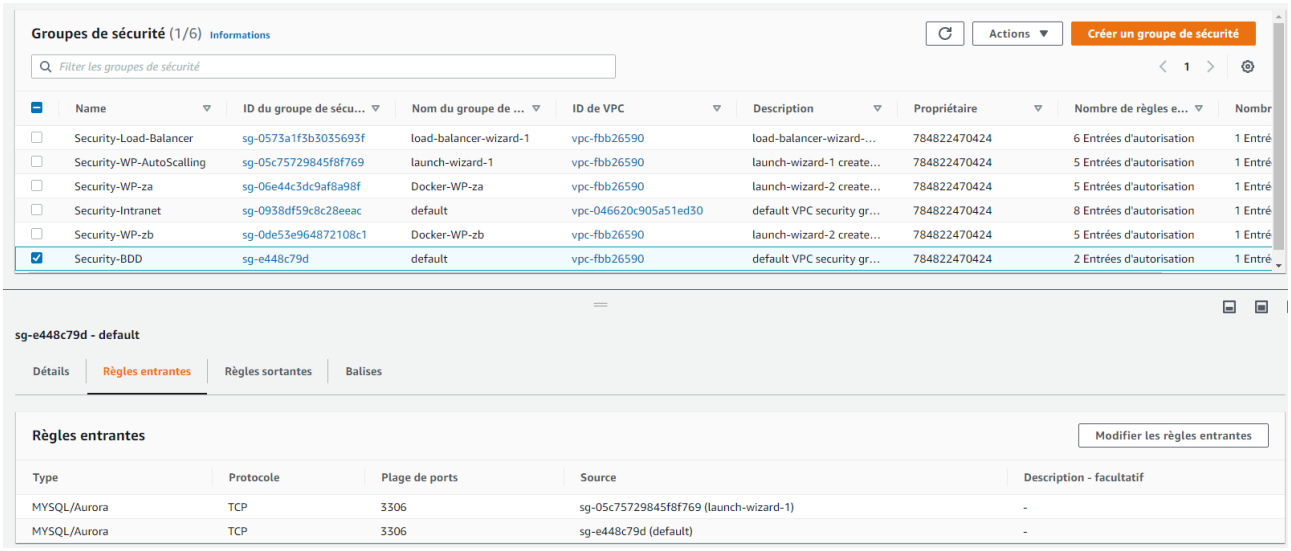


De la même façon nous paramétrons la base de donnée dans le service « RDS »



Il est également important d'apporter un maximum de sécurité en filtrant le Trafic entrant/sortant.

C'est pourquoi nous devons également configurer les groupes de sécurité en fonction des besoins comme accepter les requêtes HTTP/HTTPS (80 / 443) pour les réseaux publics ou sera installé le docker wp en zone multi a-z ou bien autoriser l'écoute unique du port 3306 pour le GS base de donnée de manière à éviter la découverte du réseau par icmp par exemple et autoriser uniquement la connexion des machines EC2 Docker/Wordpress à la BDD.



b) Configuration Docker & Wordpress :

Voici la liste des commandes utilisées afin d'installer docker composer et la création d'une base de données sécurisé avec un utilisateur non root.

Installation de Docker :

```
sudo apt-get update

sudo apt install apt-transport-https ca-certificates curl software-properties-common

curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian stretch stable"

apt-cache policy docker-ce

sudo apt install docker-ce


sudo curl -L https://github.com/docker/compose/releases/download/1.25.5/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
```

Création d'une base de donnée dédiée à Wordpress avec utilisateurs spécifique

```
apt-get install mariadb-client
```

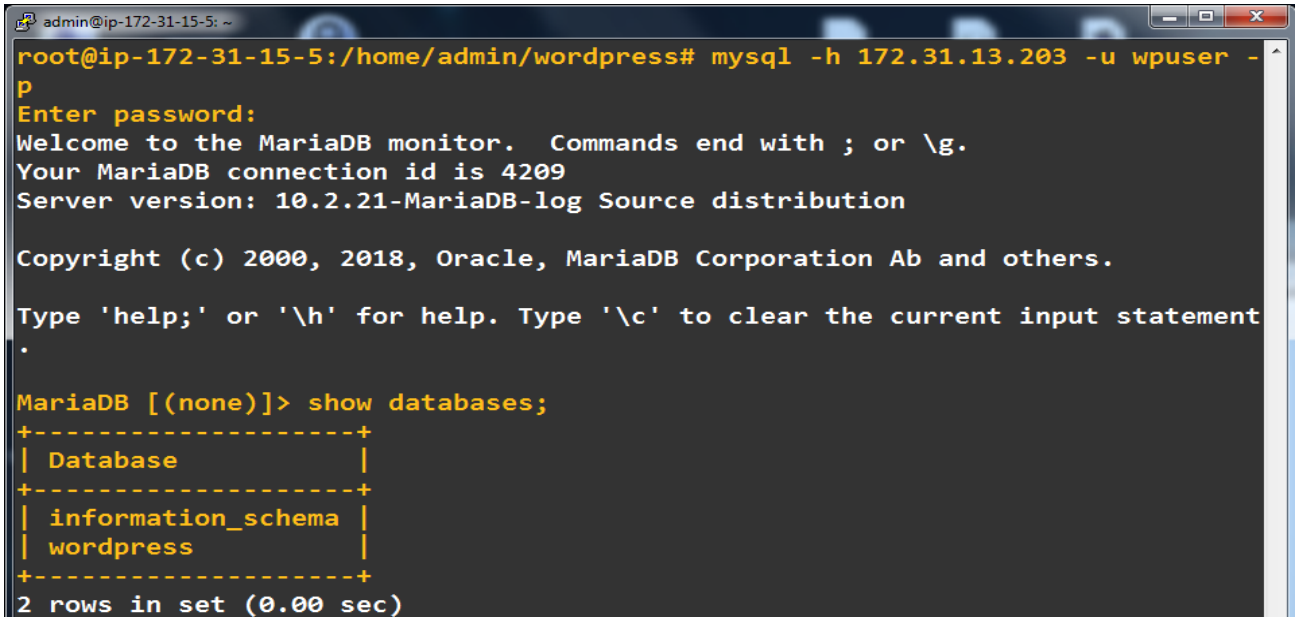
mysql -h database-1.czq2xzhlagls.us-east-2.rds.amazonaws.com -u admin -p (cnx à la bdd rds parent : mot de passe : awsawsocrocr)

CREATE DATABASE wordpress CHARACTER SET utf8 COLLATE utf8_general_ci; (creation de la bdd "wordpress" dans la bdd parent)

GRANT ALL PRIVILEGES ON wordpress.* TO 'wpuser'@'%' identified by 'wppassword';

mysql -h 172.31.13.203 -u wpuser -p (connection a la BDD RDS)

(wppassword)



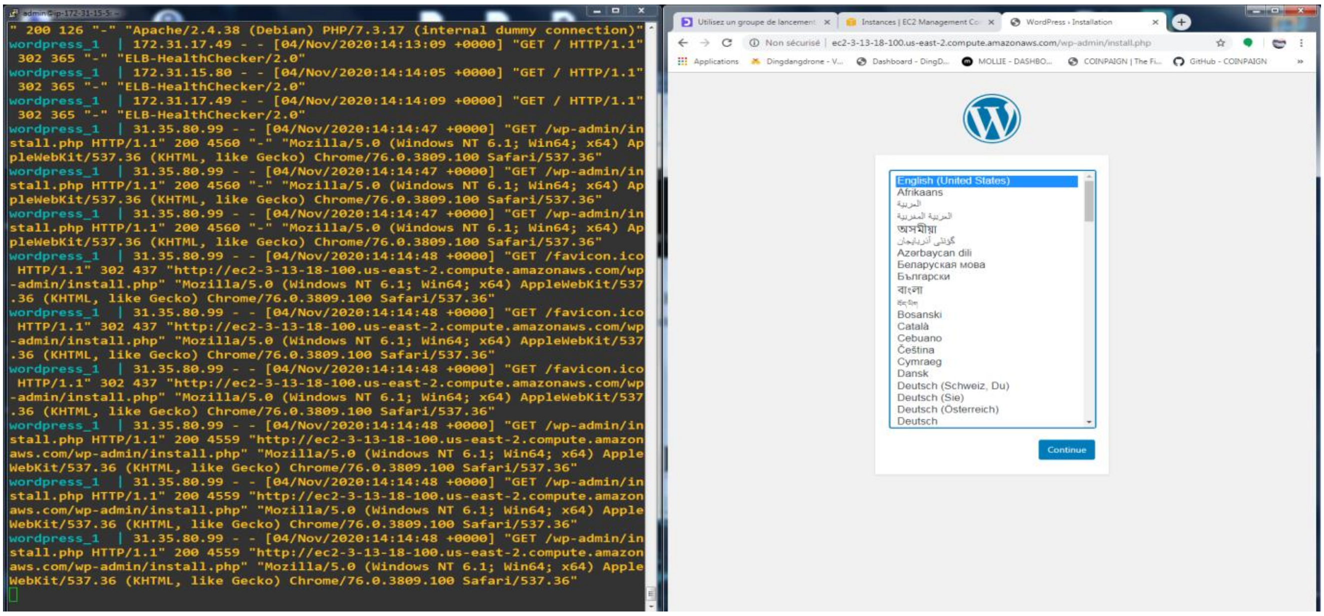
nano docker-compose.yml (creation de l'installation manuel wordpress docker)

