INF8102– Sécurité dans les environnements infonuagiques

TP4 : Infrastructure as Code Security

Soumis à :

Steph Yann TSAPI II

Soumis par :

Anis Ait-Kaci Ali – 2132752

Michlove Pierre - 2144722

Groupe : 02

Session Automne 2025

11 décembre 2025

1.

```
michlove@VivoBook-ASUSLaptop-TP420UA-TM420UA:~/TP4$ nano vpc.yaml
```

Figure 1: Création du fichier vpc.yaml

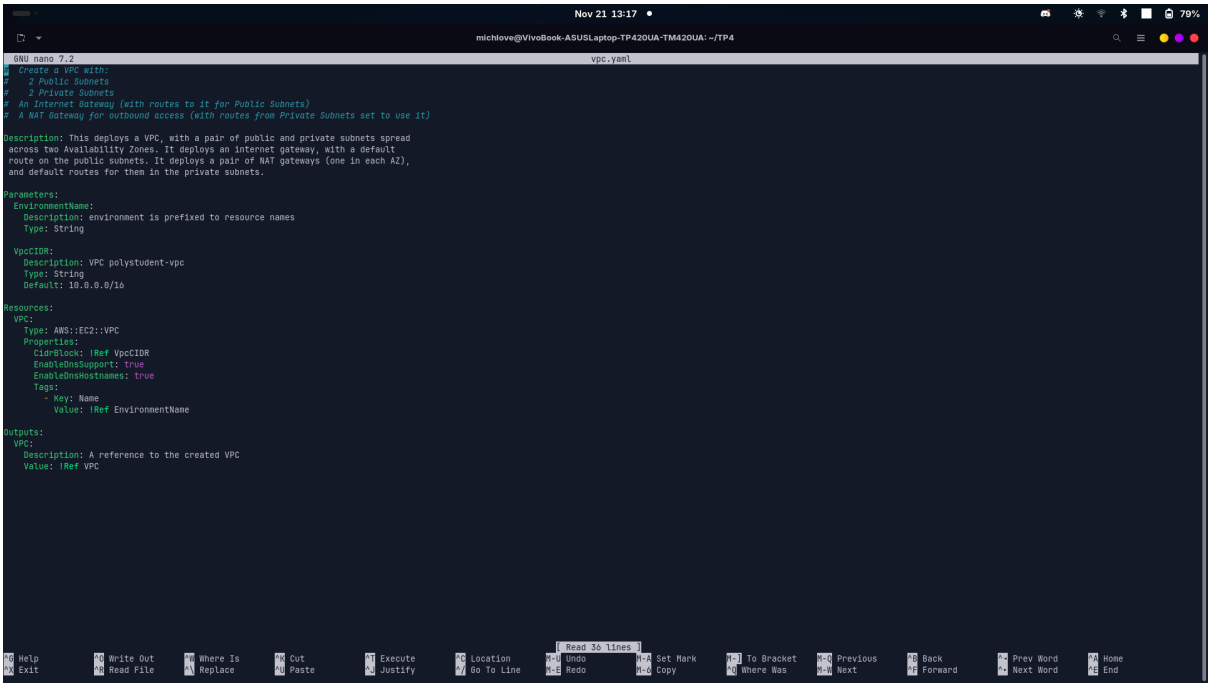


Figure 2: Ajout de la description, des paramètres et ressources dans le fichier vpc.yaml

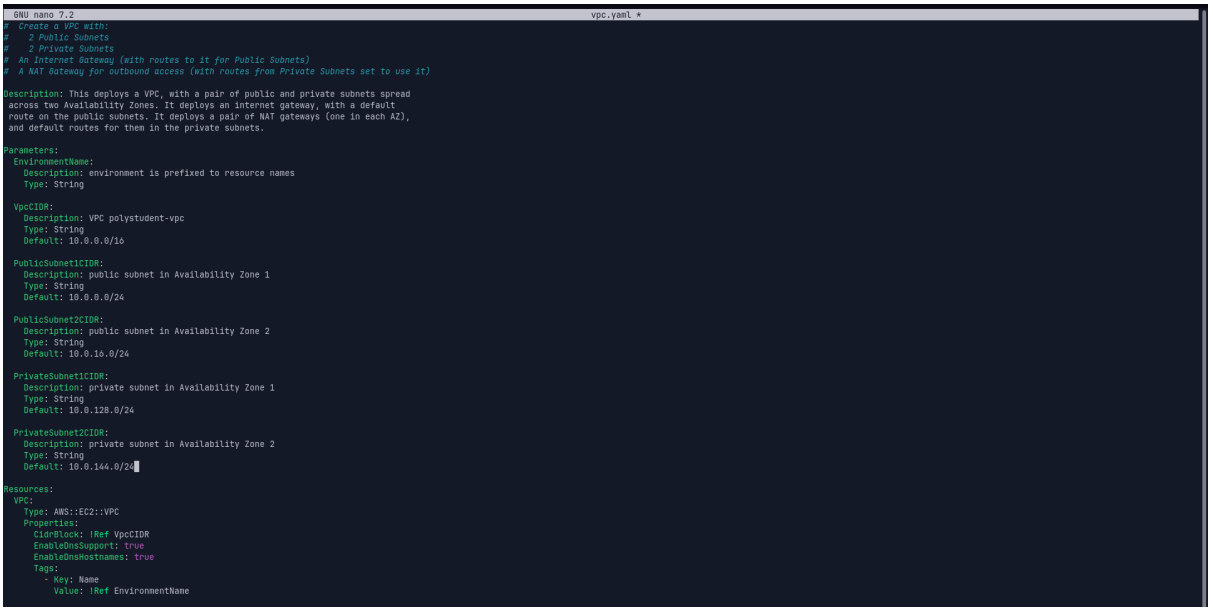


Figure 3: Ajout des paramètres supplémentaire pour la création de AZ1 et AZ2 dans le fichier vpc.yaml

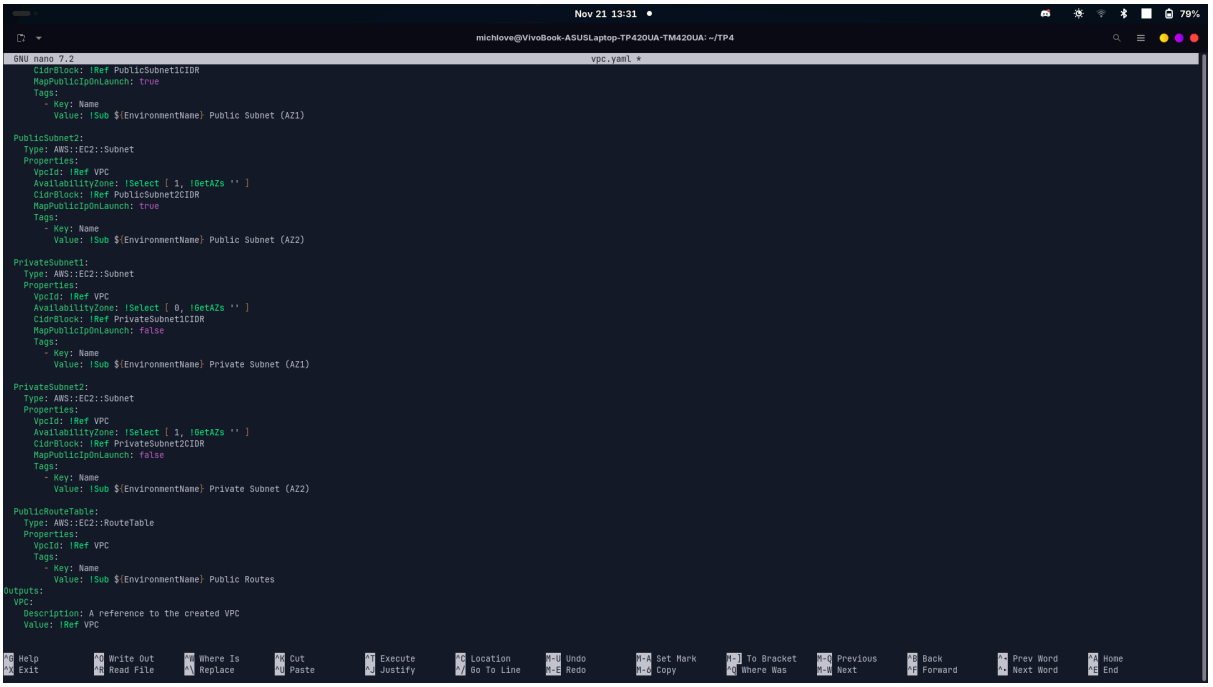


Figure 4: Ajout des ressources supplémentaires pour la création de AZ1 et AZ2 dans le fichier vpc.yaml

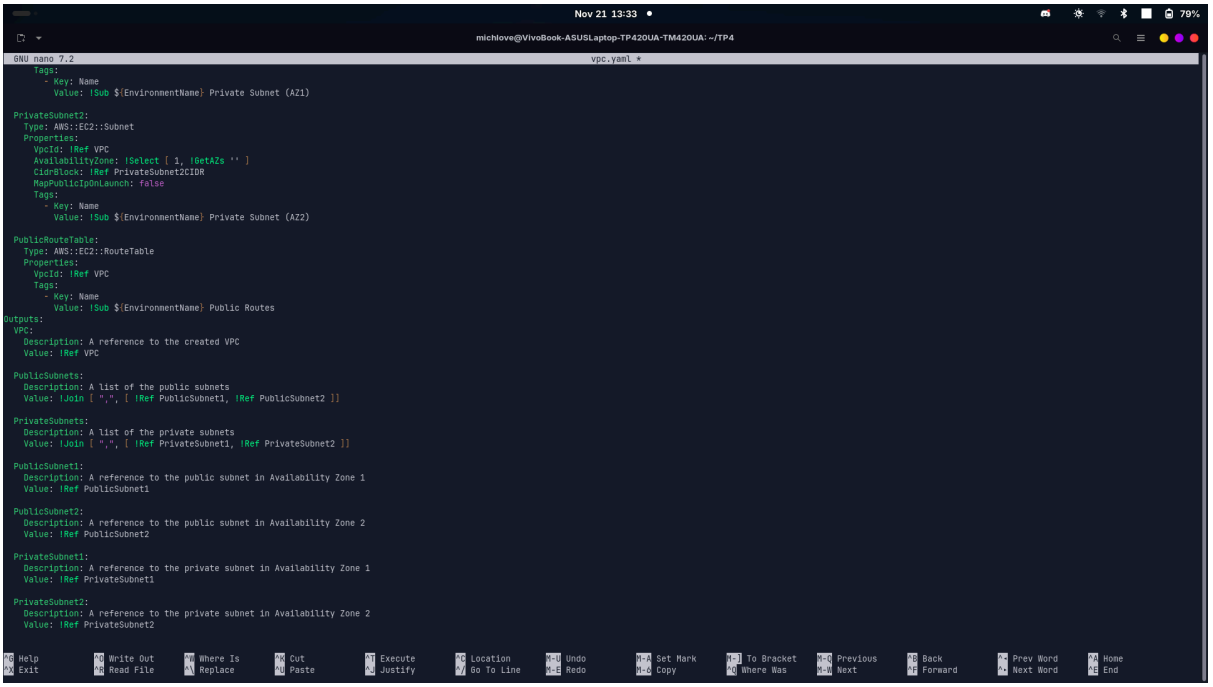


Figure 5: Ajout des configuration dans le outputs pour afficher les contenue dans le CloudFormation

