Lab — Générer le schéma de topologie VPC AWS automatiquement

Sommaire


```
C:\temp>wsl
henri@Henri:/mnt/c/temp$ pwd
/mnt/c/temp
henri@Henri:/mnt/c/temp$ dir
CLF_C02_Flashcards.csv        basic-rds.yml  response.json     trust-policy.json  vpc_topology.png
aws_vpc_map.png               http-api.yaml  s3-bucket.yml     venv
aws_vpc_network_topology.png  new\ 2.txt     sam-http-api-lab  vpc_map.py
```

Objectif

Scanner ton compte AWS et produire une image qui montre :

- VPC
- Subnets publics / privés
- IGW
- NAT
- Route tables (liaisons réelles)

Prérequis

- WSL (Ubuntu) fonctionnel
- Python 3.10+
- pip
- Accès AWS configuré (`aws configure`)
- Graphviz installé (indispensable pour Diagrams)

---

1 Installer Graphviz

```
sudo apt update
sudo apt install graphviz -y
dot -V
```

---

2 Créer le dossier du projet

Aller dans WSL :

#wsl

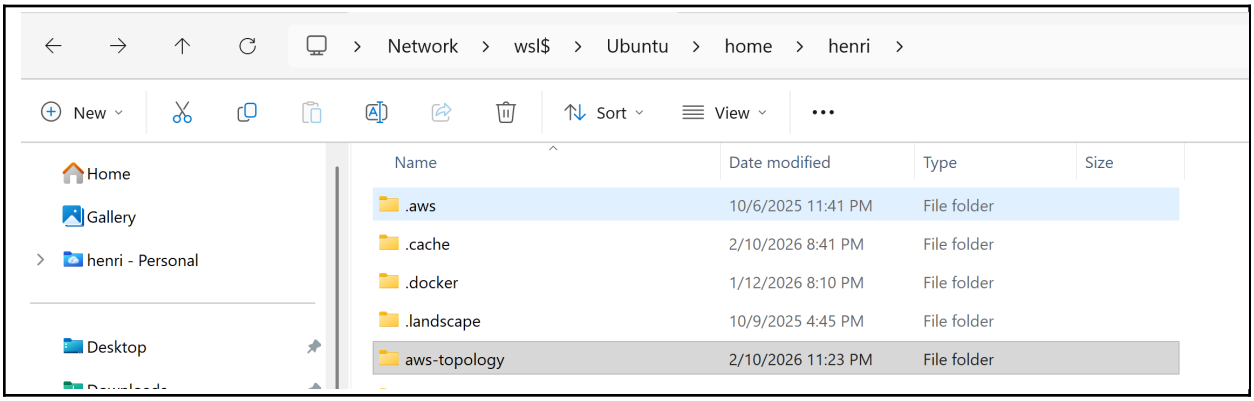
```
C:\temp>wsl
henri@Henri:/mnt/c/temp$ pwd
/mnt/c/temp
henri@Henri:/mnt/c/temp$ dir
CLF_C02_Flashcards.csv        basic-rds.yml  response.json     trust-policy.json  vpc_topology.png
aws_vpc_map.png               http-api.yaml  s3-bucket.yml     venv
aws_vpc_network_topology.png  new\ 2.txt     sam-http-api-lab  vpc_map.py
```

cd ~

```
henri@Henri:/mnt/c/temp$
henri@Henri:/mnt/c/temp$ cd ~
henri@Henri:~$
henri@Henri:~$
henri@Henri:~$ pwd
/home/henri
henri@Henri:~$
```

```
mkdir aws-topology
cd aws-topology
```

quand on fait : `cd ~`
cela nous amene sur le reprtoire racine de WSL : [\\wsl$\Ubuntu\home\henri](\\wsl$\Ubuntu\home\henri)



---

3 Créer et activer le venv

```
cd aws-topology
python3 -m venv venv
source venv/bin/activate
```

```
henri@Henri:~/aws-topology$
henri@Henri:~/aws-topology$ python3 -m venv venv
henri@Henri:~/aws-topology$ source venv/bin/activate
(venv) henri@Henri:~/aws-topology$
(venv) henri@Henri:~/aws-topology$
```