Configuration de docker afin de connecter automatiquement le service cms wordpress sous conteneur à sa bdd via Docker-compose.yml

```
version: '2'


services:

  wordpress:

    image: wordpress:latest

    ports:

      - "80:80"

    restart: always
```

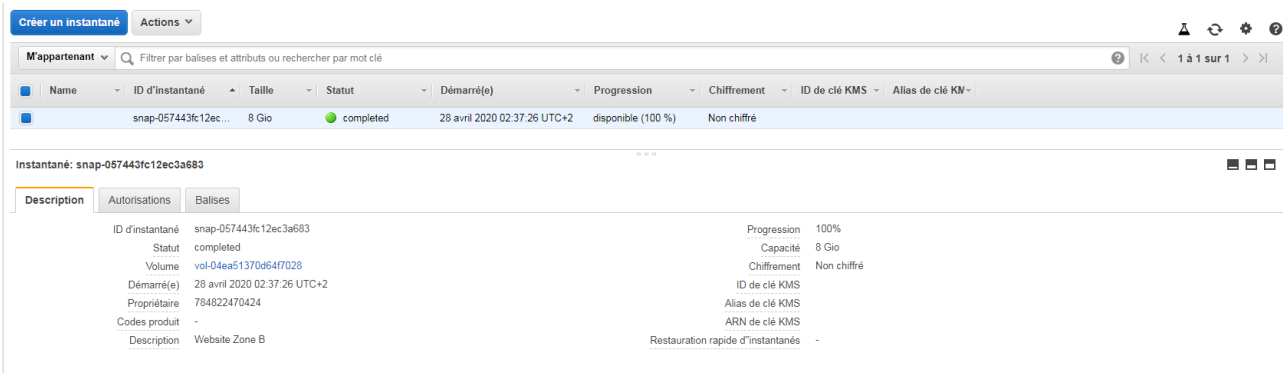
```
 environment:
   WORDPRESS_DB_HOST: database-1.czq2xzhlagls.us-east-2.rds.amazonaws.com:3306
   WORDPRESS_DB_USER: wpuser
   WORDPRESS_DB_PASSWORD: wppassword
   WORDPRESS_DB_DATABASE: wordpress

docker-compose up -d ( lancement du docker compose ( sudo avant si erreur daemon ) )
```

Les connexions HTTP sont désactivé dans le VPC par default. Il faut donc les autoriser à partir du groupe de sécurité et en modifiant les règles de connexions entrantes en ajoutant le protocol http

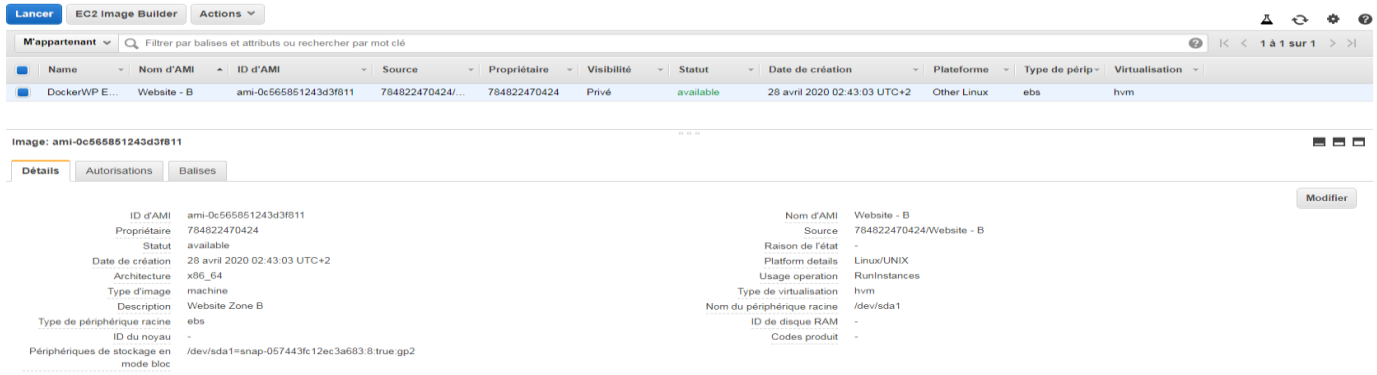


c) Snapshots & AMI personnalisée(s)

De manière à reproduire facilement notre machine ec2 de la zone A sur la zone B sans avoir à refaire l'ensemble des configurations précédentes (système uniquement. Certains paramètres devront être ajouter à cette nouvelle instance comme le groupe de sécurité ou la modification des instances à intégrer dans l'auto-scalling groupe) il faut créer une sauvegarde à un instant t du notre disque dur aussi appelé snapshot.

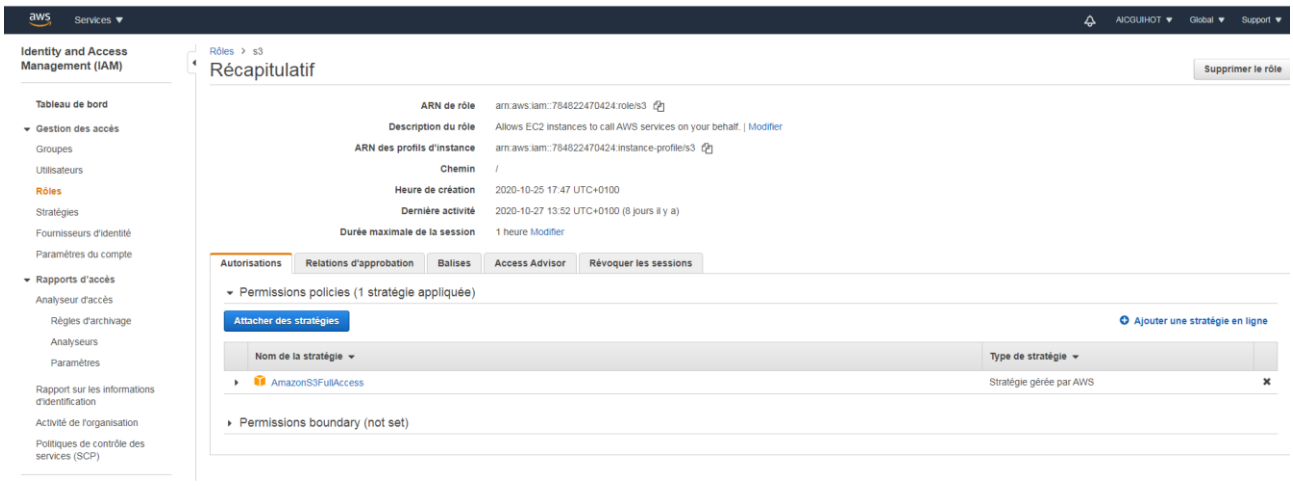


Après quoi il nous restera à créer une image système basé sur notre snapshot. Celle-ci sera utilisé pour choisir notre AMI personnalisé lorsque nous devrons déployer une nouvelle instance dans la zone B ou une mise à l'échelle automatisé en cas de Trafic élevé.