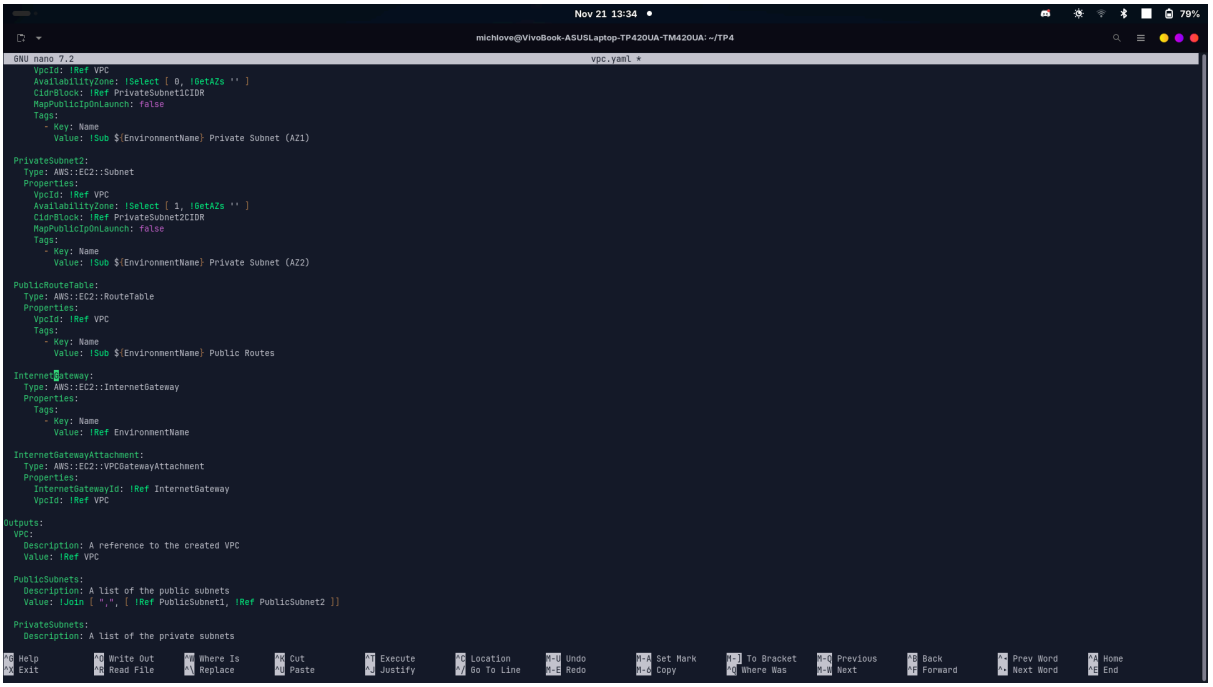


Figure 6: Ajout de l'internet gateway dans le fichier vpc.yaml

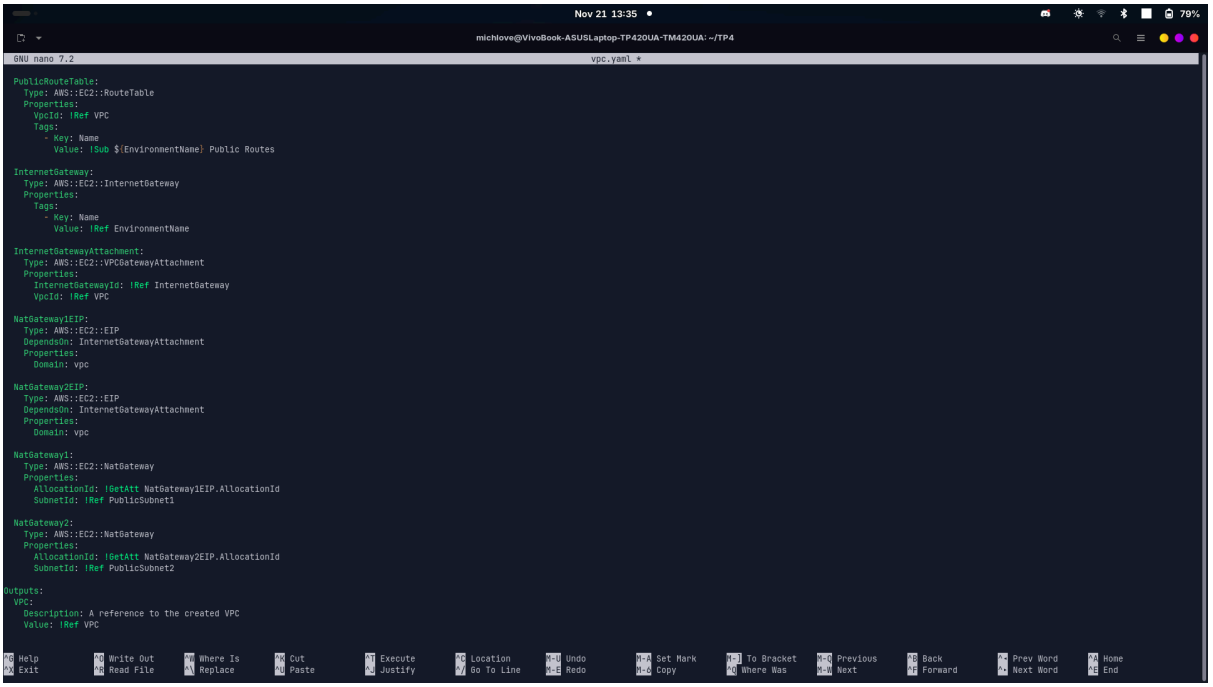


Figure 7: Ajout du NAT gateway dans le fichier vpc.yaml

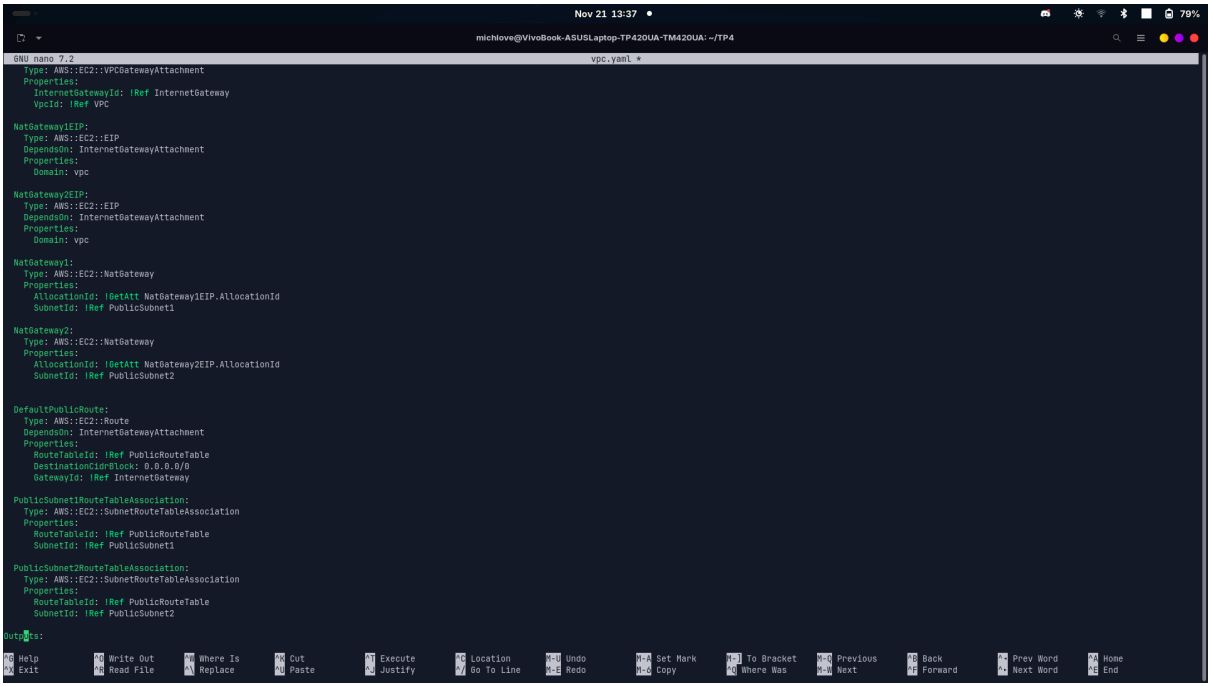


Figure 8: Création de la table de routage pour le subnet public dans le fichier vpc.yaml

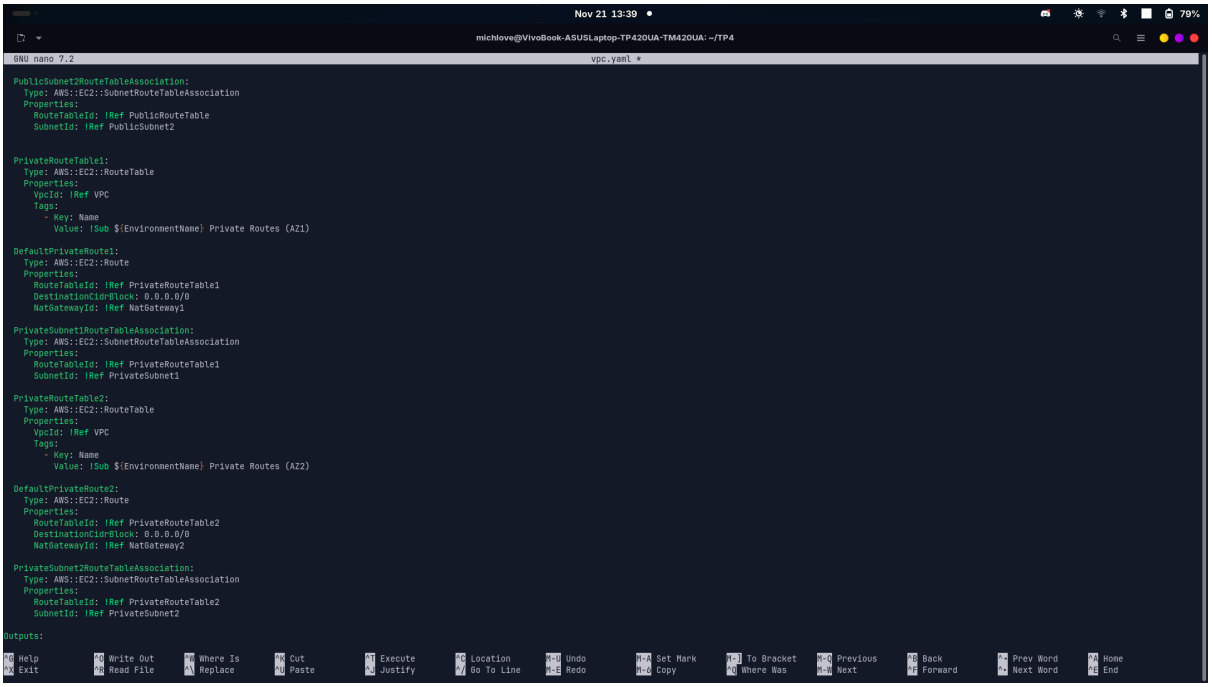


Figure 9: Création de la table de routage pour le subnet privé dans le fichier vpc.yaml