---

4 Fichier requirements.txt

Créer le fichier : requirements.txt

Dans le répertoire aws-topology

Mettre dans le fichier , les instructions ci-dessous.

boto3==1.34.131
diagrams==0.23.4
graphviz==0.20.3

Installer le fichier :

```
pip install -r requirements.txt
```

```
(venv) henri@Henri:~/aws-topology$
(venv) henri@Henri:~/aws-topology$ pip install -r requirements.txt
Collecting boto3==1.34.131 (from -r requirements.txt (line 1))
  Downloading boto3-1.34.131-py3-none-any.whl.metadata (6.6 kB)
Collecting diagrams==0.23.4 (from -r requirements.txt (line 2))
  Downloading diagrams-0.23.4-py3-none-any.whl.metadata (7.0 kB)
Collecting graphviz==0.20.3 (from -r requirements.txt (line 3))
  Downloading graphviz-0.20.3-py3-none-any.whl.metadata (12 kB)
Collecting botocore<1.35.0,>=1.34.131 (from boto3==1.34.131->-r requirements.txt (line 1))
  Downloading botocore-1.34.162-py3-none-any.whl.metadata (5.7 kB)
Collecting jmespath<2.0.0,>=0.7.1 (from boto3==1.34.131->-r requirements.txt (line 1))
  Downloading jmespath-1.1.0-py3-none-any.whl.metadata (7.6 kB)
Collecting s3transfer<0.11.0,>=0.10.0 (from boto3==1.34.131->-r requirements.txt (line 1))
  Downloading s3transfer-0.10.4-py3-none-any.whl.metadata (1.7 kB)
Collecting jinja2<4.0,>=2.10 (from diagrams==0.23.4->-r requirements.txt (line 2))
  Downloading jinja2-3.1.6-py3-none-any.whl.metadata (2.9 kB)
```

---

5 Configurer AWS (si pas déjà fait)

```
aws configure
```

Profil avec droits lecture (ReadOnly) suffit.

---

6 Créer le script topology.py

Colle **exactement** ceci dans le script :

```
import boto3
from diagrams import Diagram, Cluster
from diagrams.aws.network import VPC, PublicSubnet, PrivateSubnet,
InternetGateway, NATGateway, RouteTable

region = "ca-central-1"
ec2 = boto3.client("ec2", region_name=region)

vpcs = ec2.describe_vpcs()["Vpcs"]
subnets = ec2.describe_subnets()["Subnets"]
igws = ec2.describe_internet_gateways()["InternetGateways"]
natgws = ec2.describe_nat_gateways()["NatGateways"]
rts = ec2.describe_route_tables()["RouteTables"]

with Diagram("AWS VPC Network Topology", show=False):

    for vpc in vpcs:
        vpc_node = VPC(vpc["CidrBlock"])

        igw_node = None
        for igw in igws:
            for att in igw.get("Attachments", []):
                if att["VpcId"] == vpc["VpcId"]:
                    igw_node = InternetGateway("IGW")
                    igw_node >> vpc_node

        with Cluster(f"VPC {vpc['CidrBlock']}"):

            subnet_nodes = {}
            private_subnets = []

            for subnet in subnets:
                if subnet["VpcId"] != vpc["VpcId"]:
                    continue

                cidr = subnet["CidrBlock"]
                is_public = False
                used_rt = None

                for rt in rts:
                    for assoc in rt.get("Associations", []):
                        if assoc.get("SubnetId") == subnet["SubnetId"]:
                            used_rt = rt
                            for route in rt.get("Routes", []):
                                if route.get("GatewayId",
"").startswith("igw"):
                                    is_public = True

                if is_public:
                    sn = PublicSubnet(cidr)
                else:
                    sn = PrivateSubnet(cidr)
                    private_subnets.append((sn, used_rt))
```

```
                vpc_node >> sn
                subnet_nodes[subnet["SubnetId"]] = sn

            for nat in natgws:
                if nat["VpcId"] != vpc["VpcId"]:
                    continue

                nat_node = NATGateway("NAT")

                for sn, rt in private_subnets:
                    if not rt:
                        continue
                    for route in rt.get("Routes", []):
                        if route.get("NatGatewayId") == nat["NatGatewayId"]:
                            rt_node = RouteTable("RT")
                            sn >> rt_node >> nat_node
```