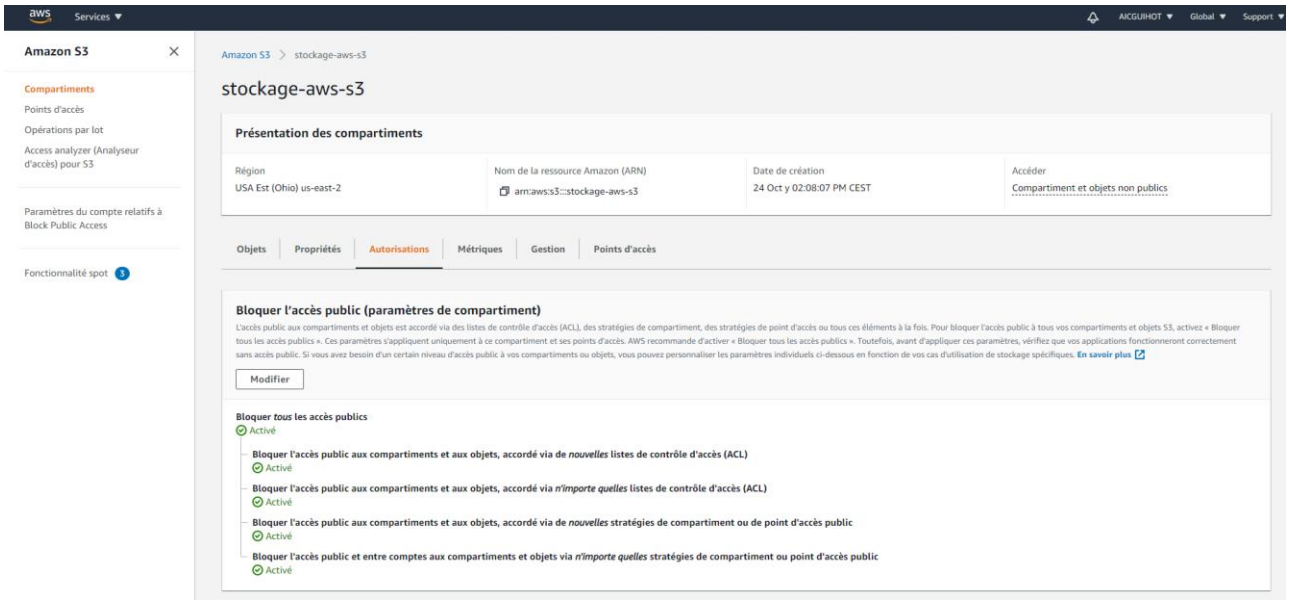


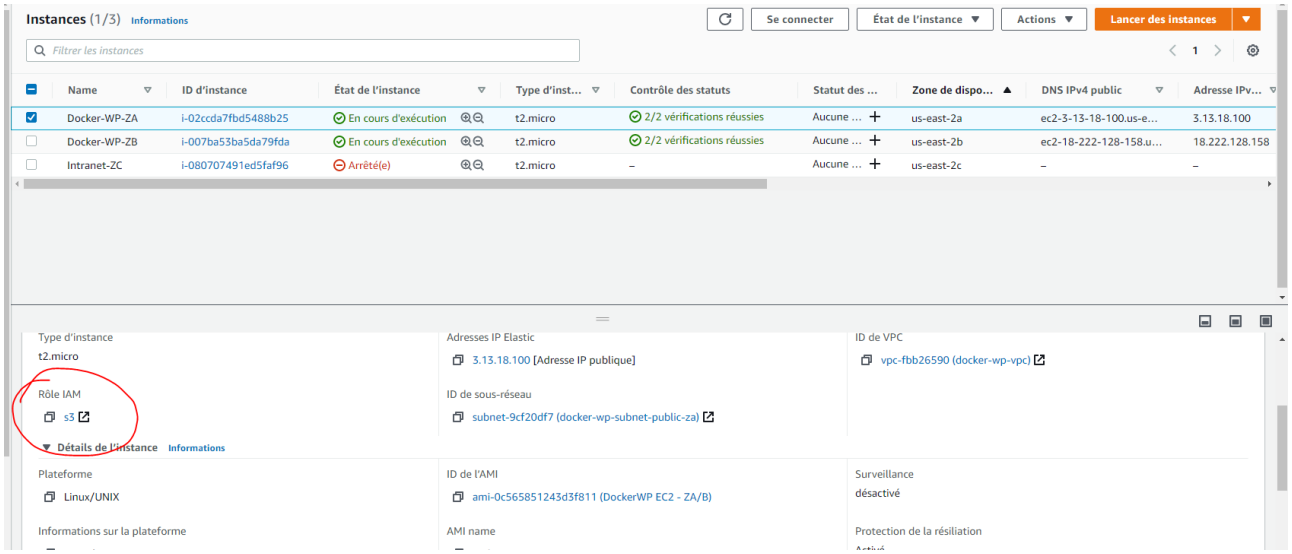
d) Bucket S3 & Role IAM

Pour connecter ses compartiments S3 depuis nos instances EC2, il nous faut :
Créer et joindre un rôle de profil AWS Identity and Access Management (IAM) aux instances qui autorise l'accès à Amazon S3.
Vérifier que la stratégie de compartiment S3 ne possède pas de stratégie refusant l'accès.
Confirmer la connectivité réseau entre l'instance EC2 et Amazon S3.

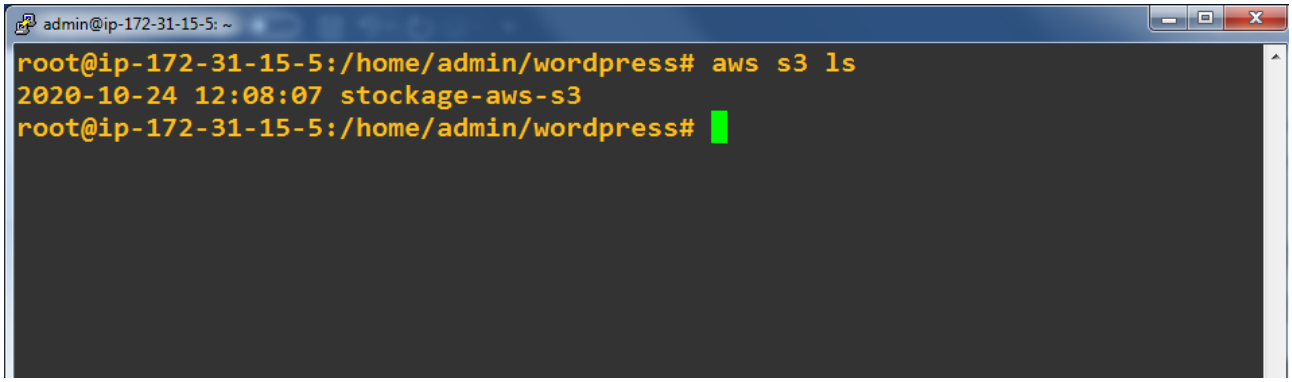


Dans notre scénario il est bien sur évident que le bucket ne doit pas être accessible depuis l'extérieur mais uniquement depuis nos instance EC2 comme serveur de média.



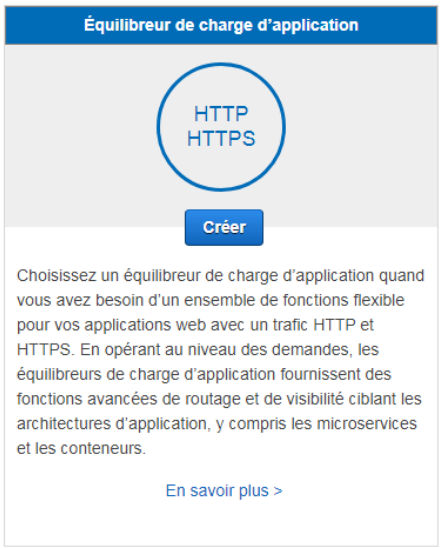


Puis nous pouvons vérifier l'accès du compartiment S3 (crée depuis la console via le service s3) depuis nos instances EC2 de la zone A/B