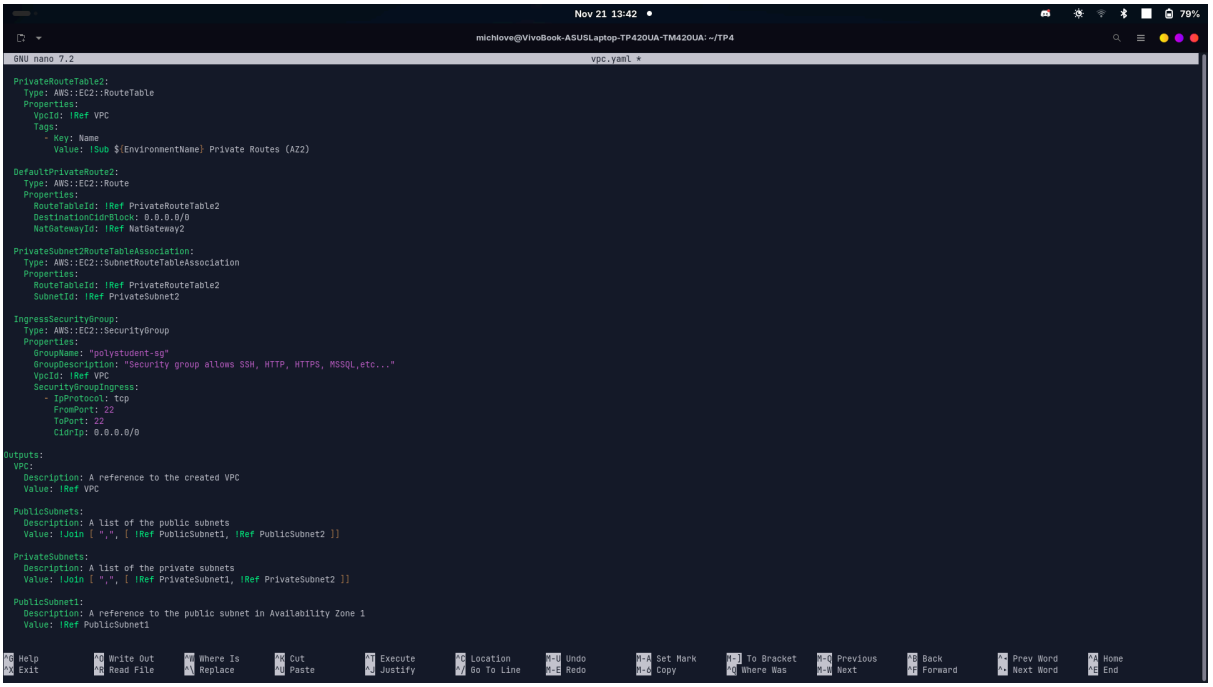


Figure 10: Création du groupe de sécurité dans le fichier vpc.yaml

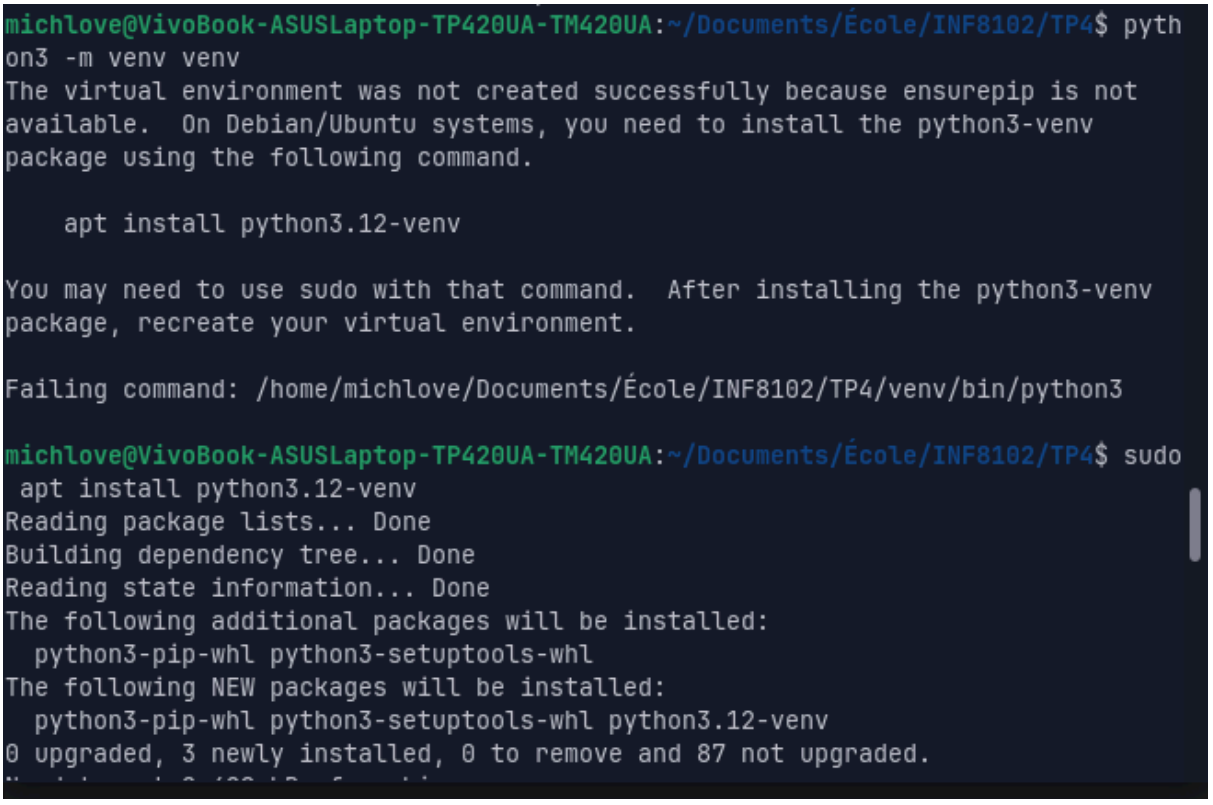


Figure 11: Installation de python3.12-venv

```
michlove@VivoBook-ASUSLaptop-TP420UA-TM420UA:~/Documents/École/INF8102/TP4$ pyth
on3 -m venv venv
michlove@VivoBook-ASUSLaptop-TP420UA-TM420UA:~/Documents/École/INF8102/TP4$ sour
ce venv/bin/activate
(venv) michlove@VivoBook-ASUSLaptop-TP420UA-TM420UA:~/Documents/École/INF8102/TP
4$ pip install troposphere boto3 pyyaml
Collecting troposphere
  Downloading troposphere-4.9.4-py3-none-any.whl.metadata (9.2 kB)
Collecting boto3
  Downloading boto3-1.41.1-py3-none-any.whl.metadata (6.8 kB)
Collecting pyyaml
  Downloading pyyaml-6.0.3-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_6
4.manylinux_2_28_x86_64.whl.metadata (2.4 kB)
Collecting cfn_flip>=1.0.2 (from troposphere)
  Downloading cfn_flip-1.3.0-py3-none-any.whl.metadata (6.0 kB)
Collecting botocore<1.42.0,>=1.41.1 (from boto3)
  Downloading botocore-1.41.1-py3-none-any.whl.metadata (5.9 kB)
Collecting jmespath<2.0.0,>=0.7.1 (from boto3)
  Downloading jmespath-1.0.1-py3-none-any.whl.metadata (7.6 kB)
```

Figure 12: Activation de venv et installation de troposphere, boto3 et pyyaml

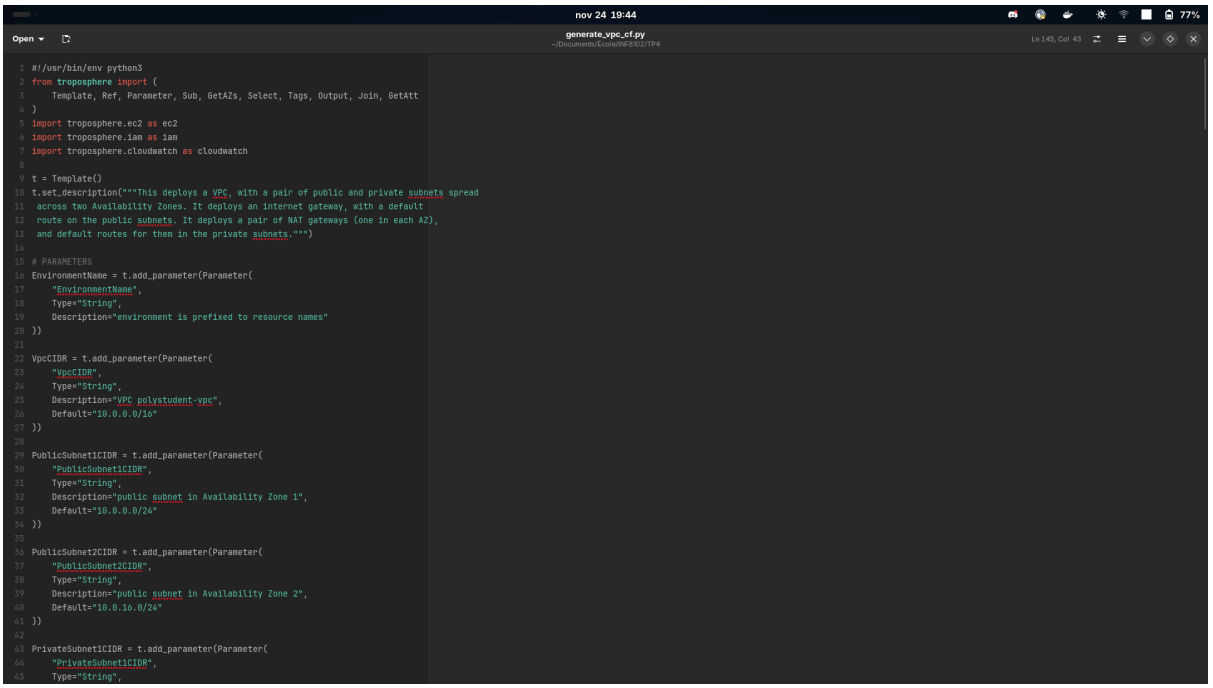
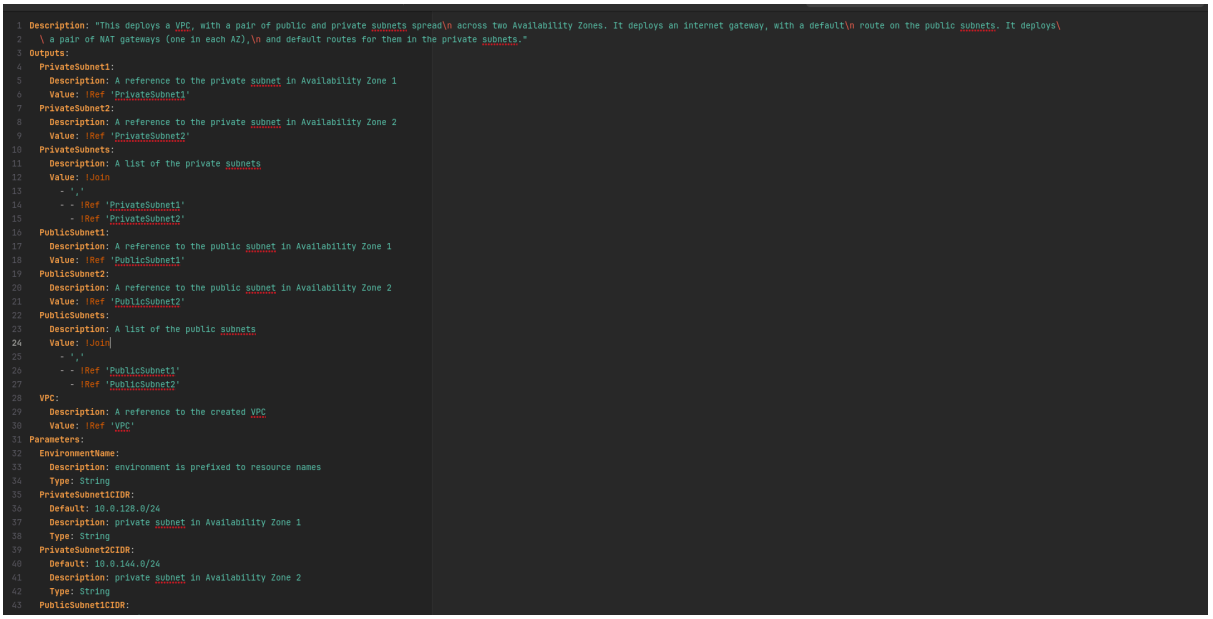


Figure 13: Création du fichier generate_vpc_cf.py pour reproduire le VPC

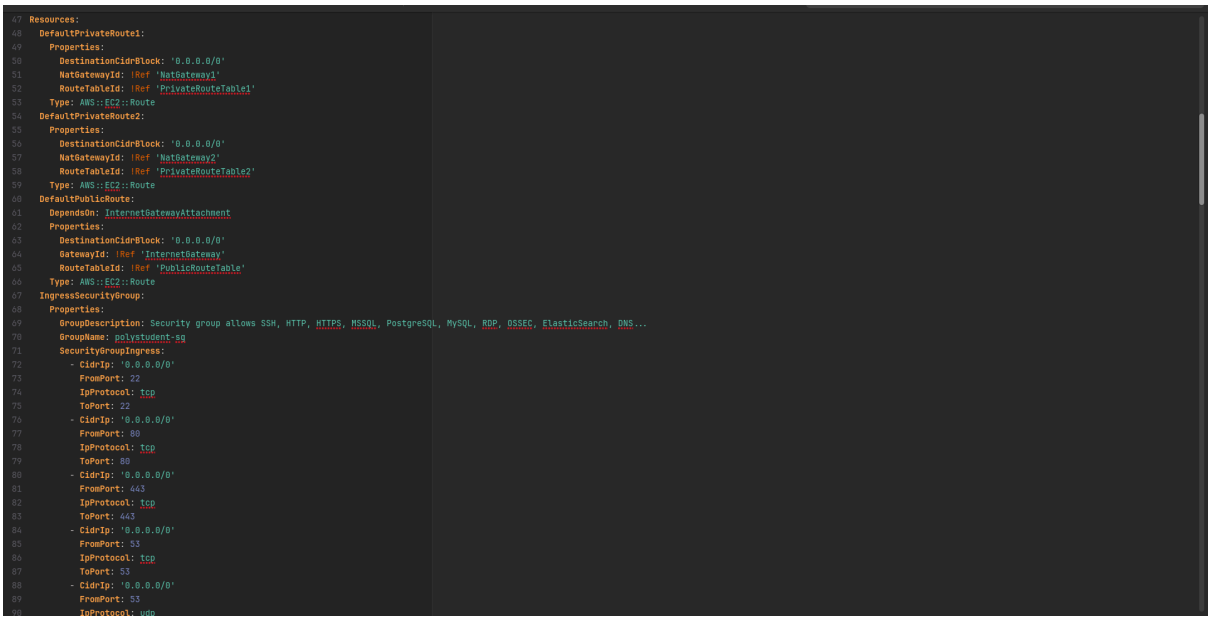
```
(venv) michlove@VivoBook-ASUSLaptop-TP420UA-TM420UA:~/Documents/École/INF8102/TP
4$ python generate_vpc_cf.py
Generated tp4_vpc.yaml
(venv) michlove@VivoBook-ASUSLaptop-TP420UA-TM420UA:~/Documents/École/INF8102/TP
4$
```

Figure 14: Exécution du fichier generate_vpc_cf.py



```
Description: "This deploys a VPC, with a pair of public and private subnets spread\n across two Availability Zones. It deploys an internet gateway, with a default\n route on the public subnets. It deploys\
 \ a pair of NAT gateways (one in each AZ),\n and default routes for them in the private subnets."
Outputs:
 PrivateSubnet1:
   Description: A reference to the private subnet in Availability Zone 1
   Value: !Ref 'PrivateSubnet1'
 PrivateSubnet2:
   Description: A reference to the private subnet in Availability Zone 2
   Value: !Ref 'PrivateSubnet2'
 PrivateSubnets:
   Description: A list of the private subnets
   Value: !Join
     - ','
     - - !Ref 'PrivateSubnet1'
       - !Ref 'PrivateSubnet2'
 PublicSubnet1:
   Description: A reference to the public subnet in Availability Zone 1
   Value: !Ref 'PublicSubnet1'
 PublicSubnet2:
   Description: A reference to the public subnet in Availability Zone 2
   Value: !Ref 'PublicSubnet2'
 PublicSubnets:
   Description: A list of the public subnets
   Value: !Join
     - ','
     - - !Ref 'PublicSubnet1'
       - !Ref 'PublicSubnet2'
 VPC:
   Description: A reference to the created VPC
   Value: !Ref 'VPC'
Parameters:
 EnvironmentName:
   Description: environment is prefixed to resource names
   Type: String
 PrivateSubnet1CIDR:
   Default: 10.0.128.0/24
   Description: private subnet in Availability Zone 1
   Type: String
 PrivateSubnet2CIDR:
   Default: 10.0.144.0/24
   Description: private subnet in Availability Zone 2
   Type: String
 PublicSubnet1CIDR:
```