7 Exécuter le script

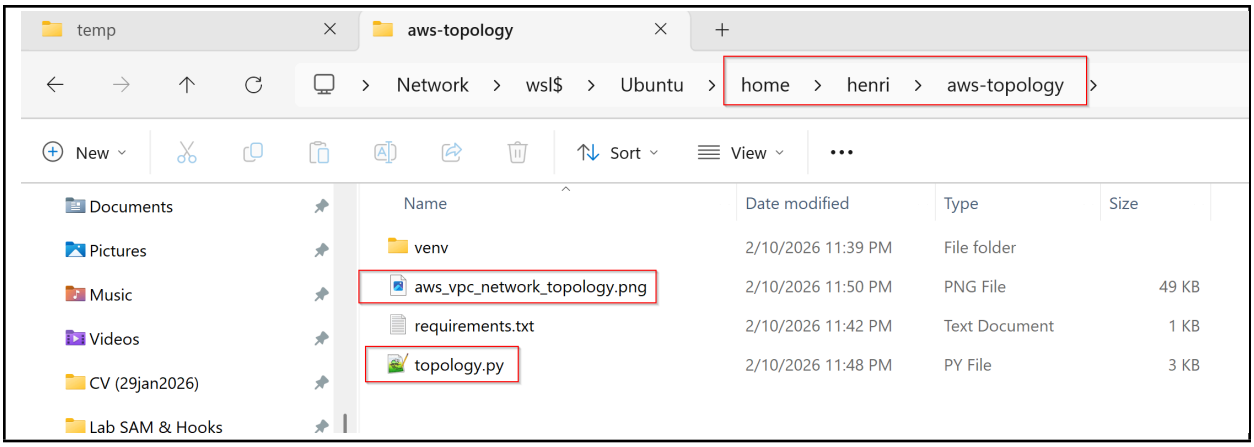
```
python topology.py
```

Un fichier PNG est généré dans le dossier.