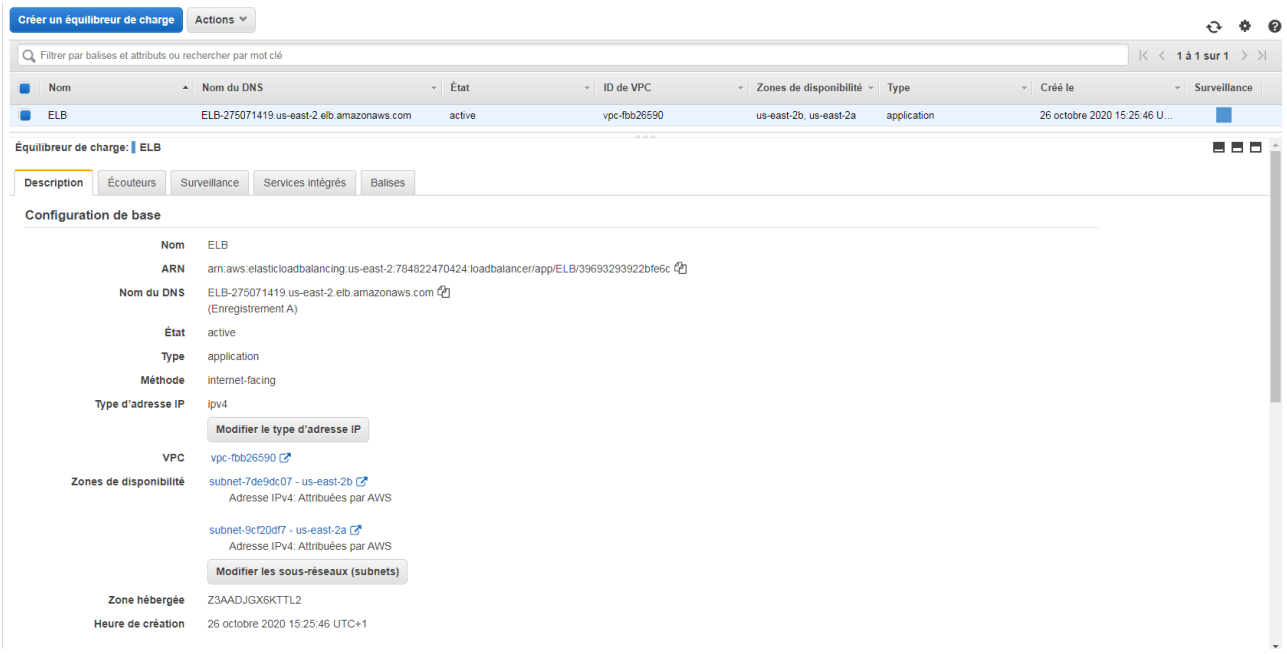
IV - Load Balancer & Mise à l'échelle

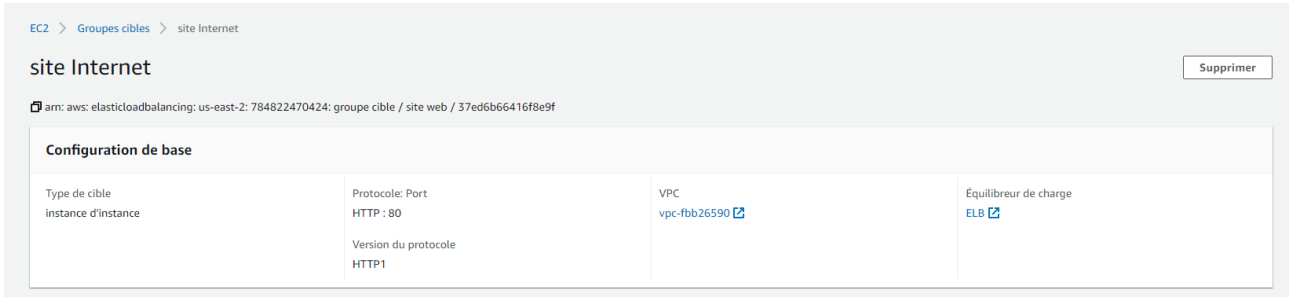
AWS fournit un **service d'équilibrage de charge** ou ***Load Balancer***, c'est-à-dire un composant qui va distribuer les requêtes réseau vers différentes machines, afin de **répartir** la charge sur plusieurs éléments de l'infrastructure, permettant de paralléliser les traitements.

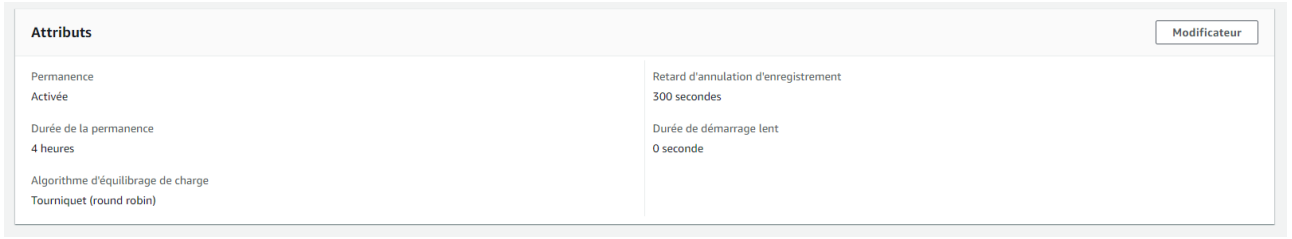


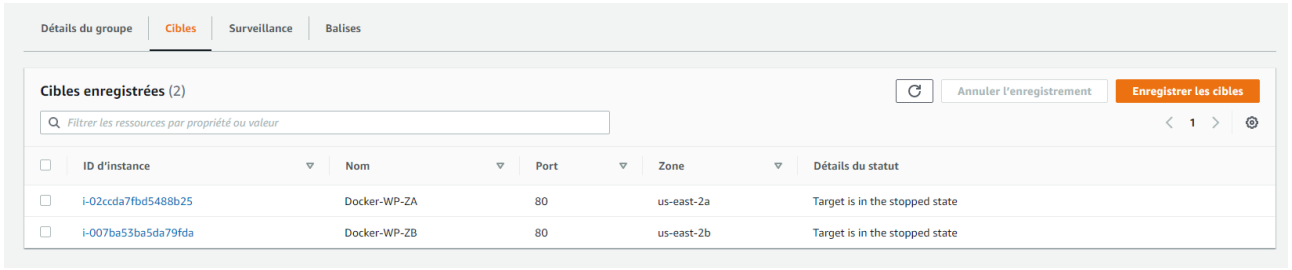
a) Configuration du Load Balancing

Les options indispensables dans notre cas sont d'ajouter la disponibilité multi a-z grâce à leurs sous-réseaux respectif de manière à équilibrer la charge entre les machines EC2 des deux zones.

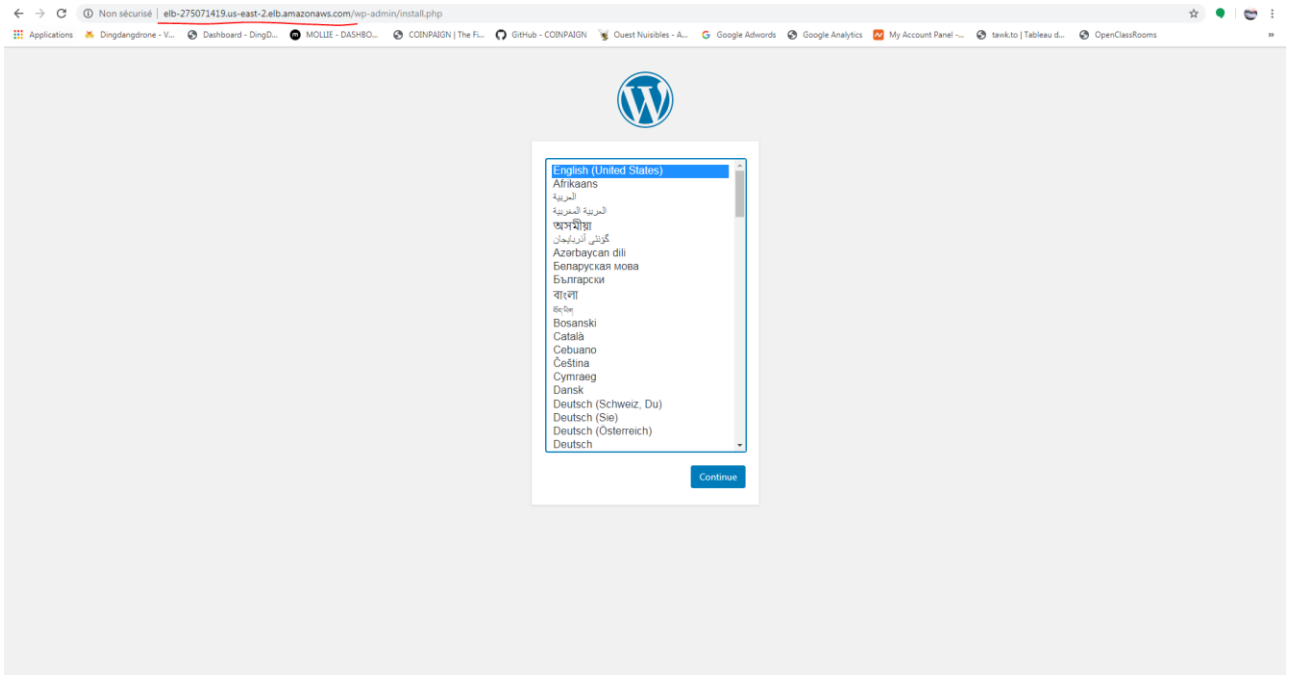


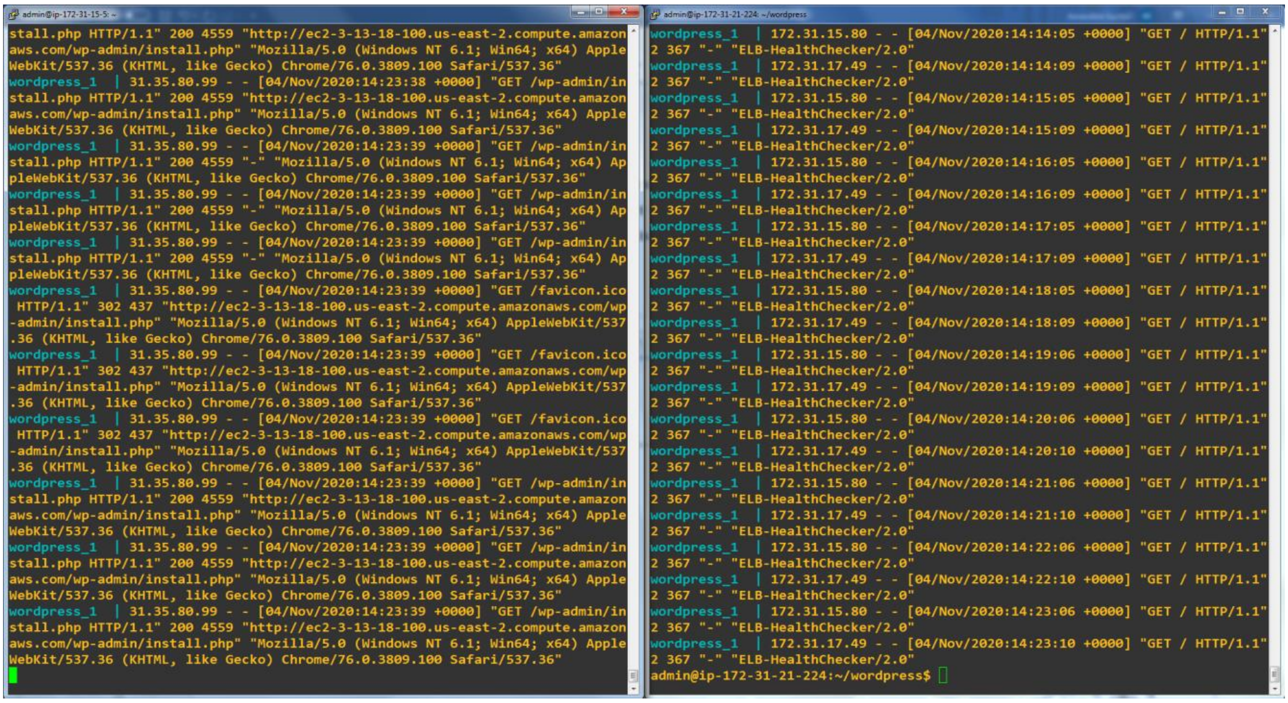






Vérification du fonctionnement ELB avec algorithme du tourniquet & permanence de 4H/Session