Figure 15: Résultat suite à l'exécution du script python

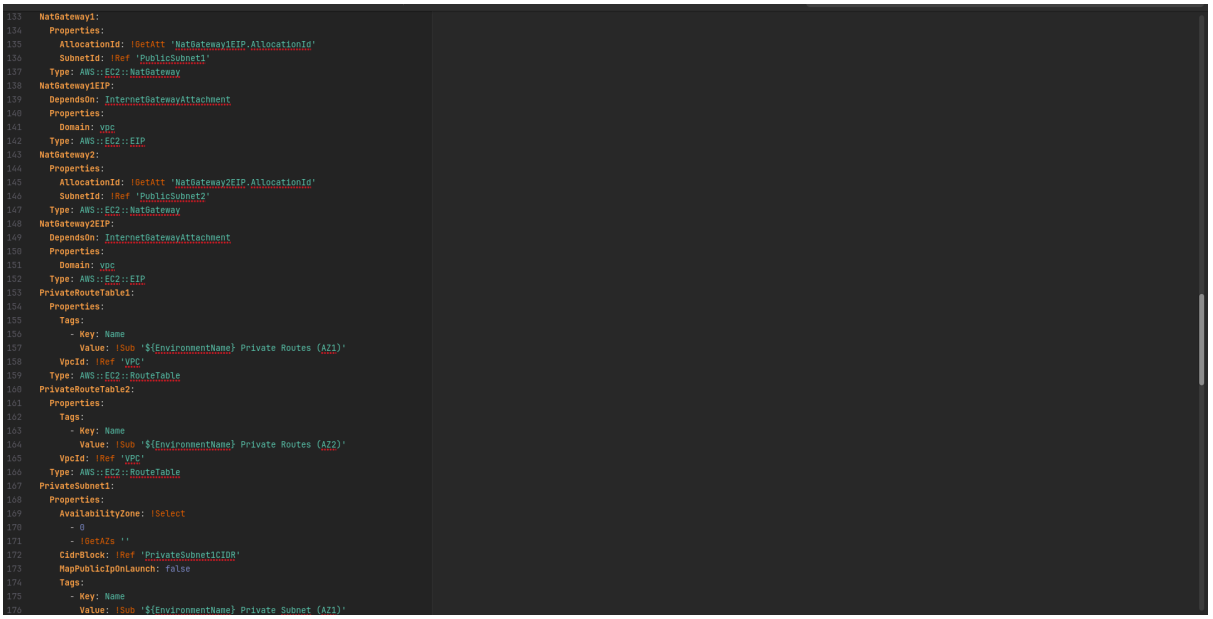


```
Resources:
  DefaultPrivateRoute1:
    Properties:
      DestinationCidrBlock: '0.0.0.0/0'
      NatGatewayId: !Ref 'NatGateway1'
      RouteTableId: !Ref 'PrivateRouteTable1'
    Type: AWS::EC2::Route
  DefaultPrivateRoute2:
    Properties:
      DestinationCidrBlock: '0.0.0.0/0'
      NatGatewayId: !Ref 'NatGateway2'
      RouteTableId: !Ref 'PrivateRouteTable2'
    Type: AWS::EC2::Route
  DefaultPublicRoute:
    DependsOn: InternetGatewayAttachment
    Properties:
      DestinationCidrBlock: '0.0.0.0/0'
      GatewayId: !Ref 'InternetGateway'
      RouteTableId: !Ref 'PublicRouteTable'
    Type: AWS::EC2::Route
  IngressSecurityGroup:
    Properties:
      GroupDescription: Security group allows SSH, HTTP, HTTPS, MSSQL, PostgreSQL, MySQL, RDP, OSSEC, ElasticSearch, DNS...
      GroupName: polystudent-sg
      SecurityGroupIngress:
        - CidrIp: '0.0.0.0/0'
          FromPort: 22
          IpProtocol: tcp
          ToPort: 22
        - CidrIp: '0.0.0.0/0'
          FromPort: 80
          IpProtocol: tcp
          ToPort: 80
        - CidrIp: '0.0.0.0/0'
          FromPort: 443
          IpProtocol: tcp
          ToPort: 443
        - CidrIp: '0.0.0.0/0'
          FromPort: 53
          IpProtocol: tcp
          ToPort: 53
        - CidrIp: '0.0.0.0/0'
          FromPort: 53
          IpProtocol: udp
```

Figure 16: Résultat suite à l'exécution du script python

```
        IpProtocol: udp
        ToPort: 53
      - CidrIp: '0.0.0.0/0'
        FromPort: 1433
        IpProtocol: tcp
        ToPort: 1433
      - CidrIp: '0.0.0.0/0'
        FromPort: 5432
        IpProtocol: tcp
        ToPort: 5432
      - CidrIp: '0.0.0.0/0'
        FromPort: 3306
        IpProtocol: tcp
        ToPort: 3306
      - CidrIp: '0.0.0.0/0'
        FromPort: 3389
        IpProtocol: tcp
        ToPort: 3389
      - CidrIp: '0.0.0.0/0'
        FromPort: 1514
        IpProtocol: udp
        ToPort: 1514
      - CidrIp: '0.0.0.0/0'
        FromPort: 9200
        IpProtocol: tcp
        ToPort: 9200
      - CidrIp: '0.0.0.0/0'
        FromPort: 9300
        IpProtocol: tcp
        ToPort: 9300
    VpcId: !Ref 'VPC'
  Type: AWS::EC2::SecurityGroup
InternetGateway:
  Properties:
    Tags:
      - Key: Name
        Value: !Ref 'EnvironmentName'
  Type: AWS::EC2::InternetGateway
InternetGatewayAttachment:
  Properties:
    InternetGatewayId: !Ref 'InternetGateway'
    VpcId: !Ref 'VPC'
  Type: AWS::EC2::VPCGatewayAttachment
NatGateway1:
```

Figure 17: Résultat suite à l'exécution du script python



```
NatGateway1:
  Properties:
    AllocationId: !GetAtt 'NatGateway1EIP.AllocationId'
    SubnetId: !Ref 'PublicSubnet1'
  Type: AWS::EC2::NatGateway
NatGateway1EIP:
  DependsOn: InternetGatewayAttachment
  Properties:
    Domain: vpc
  Type: AWS::EC2::EIP
NatGateway2:
  Properties:
    AllocationId: !GetAtt 'NatGateway2EIP.AllocationId'
    SubnetId: !Ref 'PublicSubnet2'
  Type: AWS::EC2::NatGateway
NatGateway2EIP:
  DependsOn: InternetGatewayAttachment
  Properties:
    Domain: vpc
  Type: AWS::EC2::EIP
PrivateRouteTable1:
  Properties:
    Tags:
      - Key: Name
        Value: !Sub '${EnvironmentName} Private Routes (AZ1)'
    VpcId: !Ref 'VPC'
  Type: AWS::EC2::RouteTable
PrivateRouteTable2:
  Properties:
    Tags:
      - Key: Name
        Value: !Sub '${EnvironmentName} Private Routes (AZ2)'
    VpcId: !Ref 'VPC'
  Type: AWS::EC2::RouteTable
PrivateSubnet1:
  Properties:
    AvailabilityZone: !Select
      - 0
      - !GetAZs ''
    CidrBlock: !Ref 'PrivateSubnet1CIDR'
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Name
        Value: !Sub '${EnvironmentName} Private Subnet (AZ1)'
```