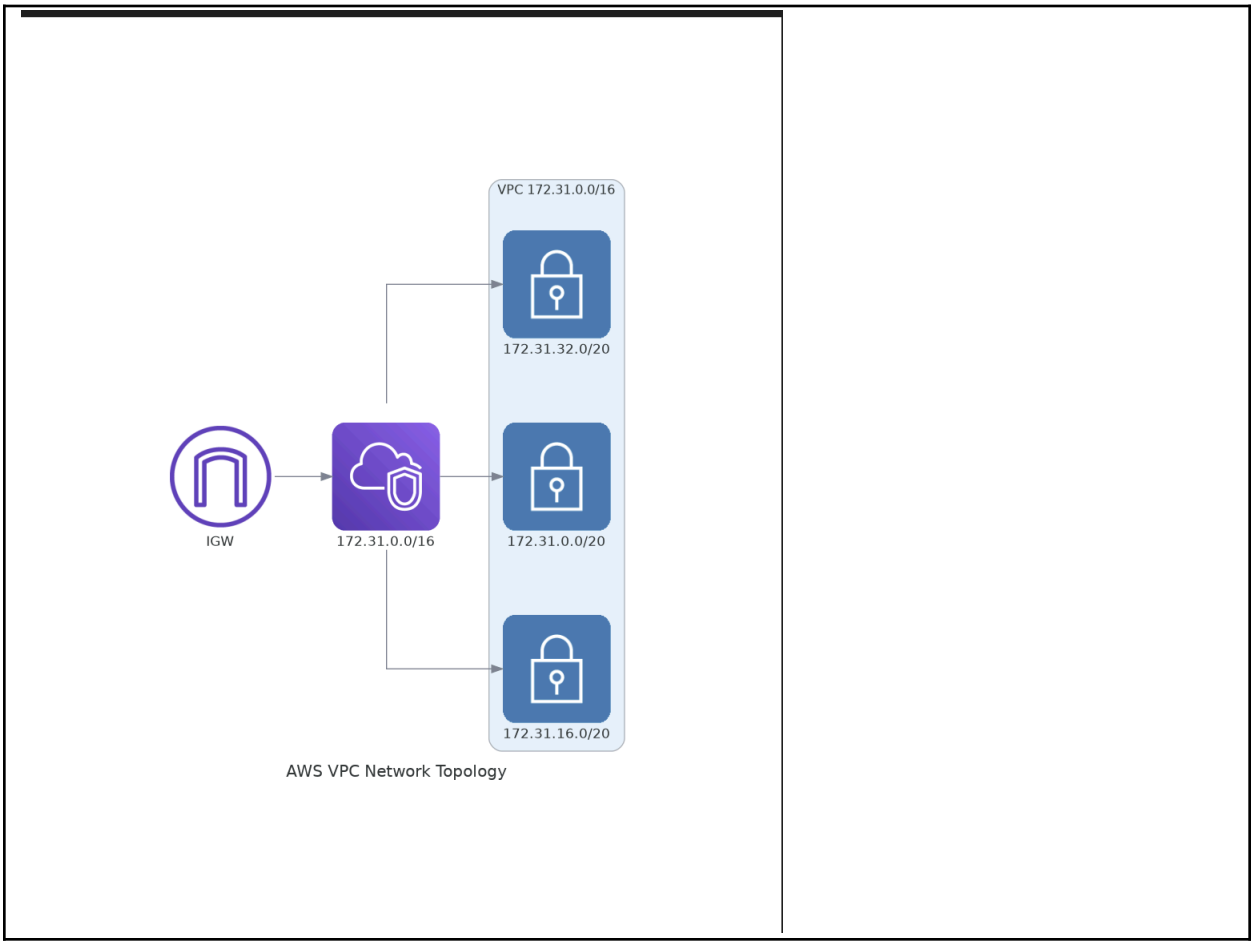
```
(venv) henri@Henri:~$ cd aws-topology
(venv) henri@Henri:~/aws-topology$
(venv) henri@Henri:~/aws-topology$ python topology.py
(venv) henri@Henri:~/aws-topology$
(venv) henri@Henri:~/aws-topology$ dir
aws_vpc_network_topology.png  requirements.txt  topology.py  venv
(venv) henri@Henri:~/aws-topology$
```

Au niveau de l'Explorateur Windows, on peut voir dans l'image ci-dessous, le répertoire racine de WSL ainsi que l'arborescence qui en découle.

On note que la commande python topology.py a généré un fichier image , de type .png . ici dans notre exemple le fichier s'intitule : aws_vpc_network_topology.py



Le fichier contient le topologie du réseau AWS.



---

8 Ce que fait réellement le script

Il **déduit** :

- Public vs privé via les routes vers IGW
- Quels subnets utilisent quel NAT via les routes
- Les vraies liaisons réseau AWS

Ce n'est **pas** un dessin statique. C'est un scan réel.

---

9 Comment refaire le graphique dans 2 jours / 2 mois

Rien à réinstaller.

```
cd ~/aws-topology
source venv/bin/activate
python topology.py
```

Le PNG reflète **l'état actuel** de ton AWS.

---

10 Dépannage rapide

Erreur	Cause	Fix
`dot not found`	Graphviz absent	`sudo apt install graphviz`
`Unable to locate credentials`	AWS non configuré	`aws configure`
`AccessDenied`	Pas ReadOnly	Attacher `ReadOnlyAccess`

---

11 Étape suivante

11.1 Ajouter au schéma :

- EC2 / ECS dans subnets
- ALB publics
- RDS privés
- VPC Endpoints

Pour obtenir **la topologie complète applicative**, pas seulement réseau, on ajoute au script python :

```
from diagrams.aws.compute import EC2, ECS
from diagrams.aws.network import ELB, VPCEndpoint
from diagrams.aws.database import RDS
```