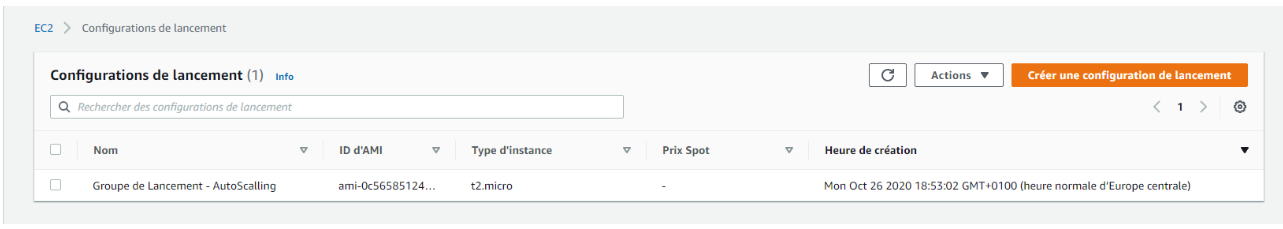


b) Configuration de la mise à l'échelle automatique (auto scalling)

Le but de la mise à l'échelle automatisé est de lancée un nombre x d'instance temporaire et de même image (AMI) afin de supporter les charges de Trafic importantes selon certains critères (dans notre cas lorsque la charge CPU>80%). Par définition, l'élasticité de notre infrastructure adaptera d'elle-même son nombres d'instances en zone multi a-z **selon une stratégie de mise à l'échelle que nous allons configurer dans notre groupe autoscalling.**

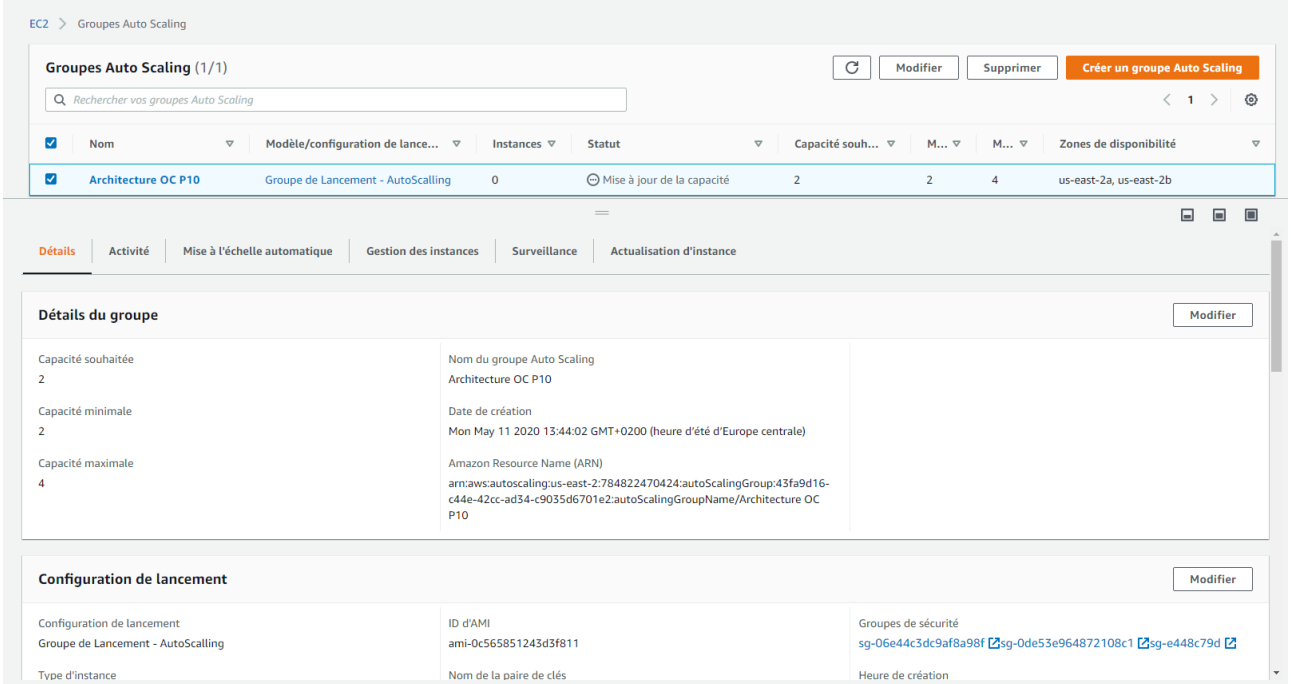
Tout d'abord nous devons configurer un groupe de lancement afin de paramétrer les différentes options de lancement des futures instances. Ce groupe de lancement est indépendant du groupe de mise à l'échelle car c'est lui qui est en charge de répondre à des critères comme : OU & QUOI, QUI & COMMENT (ces deux derniers points pourront être modifier dans le groupe autoscalling)

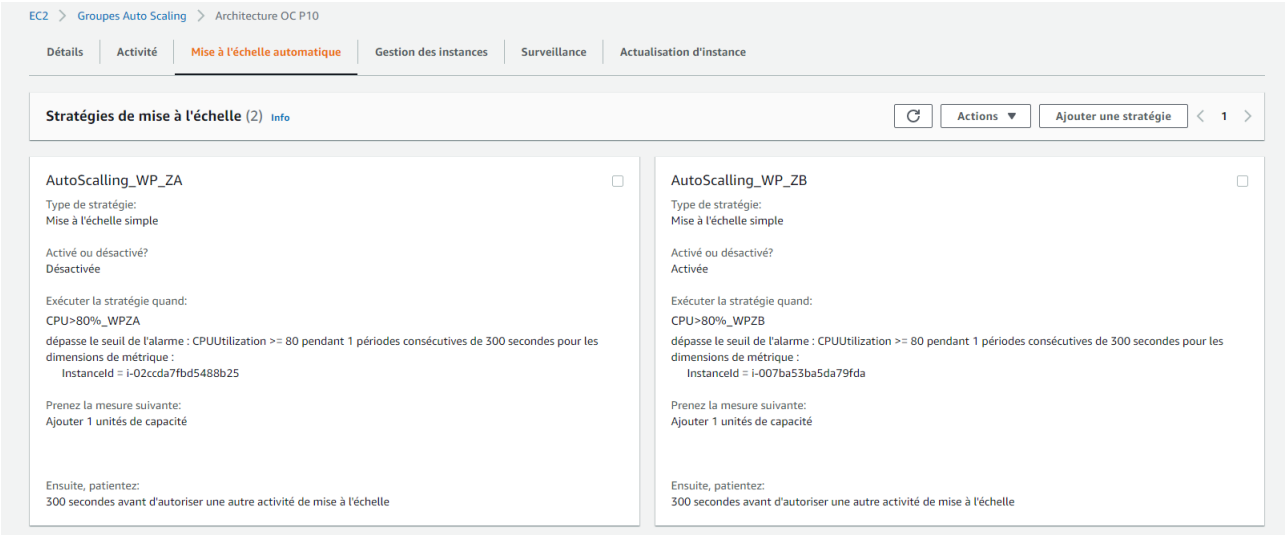
(Définition des Zones de lancement, groupe de sécurité, sans oublier l'image (AMI) à utiliser et le type d'instance (t2.micro dans notre cas))



Il nous faut donc crée un groupe de mise à l'échelle et associé nos instances EC2 de la zone A & B afin que ces dernières fassent partie intégrante de ce groupe. On définira également une stratégie

de mise à l'échelle selon des alarmes basées sur des métriques d'instance, la capacité souhaitée, minimal & maximal de notre déploiement automatisé.