Figure 18: Résultat suite à l'exécution du script python

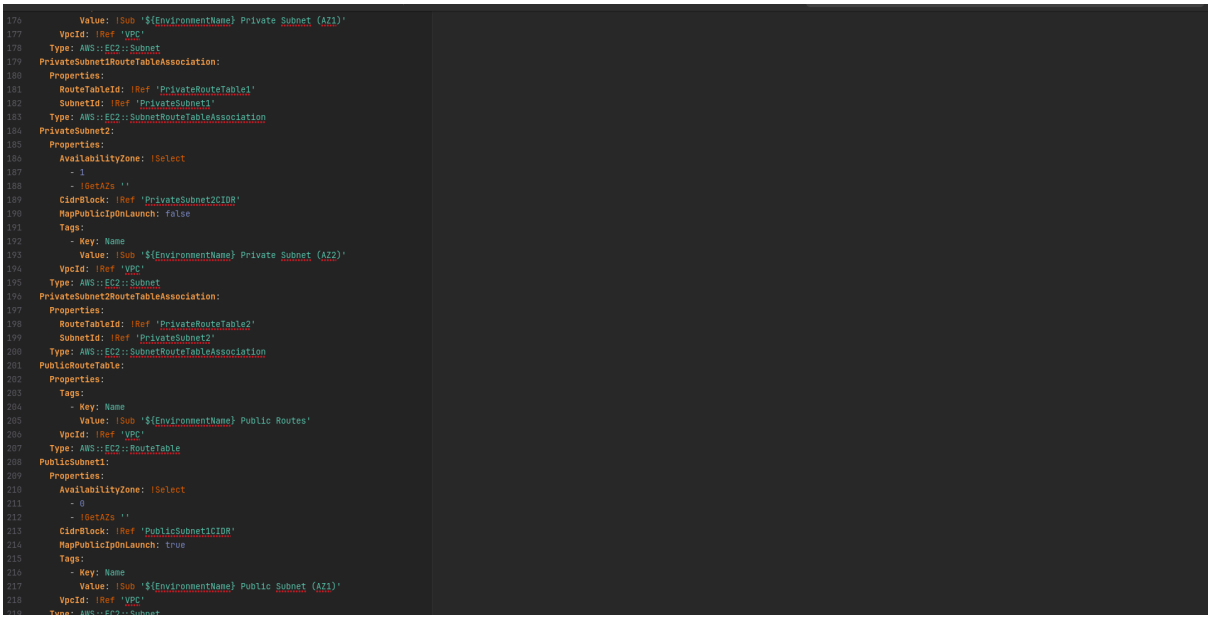


Figure 19: Résultat suite à l'exécution du script python

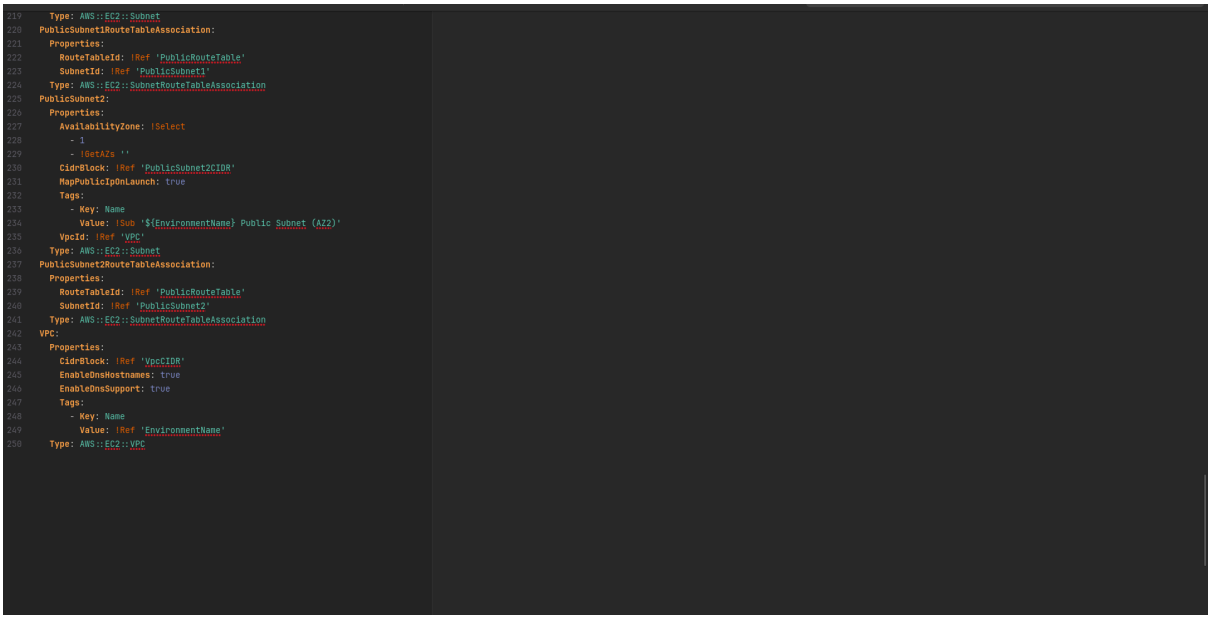


Figure 20: Résultat suite à l'exécution du script python

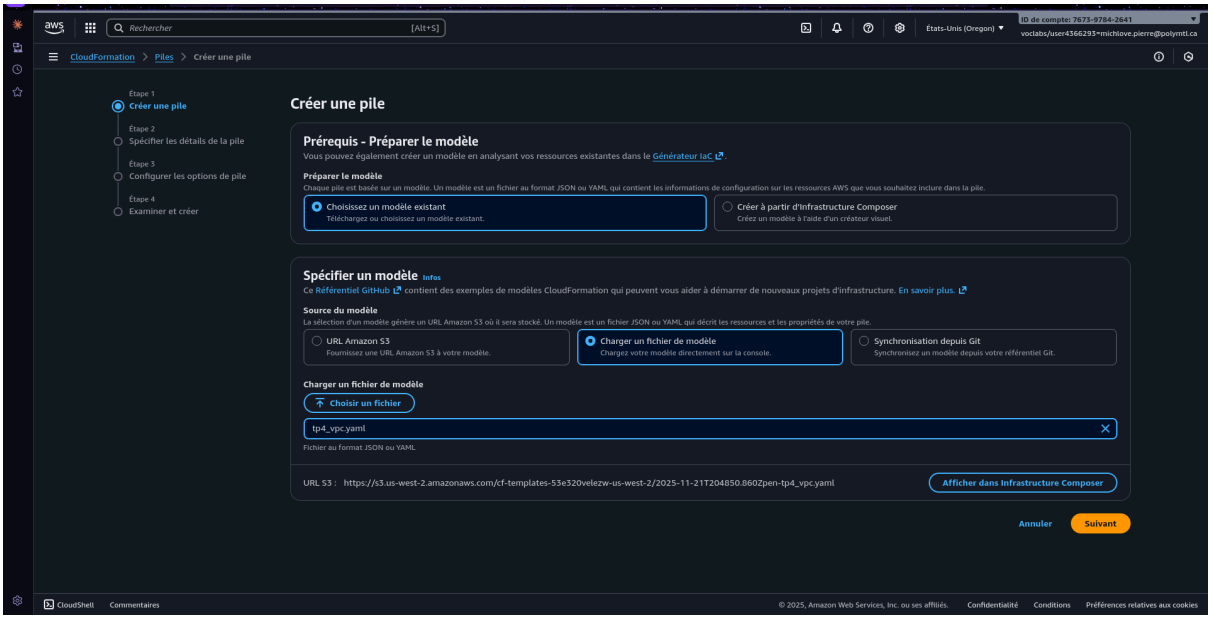


Figure 21: Création de la pile et chargement du fichier yaml obtenu à l'aide du script

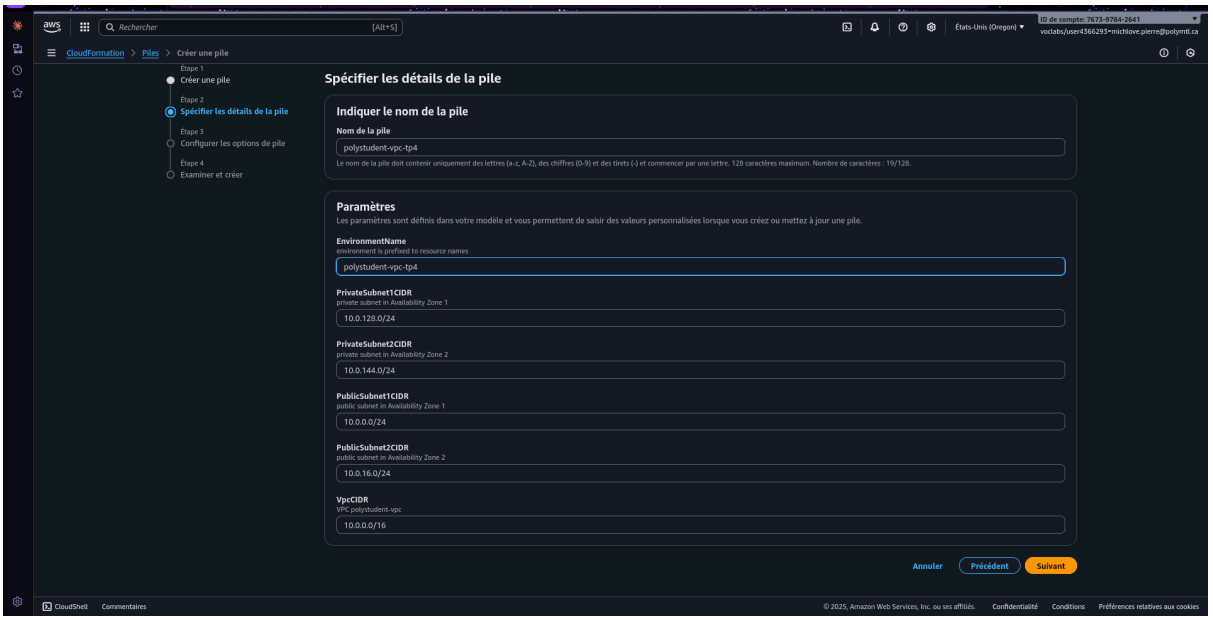


Figure 22: Inscription du nom de la pile

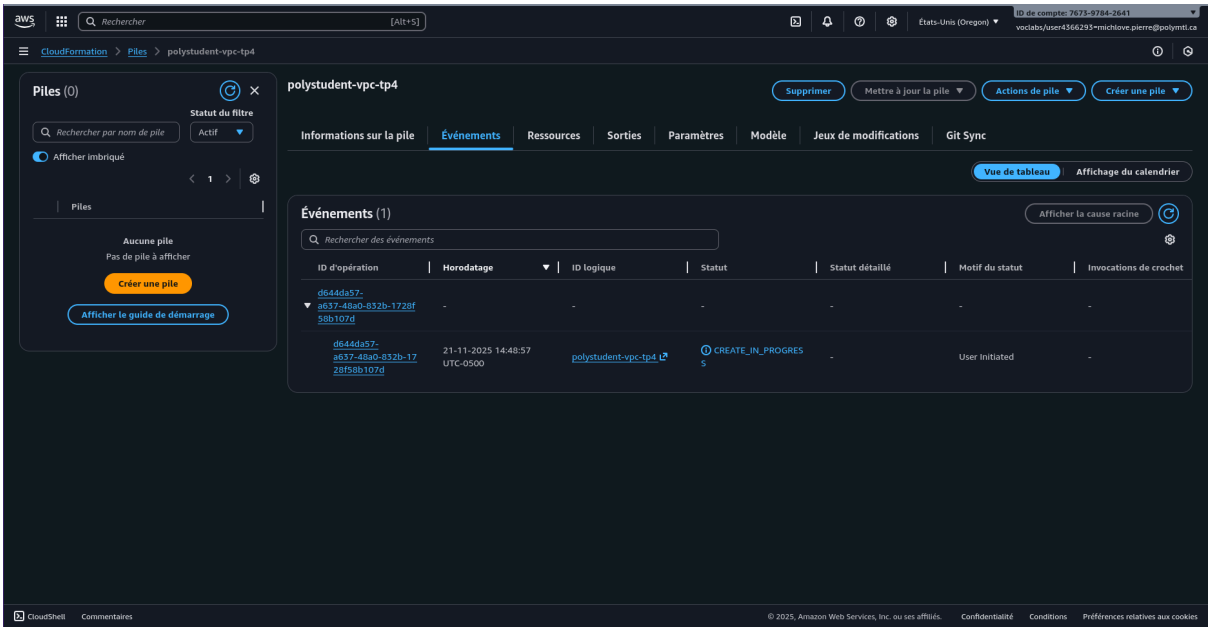


Figure 23: Démontrer que la pile est en cours d'exécution

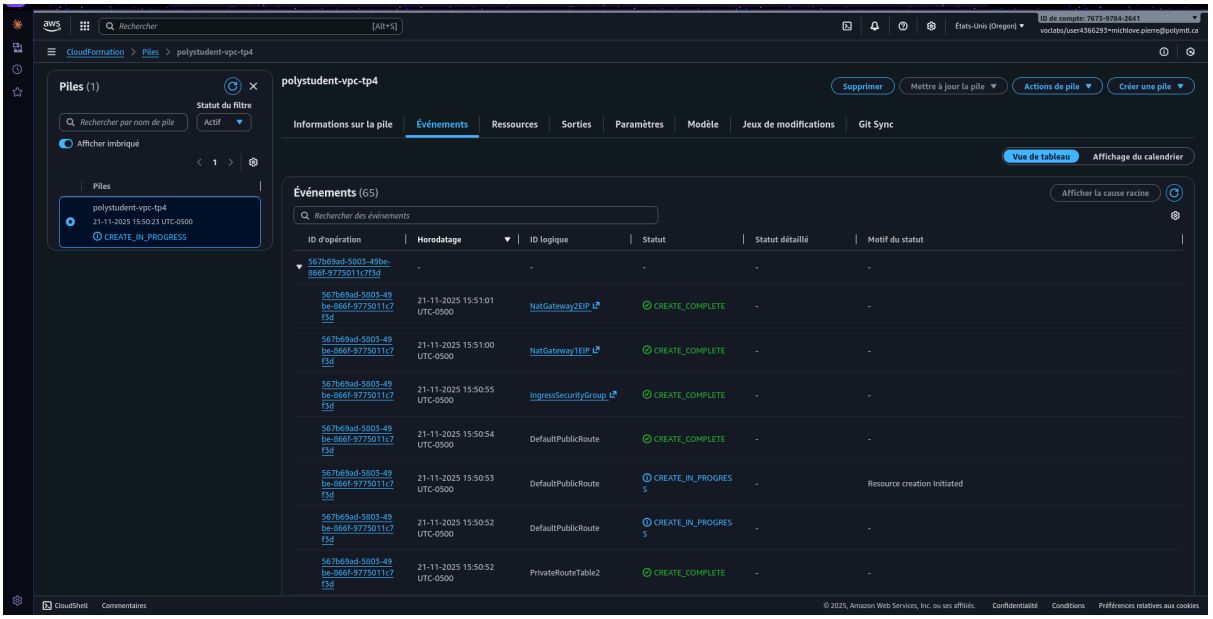


Figure 24: Démontrer que le NAT Gateway, le groupe de sécurité et les tables de routage sont bien créer à l'aide du YAML