11.2 script python :

```
import boto3
from diagrams import Diagram, Cluster
from diagrams.aws.network import VPC, PublicSubnet, PrivateSubnet, InternetGateway, NATGateway,
RouteTable, ELB, VPCEndpoint
from diagrams.aws.compute import EC2, ECS
from diagrams.aws.database import RDS

region = "ca-central-1"

ec2 = boto3.client("ec2", region_name=region)
elbv2 = boto3.client("elbv2", region_name=region)
rds = boto3.client("rds", region_name=region)
ecs = boto3.client("ecs", region_name=region)

vpcs = ec2.describe_vpcs()["Vpcs"]
subnets = ec2.describe_subnets()["Subnets"]
igws = ec2.describe_internet_gateways()["InternetGateways"]
natgws = ec2.describe_nat_gateways()["NatGateways"]
rts = ec2.describe_route_tables()["RouteTables"]
instances = ec2.describe_instances()["Reservations"]
lbs = elbv2.describe_load_balancers()["LoadBalancers"]
dbs = rds.describe_db_instances()["DBInstances"]
endpoints = ec2.describe_vpc_endpoints()["VpcEndpoints"]

with Diagram("AWS Full Topology", show=False):

    for vpc in vpcs:
        vpc_node = VPC(vpc["CidrBlock"])

        # IGW
        for igw in igws:
            for att in igw.get("Attachments", []):
                if att["VpcId"] == vpc["VpcId"]:
                    igw_node = InternetGateway("IGW")
                    igw_node >> vpc_node

        with Cluster(f"VPC {vpc['CidrBlock']}"):
```

```
        subnet_map = {}
        private_subnets = []

        # Subnets
        for subnet in subnets:
            if subnet["VpcId"] != vpc["VpcId"]:
                continue

            cidr = subnet["CidrBlock"]
            is_public = False
            used_rt = None

            for rt in rts:
                for assoc in rt.get("Associations", []):
                    if assoc.get("SubnetId") == subnet["SubnetId"]:
                        used_rt = rt
                        for route in rt.get("Routes", []):
                            if route.get("GatewayId", "").startswith("igw"):
                                is_public = True

            sn = PublicSubnet(cidr) if is_public else PrivateSubnet(cidr)
            vpc_node >> sn
            subnet_map[subnet["SubnetId"]] = sn

            if not is_public:
                private_subnets.append((sn, used_rt))

        # NAT
        for nat in natgws:
            if nat["VpcId"] != vpc["VpcId"]:
                continue

            nat_node = NATGateway("NAT")

            for sn, rt in private_subnets:
                if not rt:
                    continue
                for route in rt.get("Routes", []):
                    if route.get("NatGatewayId") == nat["NatGatewayId"]:
                        rt_node = RouteTable("RT")
                        sn >> rt_node >> nat_node

        # EC2
        for res in instances:
            for inst in res["Instances"]:
                sid = inst.get("SubnetId")
                if sid in subnet_map:
                    subnet_map[sid] >> EC2(inst["InstanceId"])
```

```
    # ALB
    for lb in lbs:
        for sid in lb.get("AvailabilityZones", []):
            subnet_id = sid.get("SubnetId")
            if subnet_id in subnet_map:
                subnet_map[subnet_id] >> ELB(lb["LoadBalancerName"])

    # RDS
    for db in dbs:
        for subnet in db["DBSubnetGroup"]["Subnets"]:
            sid = subnet["SubnetIdentifier"]
            if sid in subnet_map:
                subnet_map[sid] >> RDS(db["DBInstanceIdentifier"])

    # VPC Endpoints
    for ep in endpoints:
        for sid in ep.get("SubnetIds", []):
            if sid in subnet_map:
                subnet_map[sid] >> VPCEndpoint(ep["ServiceName"].split(".")[-1])
```

▶️ Exécution

```
cd ~/aws-topology
source venv/bin/activate
python topology.py
```

Dans le cadre de ce lab , le schémas ne montrera rien de plus que précédemment , car je n'ai pas de base RDS, ou de ALB, ou de VPC , du moins présentement. J'en installe, mais régulièrement je les enleve, afin de ne pas avoir de cout inutile ou de couts éventuels et afin de ne pas surcharger la topologie. Il s'agit d'une topologie de lab .

✅ Résultat

Le PNG montre maintenant :

- Réseau réel
- Compute réel (EC2/ECS)
- Load balancers
- Bases de données
- Endpoints privés

C’est **la photo exacte** de mon architecture AWS.