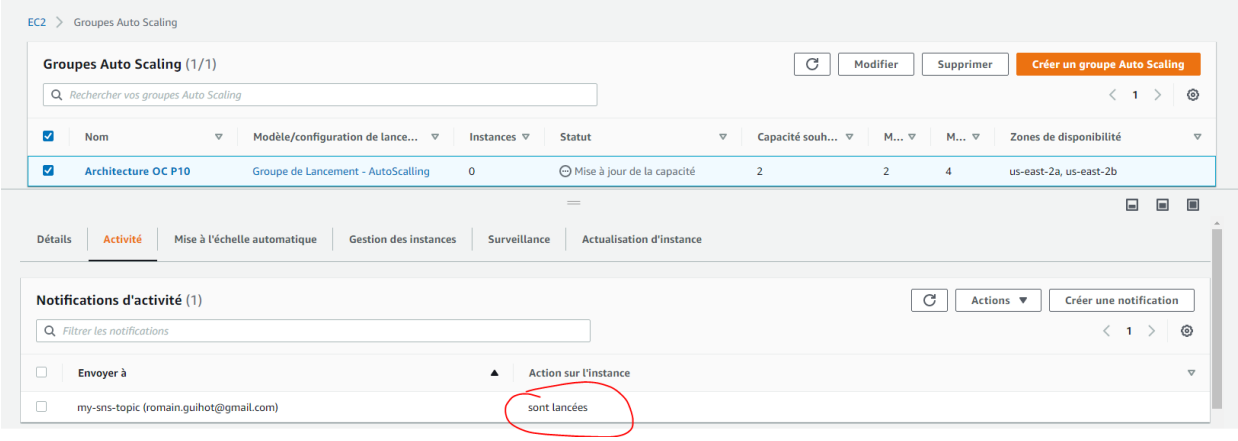




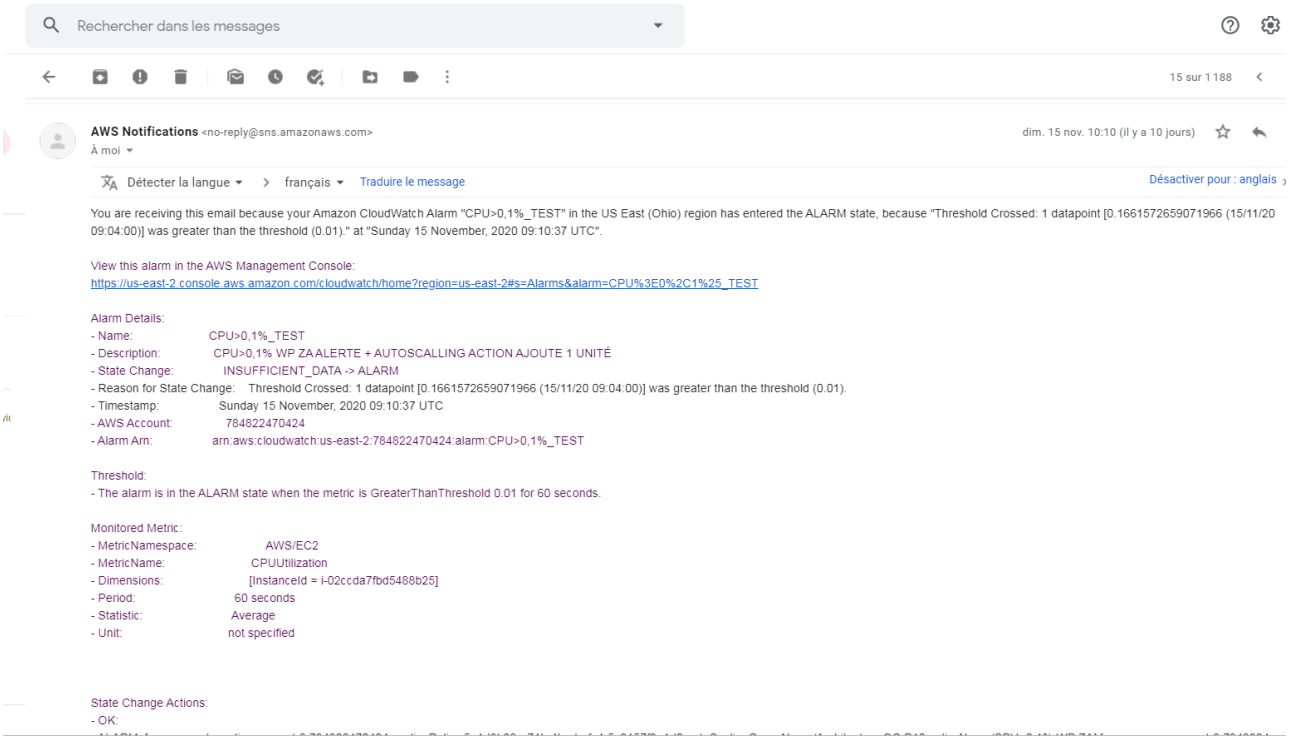
Enfin nous veillerons à être averti par notification email dès qu'une machine est lancée par effet de mise à l'échelle selon notre stratégie de CPU>80%.

Ces stratégies nécessitent la création d'alarmes « cloudwatch » avec action. Cette prise de mesure ou action est activé dès lors que le seuil de l'alarme est dépassé afin d'ajouter une unité de capacité supplémentaire à notre groupe de mise à l'échelle.

C'est à dire dimensionner de façon automatisée et donc « scaller » notre infrastructure (élastique) pour répondre aux besoins croissants d'une application qui fonctionne et qui obtient toujours plus de trafic, ou simplement des « pics » à des moments clés.

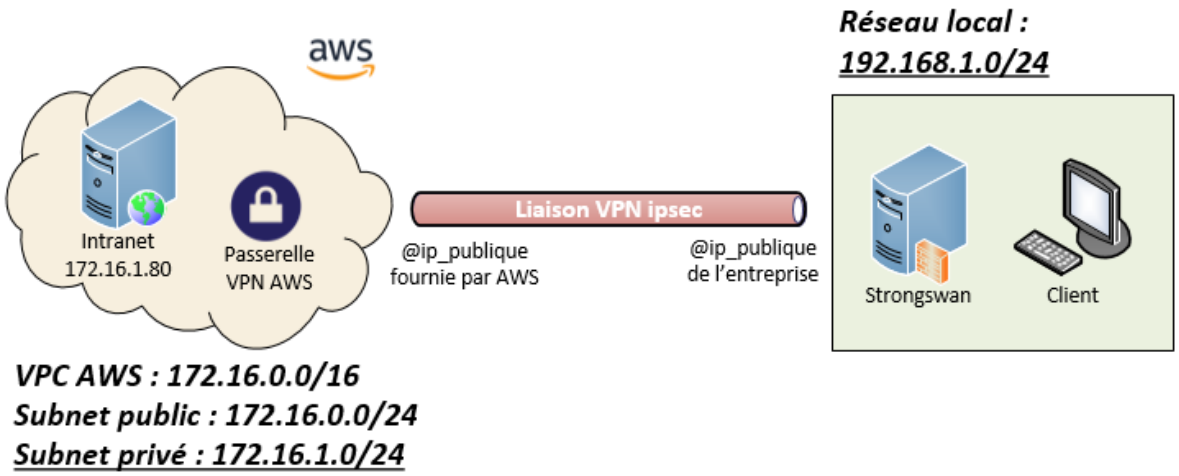


Un test a été réalisé avec la création d'une stratégie pour CPU>0,1% de manière à vérifier la scalabilité, et les notifications 😊



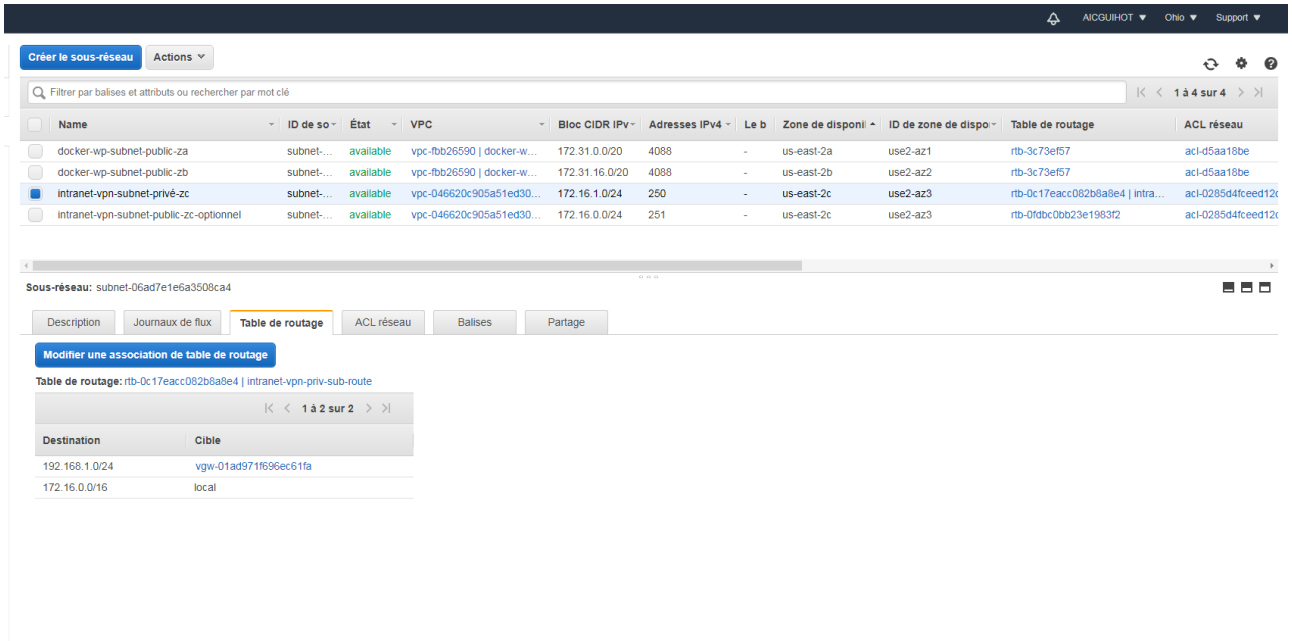
V - VPN site à site & Intranet isolé :

Le but étant de faire communiquer notre réseau d'entreprise locale (simulé par des VM de distribution debian sur mon réseau privé) à l'intranet hébergé chez aws dans un sous réseau privé isolé et inaccessible autrement que par le tunnel VPN.