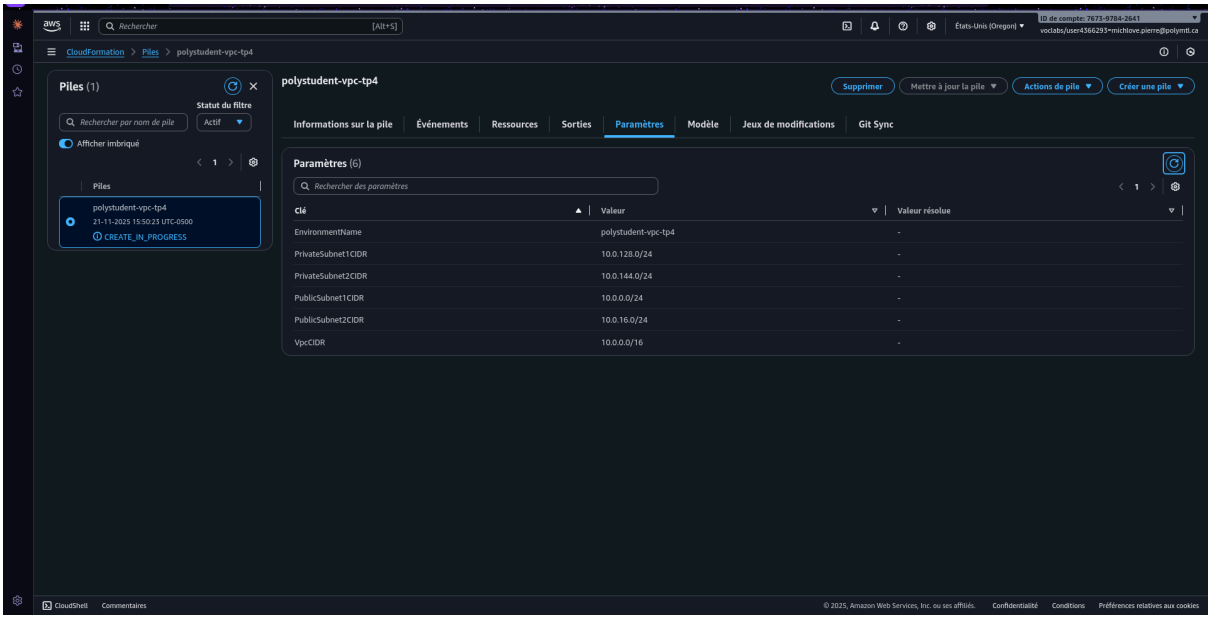


Figure 25: Afficher les paramètres créés grâce au fichier YAML

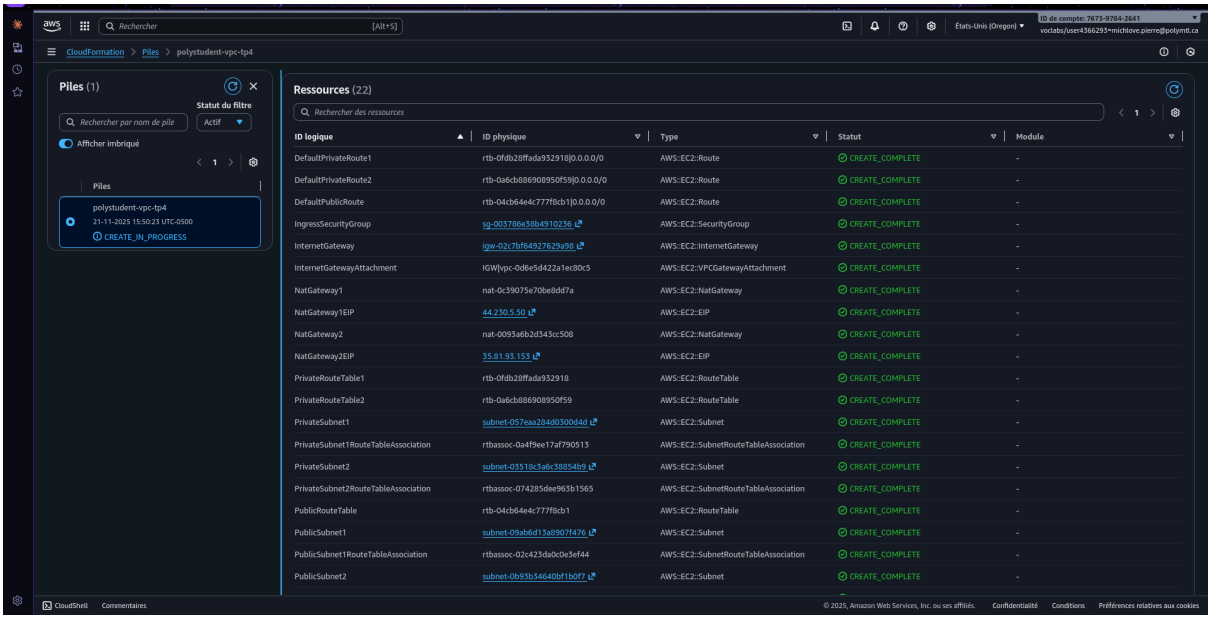


Figure 26: Afficher les ressources créés grâce au fichier YAML

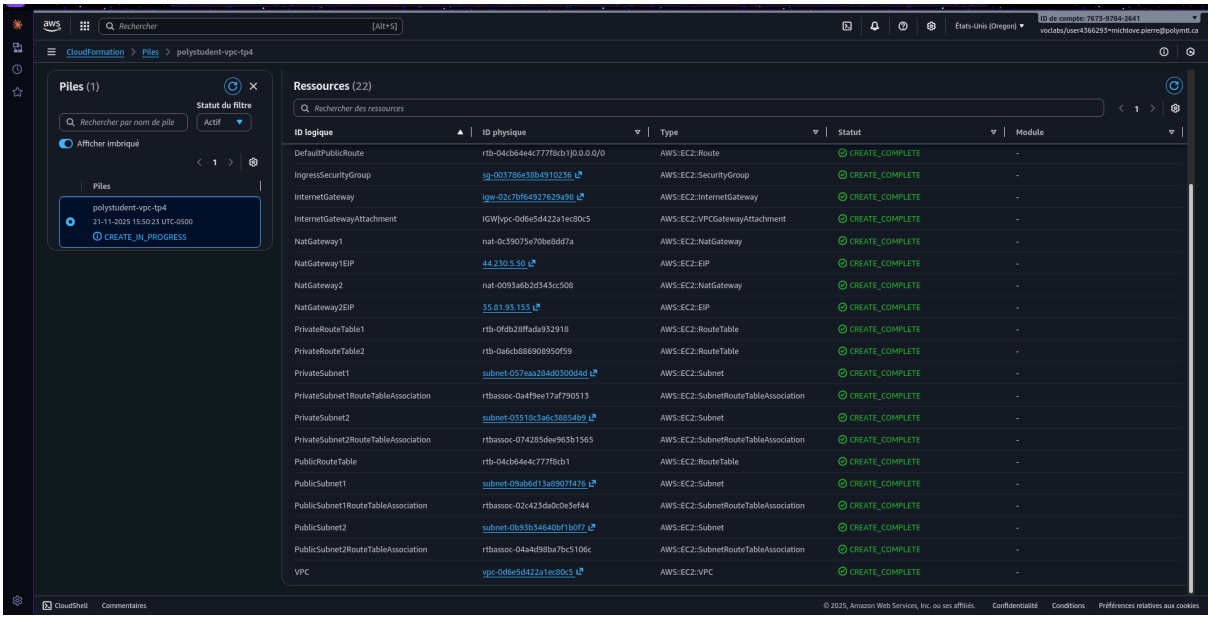


Figure 27: Afficher les ressources créés grâce au fichier YAML(2)

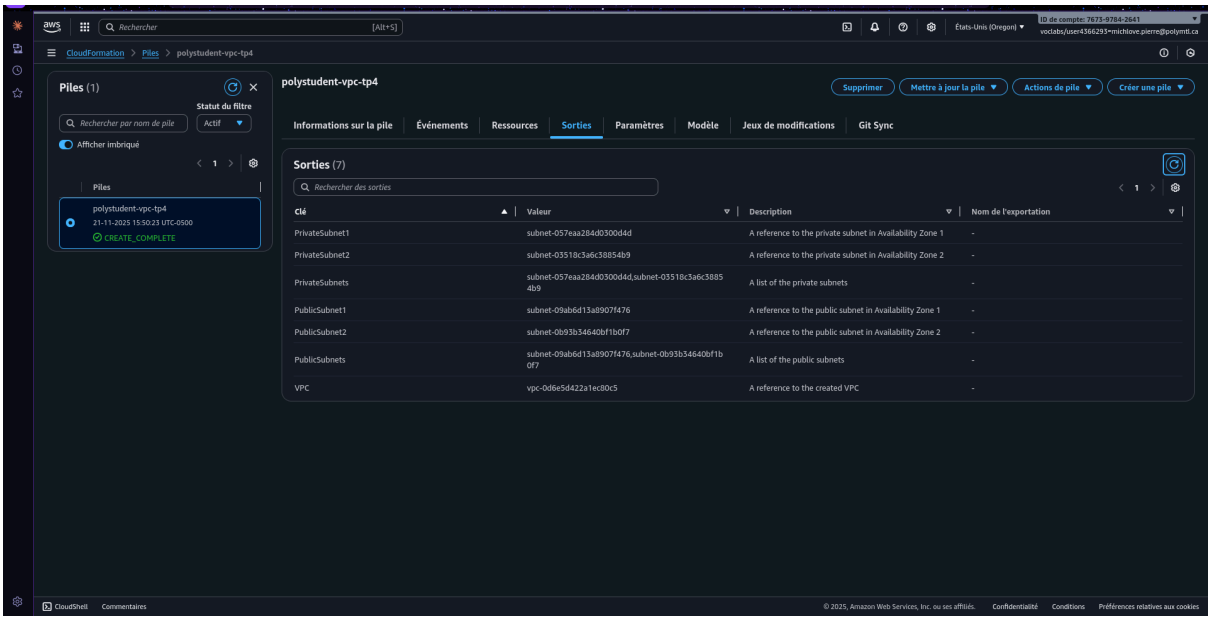


Figure 28: Afficher les sorties créés grâce au fichier YAML

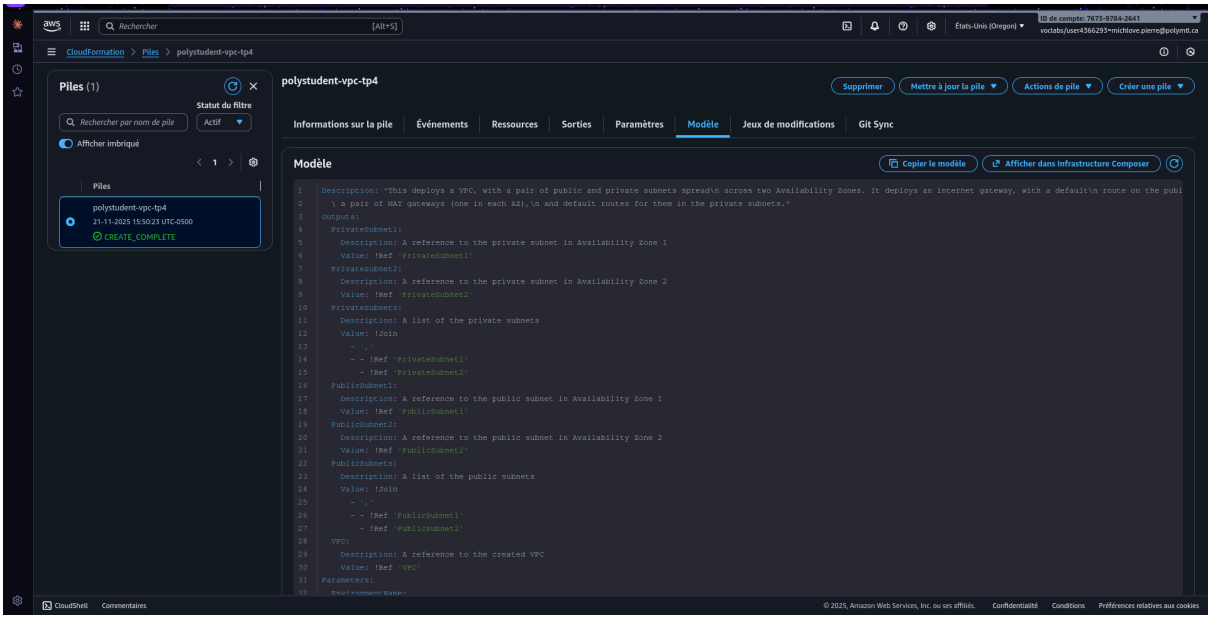


Figure 29: Afficher le modèle extrait du fichier YAML

2.

```python
import boto3
from botocore.exceptions import ClientError

KMS_KEY_ARN = "arn:aws:kms:us-west-2:767397842641:key/44fd78a9-01ad-4323-8ccd-24e608de0197"
BUCKET_NAME = "polystudents3-anis-michlove-unique"
REGION = "us-west-2"

s3_client = boto3.client("s3", region_name=REGION)


def create_bucket(bucket):
    try:
        s3_client.create_bucket(
            Bucket=bucket,
            CreateBucketConfiguration={"LocationConstraint": REGION}
        )
        print(f'Bucket {BUCKET_NAME} créé avec succès dans la région {REGION}.')
    except ClientError as e:
        if e.response["Error"]["Code"] in ["BucketAlreadyOwnedByYou", "BucketAlreadyExists"]:
            print(f"\ Bucket {bucket} existe déjà.")
        else:
            raise e

def secure_bucket(bucket):
    s3_client.put_bucket_versioning(
        Bucket=bucket,
        VersioningConfiguration={"Status": "Enabled"}
    )
    print(f"Versioning activé pour {bucket}.")

    if bucket == BUCKET_NAME:
        s3_client.put_bucket_encryption(
            Bucket=bucket,
            ServerSideEncryptionConfiguration={
                "Rules": [{
                    "ApplyServerSideEncryptionByDefault": {
                        "SSEAlgorithm": "aws:kms",
                        "KMSMasterKeyID": KMS_KEY_ARN
                    }
                }]
            }
        )
        print(f'Chiffrement SSE-KMS activé pour {BUCKET_NAME} avec la clé {KMS_KEY_ARN}.')
```

Figure 30: Création du fichier generate_s3_bucket.py

```
            Bucket=bucket,
            CreateBucketConfiguration={"LocationConstraint": REGION}
        )
        print(f'Bucket {BUCKET_NAME} créé avec succès dans la région {REGION}.')
    except ClientError as e:
        if e.response["Error"]["Code"] in ["BucketAlreadyOwnedByYou", "BucketAlreadyExists"]:
            print(f"ℹ Bucket {bucket} existe déjà.")
        else:
            raise e

def secure_bucket(bucket):
    s3_client.put_bucket_versioning(
        Bucket=bucket,
        VersioningConfiguration={"Status": "Enabled"}
    )
    print(f"Versioning activé pour {bucket}.")

    if bucket == BUCKET_NAME:
        s3_client.put_bucket_encryption(
            Bucket=bucket,
            ServerSideEncryptionConfiguration={
                "Rules": [{
                    "ApplyServerSideEncryptionByDefault": {
                        "SSEAlgorithm": "aws:kms",
                        "KMSMasterKeyID": KMS_KEY_ARN
                    }
                }]
            }
        )
        print(f'Chiffrement SSE-KMS activé pour {BUCKET_NAME} avec la clé {KMS_KEY_ARN}.')

    s3_client.put_public_access_block(
        Bucket=bucket,
        PublicAccessBlockConfiguration={
            "BlockPublicAcls": True,
            "IgnorePublicAcls": True,
            "BlockPublicPolicy": True,
            "RestrictPublicBuckets": True
        }
    )
    print(f"Accès public bloqué pour {bucket}.")
    print(f'Bucket {BUCKET_NAME} configuré avec succès selon les exigences de sécurité.')
    print(f'Politiques appliquées : Versioning, SSE-KMS, Contrôle d\'accès privé, Blocage d\'accès public.')
```

Figure 31: Création du fichier generate_s3_bucket.py (2)

```
(venv) michlove@VivoBook-ASUSLaptop-TP420UA-TM420UA:~/Documents/École/INF8102/TP
4$ python generate_s3_bucket.py
Bucket polystudents3-anis-michlove-unique créé avec succès dans la région us-wes
t-2.
Versioning activé pour polystudents3-anis-michlove-unique.
Chiffrement SSE-KMS activé pour polystudents3-anis-michlove-unique avec la clé a
rn:aws:kms:us-west-2:767397842641:key/44fd78a9-01ad-4323-8ccd-24e608de0197.
Accès public bloqué pour polystudents3-anis-michlove-unique.
Bucket polystudents3-anis-michlove-unique configuré avec succès selon les exigen
ces de sécurité.
Politiques appliquées : Versioning, SSE-KMS, Contrôle d'accès privé, Blocage d'a
ccès public.
(venv) michlove@VivoBook-ASUSLaptop-TP420UA-TM420UA:~/Documents/École/INF8102/TP
4$
```

Figure 32: Exécution du fichier generate_s3_bucket.py

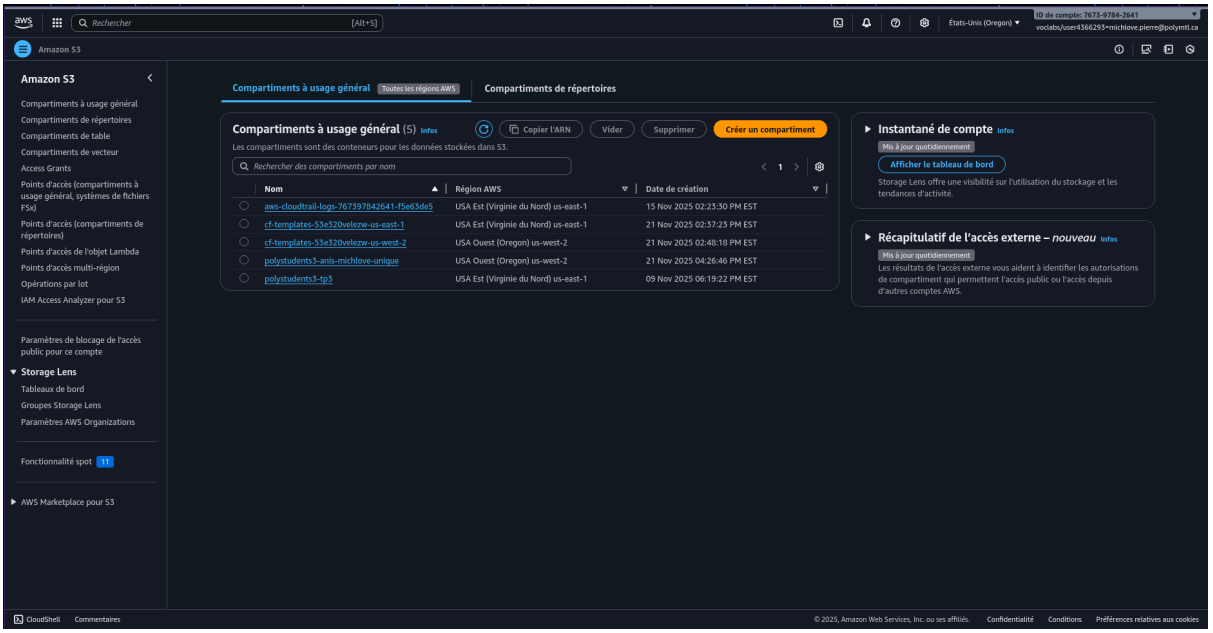
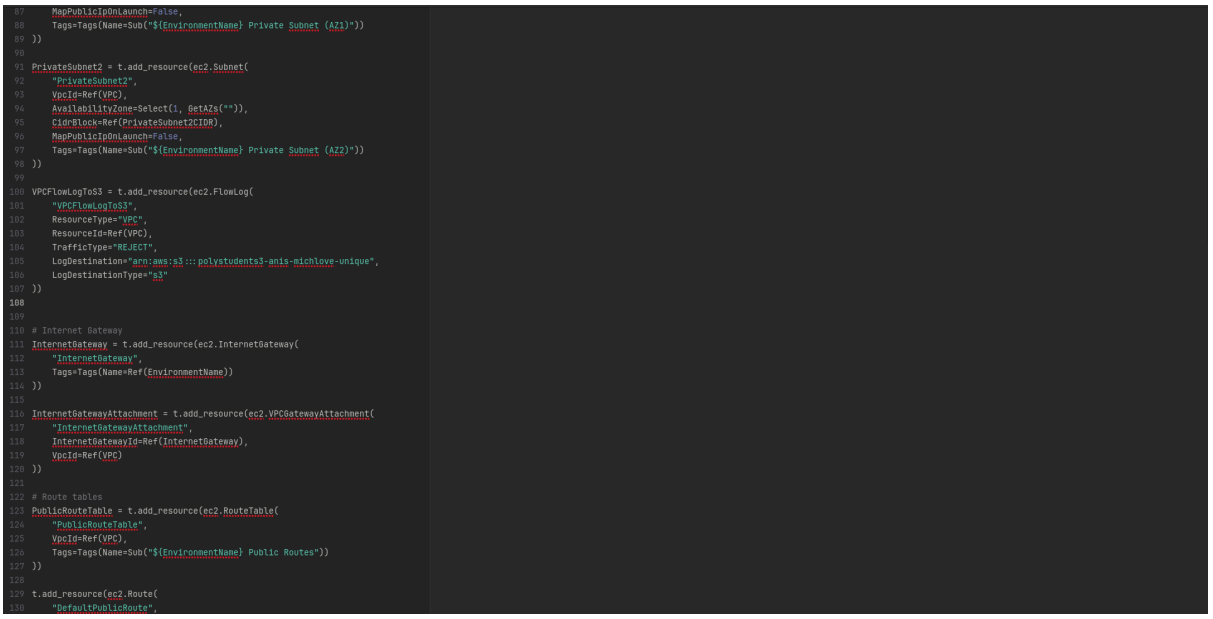


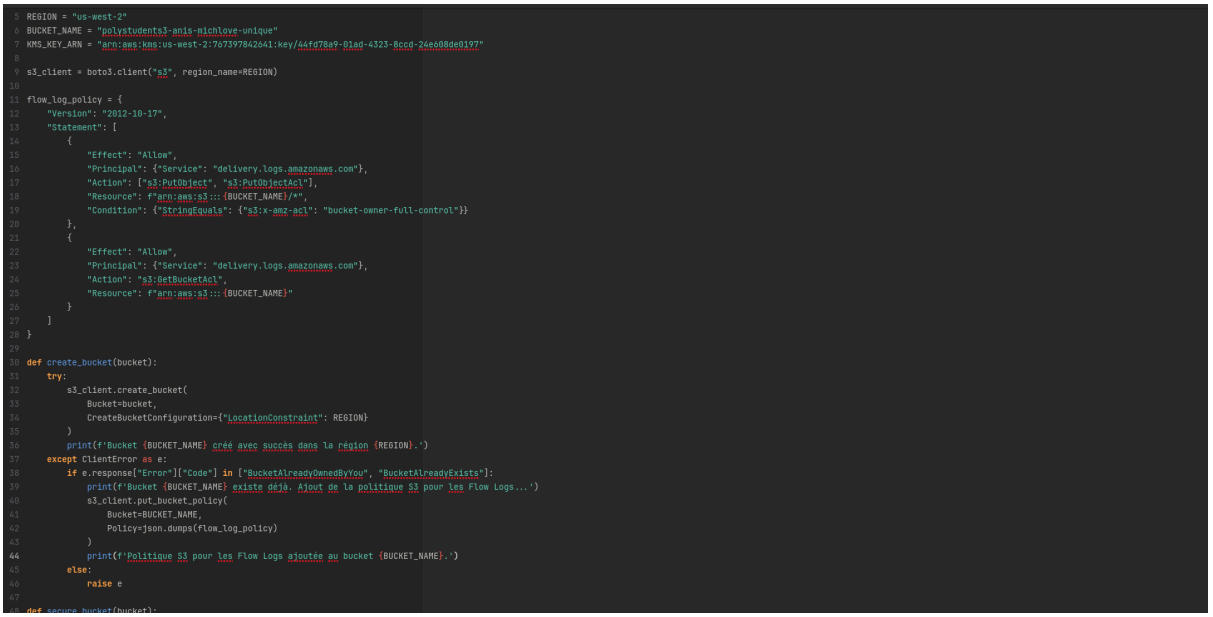
Figure 33: Démontrer que le bucket S3 polystudents3-anis-michlove-unique est créé

3.1



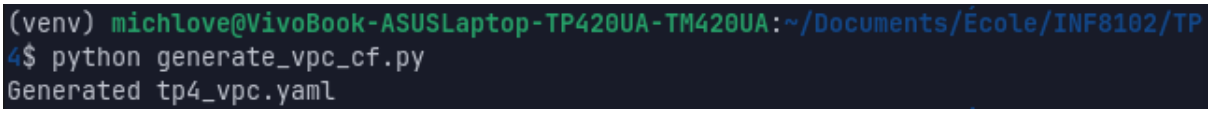
```
    MapPublicIpOnLaunch=False,
    Tags=Tags(Name=Sub("${EnvironmentName} Private Subnet (AZ1)"))
))

PrivateSubnet2 = t.add_resource(ec2.Subnet(
    "PrivateSubnet2",
    VpcId=Ref(VPC),
    AvailabilityZone=Select(1, GetAZs("")),
    CidrBlock=Ref(PrivateSubnet2CIDR),
    MapPublicIpOnLaunch=False,
    Tags=Tags(Name=Sub("${EnvironmentName} Private Subnet (AZ2)"))
))

VPCFlowLogToS3 = t.add_resource(ec2.FlowLog(
    "VPCFlowLogToS3",
    ResourceType="VPC",
    ResourceId=Ref(VPC),
    TrafficType="REJECT",
    LogDestination="arn:aws:s3:::polystudents3-anis-michlove-unique",
    LogDestinationType="s3"
))


# Internet Gateway
InternetGateway = t.add_resource(ec2.InternetGateway(
    "InternetGateway",
    Tags=Tags(Name=Ref(EnvironmentName))
))

InternetGatewayAttachment = t.add_resource(ec2.VPCGatewayAttachment(
    "InternetGatewayAttachment",
    InternetGatewayId=Ref(InternetGateway),
    VpcId=Ref(VPC)
))

# Route tables
PublicRouteTable = t.add_resource(ec2.RouteTable(
    "PublicRouteTable",
    VpcId=Ref(VPC),
    Tags=Tags(Name=Sub("${EnvironmentName} Public Routes"))
))

t.add_resource(ec2.Route(
    "DefaultPublicRoute",
```

Figure 34: Modification du fichier generate_vpc_cf.py pour ajouter le VPC flow logs



```
REGION = "us-west-2"
BUCKET_NAME = "polystudents3-anis-michlove-unique"
KMS_KEY_ARN = "arn:aws:kms:us-west-2:767397842641:key/44fd78a9-01ad-4323-8ccd-24e608de0197"

s3_client = boto3.client("s3", region_name=REGION)

flow_log_policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {"Service": "delivery.logs.amazonaws.com"},
            "Action": ["s3:PutObject", "s3:PutObjectAcl"],
            "Resource": f"arn:aws:s3:::{BUCKET_NAME}/*",
            "Condition": {"StringEquals": {"s3:x-amz-acl": "bucket-owner-full-control"}}
        },
        {
            "Effect": "Allow",
            "Principal": {"Service": "delivery.logs.amazonaws.com"},
            "Action": "s3:GetBucketAcl",
            "Resource": f"arn:aws:s3:::{BUCKET_NAME}"
        }
    ]
}

def create_bucket(bucket):
    try:
        s3_client.create_bucket(
            Bucket=bucket,
            CreateBucketConfiguration={"LocationConstraint": REGION}
        )
        print(f'Bucket {BUCKET_NAME} créé avec succès dans la région {REGION}.')
    except ClientError as e:
        if e.response["Error"]["Code"] in ["BucketAlreadyOwnedByYou", "BucketAlreadyExists"]:
            print(f'Bucket {BUCKET_NAME} existe déjà. Ajout de la politique S3 pour les Flow Logs...')
            s3_client.put_bucket_policy(
                Bucket=BUCKET_NAME,
                Policy=json.dumps(flow_log_policy)
            )
            print(f'Politique S3 pour les Flow Logs ajoutée au bucket {BUCKET_NAME}.')
        else:
            raise e

def secure_bucket(bucket):
```