Coté AWS, il est indispensable de créer un sous réseau privée et un sous réseau public qui sera utilisé uniquement afin de sortir sur internet provisoirement pour installer les packets d'un frontal web (apache, php : service httpd)

On pensera également à créer un groupe de sécurité pour n'autoriser que notre réseau privée d'entreprise à communiquer et définir les routes associées.



Comme pour AWS, notre serveur VPN local de distribution debian devra être en mode routeur.

En editant le fichier /etc/sysctl.conf

```
net.ipv4.ip_forward = 1
```

Explication de l'installation de Strongswan sur notre serveur VPN local avec la commande suivante :

```
apt-get install strongswan -y
```

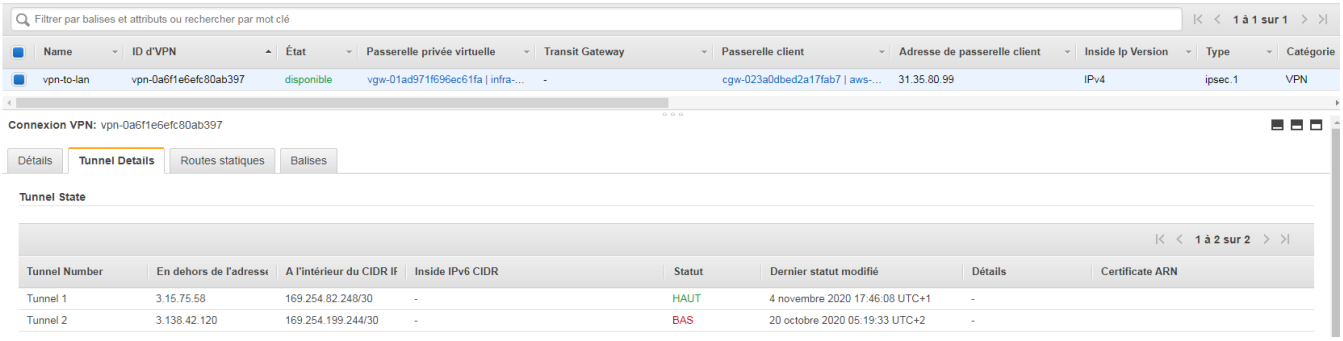
Ensuite on va préparer la configuration de la future liaison VPN. En éditant le fichier /etc/ipsec.conf

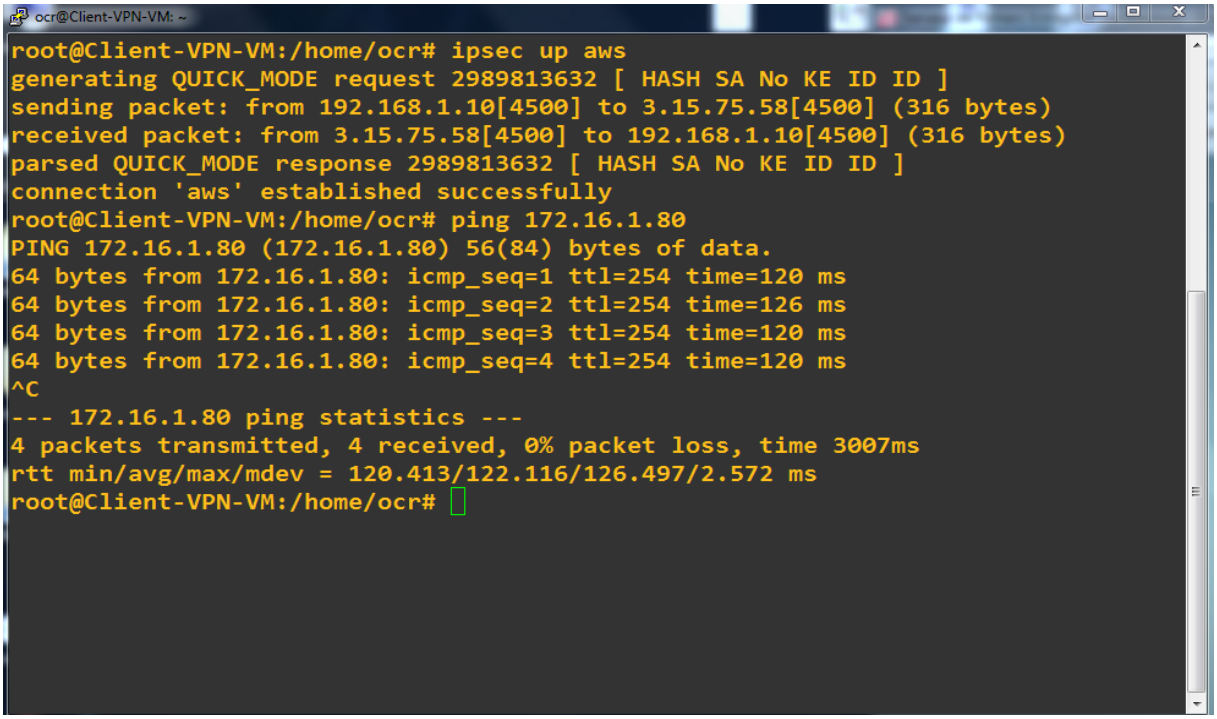
```
conn aws
        type=tunnel
        authby=secret
        left=192.168.1.10  # Addresse IP Du serveur VPN Local
        leftid=31.35.80.99 # Addresse Public Local Entreprise ( Fournit par mon FAI )
        right=3.15.75.58 # Addresse Public AWS
        leftauth=psk
        rightauth=psk
        keyexchange=ikev1
        ike=aes128-sha1-modp1024
        ikelifetime=8h
        esp=aes128-sha1-modp1024
        lifetime=1h
        keyingtries=%forever
        leftsubnet=192.168.1.0/24 # Réseau Privé Local Entreprise
        rightsubnet=172.16.1.0/24 # Réseau Privé AWS Zone C
        dpddelay=10s
        dpdtimeout=30s
        dpdaction=restart
        auto=start
```

On edite également le fichier /etc/ipsec.secrets afin de procéder à l'échange de clé entre nos deux points de terminaison.

```
31.35.80.99 3.15.75.58 : PSK "NBr3REZUHoC8pgjooi6C0VNoHuToX_Oq"
```

Après un redémarrage du vpn client/serveur notre tunnel vpn est montée et nous pouvons le vérifier des deux côtés de notre infrastructure (nous constatons également que nos routes & groupes de sécurité fonctionne correctement puisque depuis le routeur vpn local nous accédons aux travers du tunnel à notre intranet en 172.16.1.80 :



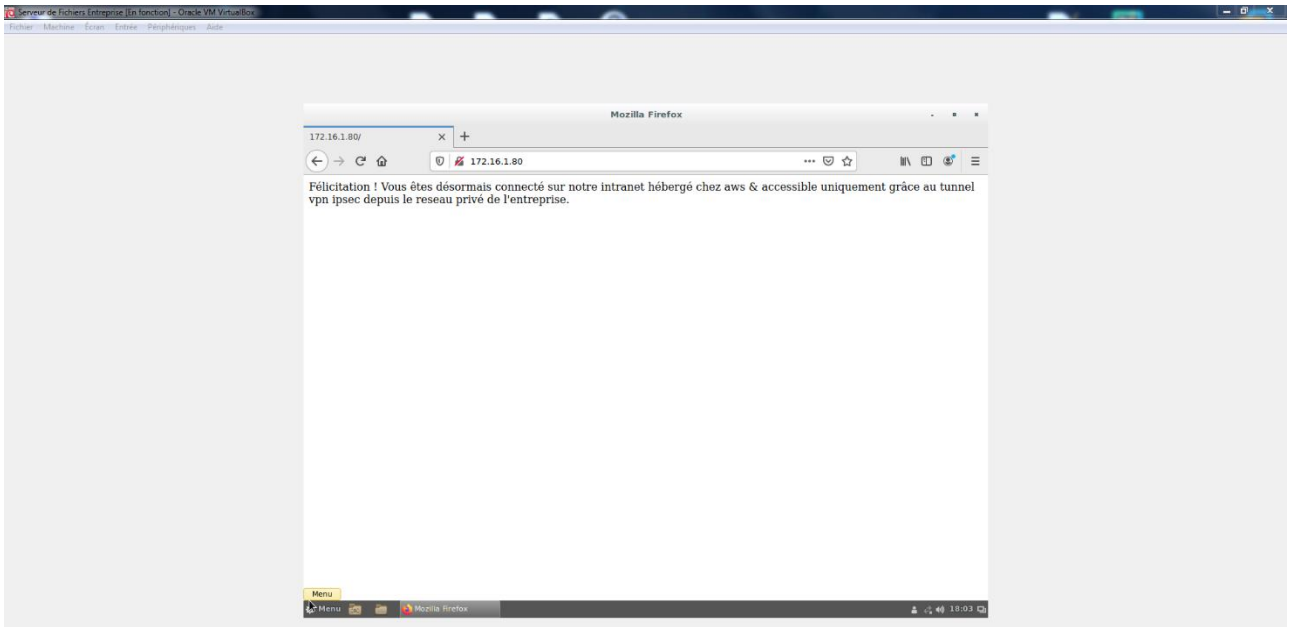


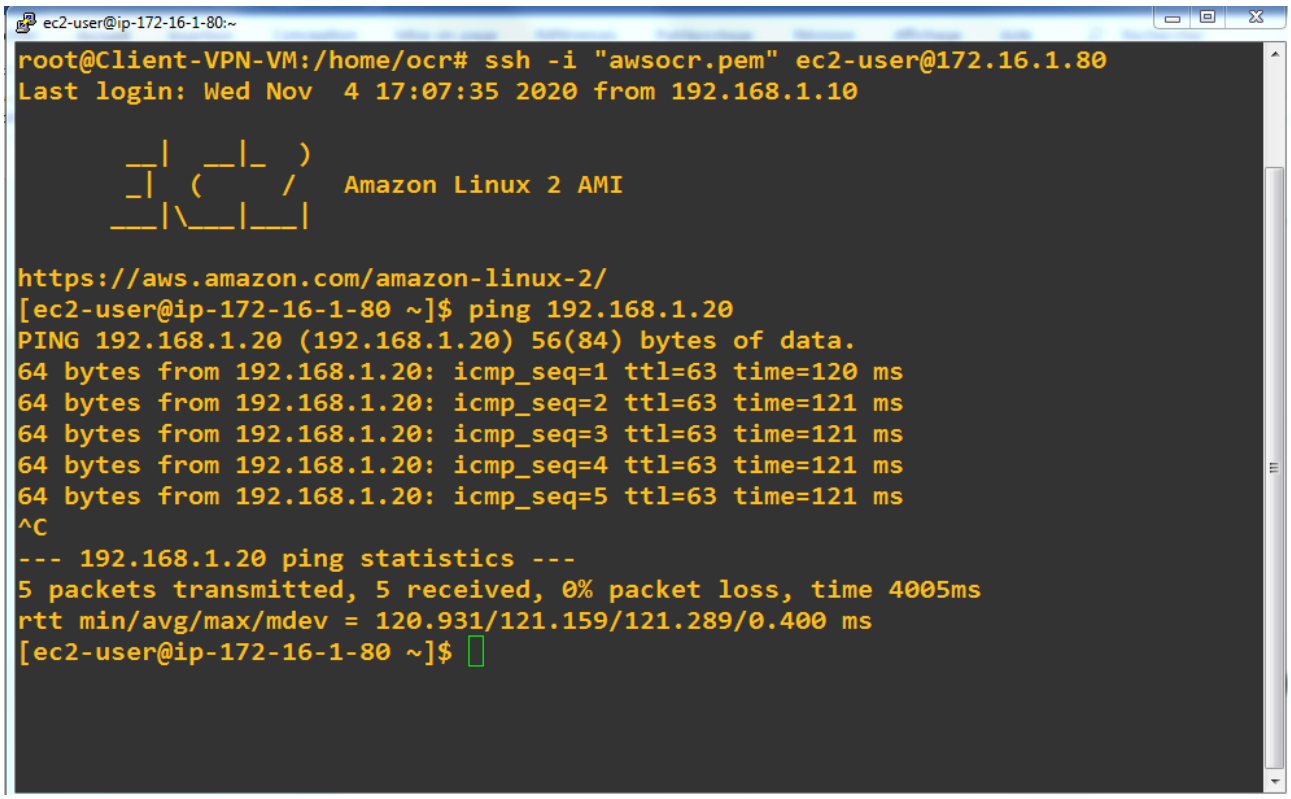
Il nous faudra bien sur définir à notre serveur de fichier local (192.168.1.20) comme passerelle par défault l'addresse ip de notre serveur vpn local (192.168.1.10)

De ce fait nos postes clients ou tout autre serveur d'entreprises pourront traverser le tunnel VPN à condition d'avoir toujours comme route par défault le serveur VPN local en mode routeur.

```
PING 172.16.1.80 (172.16.1.80) 56(84) bytes of data.
From 192.168.1.10: icmp_seq=1 Redirect Host(New nexthop: 192.168.1.254)
64 bytes from 172.16.1.80: icmp_seq=1 ttl=253 time=120 ms
From 192.168.1.10: icmp_seq=2 Redirect Host(New nexthop: 192.168.1.254)
64 bytes from 172.16.1.80: icmp_seq=2 ttl=253 time=120 ms
From 192.168.1.10: icmp_seq=3 Redirect Host(New nexthop: 192.168.1.254)
64 bytes from 172.16.1.80: icmp_seq=3 ttl=253 time=120 ms
From 192.168.1.10: icmp_seq=4 Redirect Host(New nexthop: 192.168.1.254)
64 bytes from 172.16.1.80: icmp_seq=4 ttl=253 time=121 ms
From 192.168.1.10: icmp_seq=5 Redirect Host(New nexthop: 192.168.1.254)
64 bytes from 172.16.1.80: icmp_seq=5 ttl=253 time=121 ms
From 192.168.1.10: icmp_seq=6 Redirect Host(New nexthop: 192.168.1.254)
64 bytes from 172.16.1.80: icmp_seq=6 ttl=253 time=121 ms
^C
--- 172.16.1.80 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5007ms
rtt min/avg/max/mdev = 120.524/120.993/121.355/0.306 ms
root@Client-VPN-VM:/home/ocr# route -n
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic Metric Ref    Use Iface
0.0.0.0         192.168.1.10    0.0.0.0         UG    0      0        0 enp0s3
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 enp0s3
```

Il nous reste à vérifier que l'on accède via le service web de notre serveur de fichier à la page intranet de notre serveur sur le réseau privé aws au travers du tunnel ipsec 😊





VI - Script Cloud Formation :

La force de ce projet est également la capacité à concevoir entièrement son architecture grâce à la programmation avec la création de modèle aussi appelé « pile ».

Ces piles au format json ou yaml regroupes l'ensembles des ressources et dépendances nécessaires à la création complète d'une infrastructure répondant à divers besoins.

Coinbase utilise par exemple la puissance de L'IaC AWS pour centraliser, maitriser, tester, concevoir et déployer en production toutes les ressources de l'infrastructure hébergeant l'application de portefeuilles numériques basé sur les technologies de la blockchain.

- Veuillez consulter le fichier .yaml afin de prendre connaissance du modèle complet pour la création totale de l'infrastructure du projet.

```yaml
Description: "Architecture Projet 10 OCR"
Resources:
    EC2VPC:
        Type: "AWS::EC2::VPC"
        Properties:
            CidrBlock: "172.31.0.0/16"
            EnableDnsSupport: true
            EnableDnsHostnames: true
            InstanceTenancy: "default"
            Tags:
              -
                Key: "Name"
                Value: "docker-wp-vpc"

    EC2VPC2:
        Type: "AWS::EC2::VPC"
        Properties:
            CidrBlock: "172.16.0.0/16"
            EnableDnsSupport: true
            EnableDnsHostnames: false
            InstanceTenancy: "default"
            Tags:
              -
                Key: "Name"
                Value: "intranet-vpn-vpc"

    EC2Subnet:
        Type: "AWS::EC2::Subnet"
        Properties:
            AvailabilityZone: !GetAtt EC2Instance3.AvailabilityZone
            CidrBlock: "172.31.0.0/20"
            VpcId: !Ref EC2VPC
            MapPublicIpOnLaunch: true
            Tags:
              -
                Key: "Name"
                Value: "docker-wp-subnet-public-za"

    EC2Subnet2:
        Type: "AWS::EC2::Subnet"
        Properties:
            AvailabilityZone: !GetAtt EC2Instance.AvailabilityZone
            CidrBlock: "172.31.16.0/20"
            VpcId: !Ref EC2VPC
            MapPublicIpOnLaunch: true
            Tags:
              -
                Key: "Name"
                Value: "docker-wp-subnet-public-zb"
```