Figure 35: Modification du fichier generate_s3_bucket.py pour ajouter une politique dans S3 pour faire fonctionner flow logs



```
(venv) michlove@VivoBook-ASUSLaptop-TP420UA-TM420UA:~/Documents/École/INF8102/TP
4$ python generate_vpc_cf.py
Generated tp4_vpc.yaml
```

Figure 35: Exécution du fichier generate_vpc_cf.py

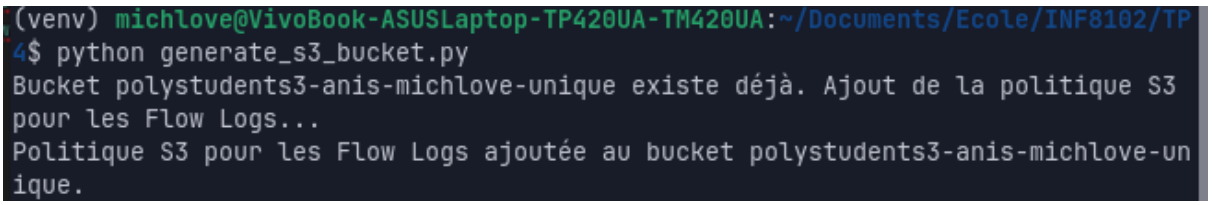


Figure 35: Exécution du fichier generate_s3_bucket.py

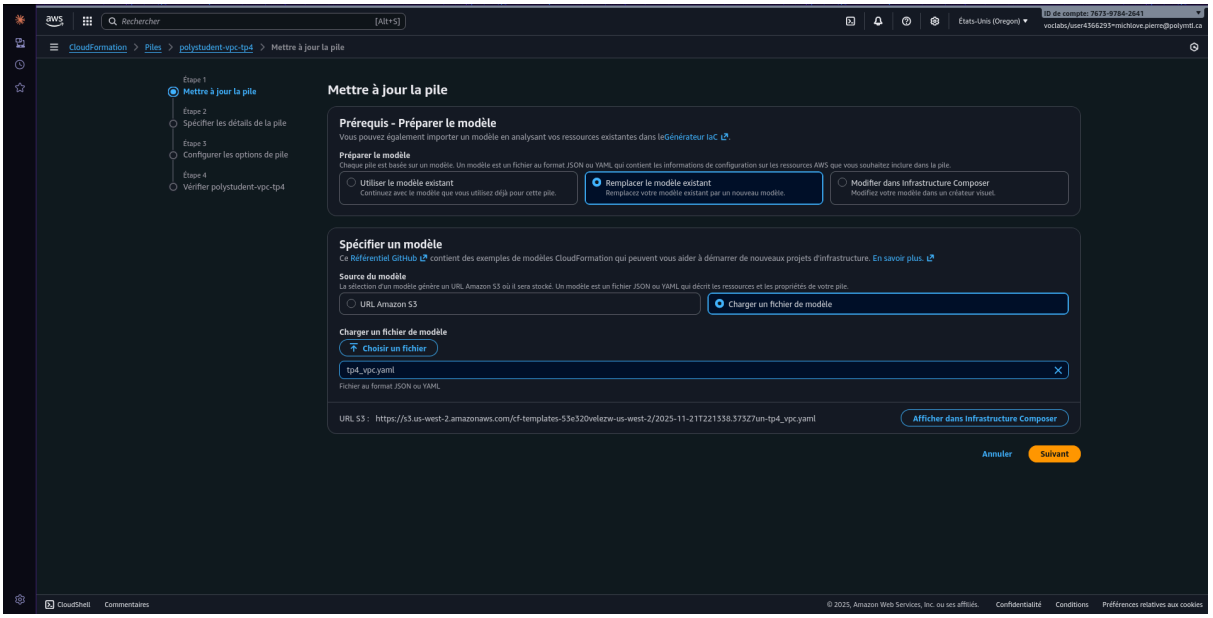


Figure 36: Mise à jour de la pile et chargement du fichier yaml obtenu à l'aide du script

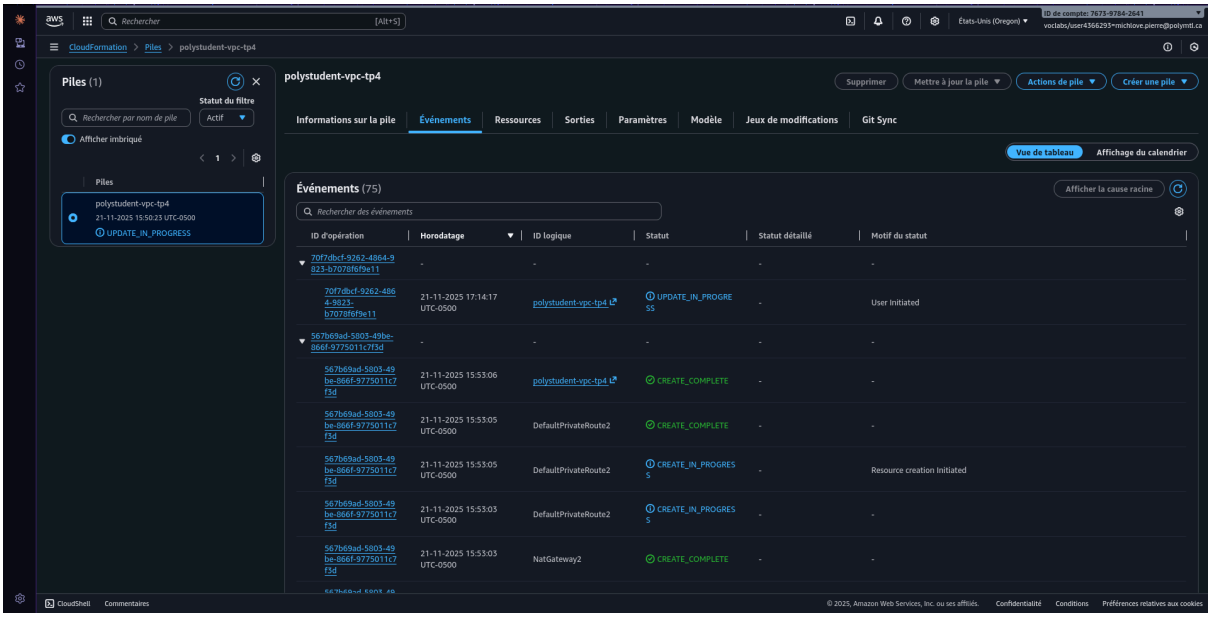


Figure 37: Démontrer que la pile est en cours de mise à jour

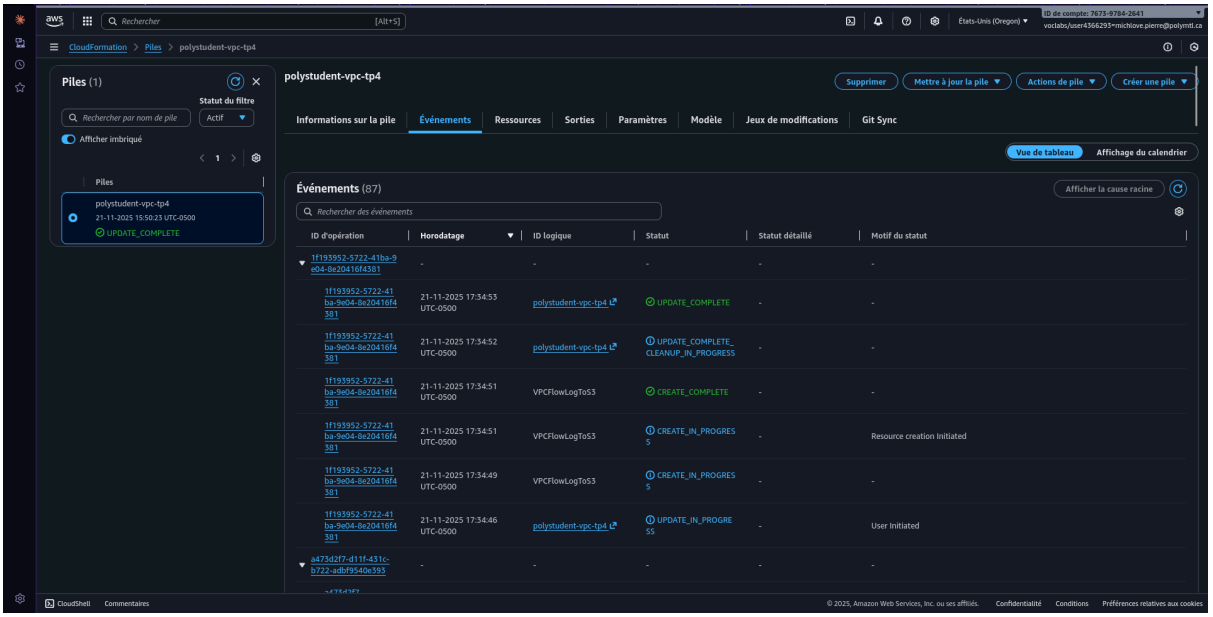


Figure 38: Démontrer que la pile à été mise à jour avec succès

3.2

```python
# Public Instances
PublicInstance1 = t.add_resource(ec2.Instance(
    "PublicInstance1",
    ImageId="ami-00f46ccd1cbfb363e",
    InstanceType="t3.micro",
    SubnetId=Ref(PublicSubnet1),
    IamInstanceProfile="LabInstanceProfile",
    Tags=Tags(Name=Sub("${EnvironmentName} Public Instance 1"))
))

PublicInstance2 = t.add_resource(ec2.Instance(
    "PublicInstance2",
    ImageId="ami-00f46ccd1cbfb363e",
    InstanceType="t3.micro",
    SubnetId=Ref(PublicSubnet2),
    IamInstanceProfile="LabInstanceProfile",
    Tags=Tags(Name=Sub("${EnvironmentName} Public Instance 2"))
))

# Private Instances
PrivateInstance1 = t.add_resource(ec2.Instance(
    "PrivateInstance1",
    ImageId="ami-00f46ccd1cbfb363e",
    InstanceType="t3.micro",
    SubnetId=Ref(PrivateSubnet1),
    IamInstanceProfile="LabInstanceProfile",
    Tags=Tags(Name=Sub("${EnvironmentName} Private Instance 1"))
))

PrivateInstance2 = t.add_resource(ec2.Instance(
    "PrivateInstance2",
    ImageId="ami-00f46ccd1cbfb363e",
    InstanceType="t3.micro",
    SubnetId=Ref(PrivateSubnet2),
    IamInstanceProfile="LabInstanceProfile",
    Tags=Tags(Name=Sub("${EnvironmentName} Private Instance 2"))
))

for i, instance in enumerate([PublicInstance1, PublicInstance2, PrivateInstance1, PrivateInstance2], 1):
```

Figure 39: Mise à jour du script python generate_vpc_cf.py pour mettre en place des instances EC2 public et privé

```python
for i, instance in enumerate([PublicInstance1, PublicInstance2, PrivateInstance1, PrivateInstance2], 1):
    t.add_resource(cloudwatch.Alarm(
        f"IngressPacketsAlarm{i}",
        AlarmDescription=f"Alarm if ingress packets exceed 1000 pkts/sec on instance {i}",
        Namespace="AWS/EC2",
        MetricName="NetworkPacketsIn",
        Dimensions=[cloudwatch.MetricDimension(
            Name="InstanceId",
            Value=Ref(instance)
        )],
        Statistic="Average",
        Period=60,
        EvaluationPeriods=1,
        Threshold=1000,
        ComparisonOperator="GreaterThanThreshold",
        AlarmActions=[],
        OKActions=[]
    ))
```

Figure 40: Mise à jour du script python generate_vpc_cf.py pour mettre en place des alarmes aux instances EC2 public et privé

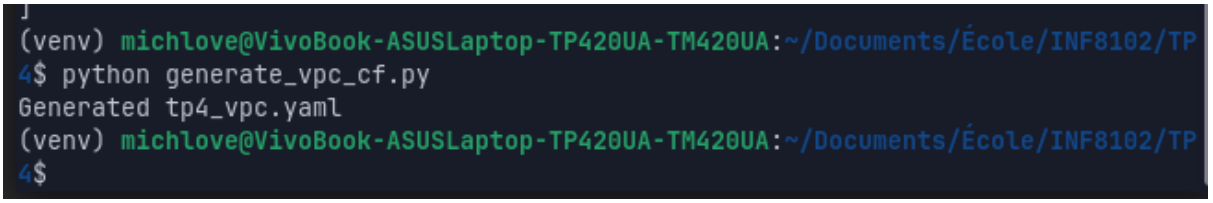


Figure 41: Exécution du script afin de mettre à jour le code du script generate_vpc_cf.py

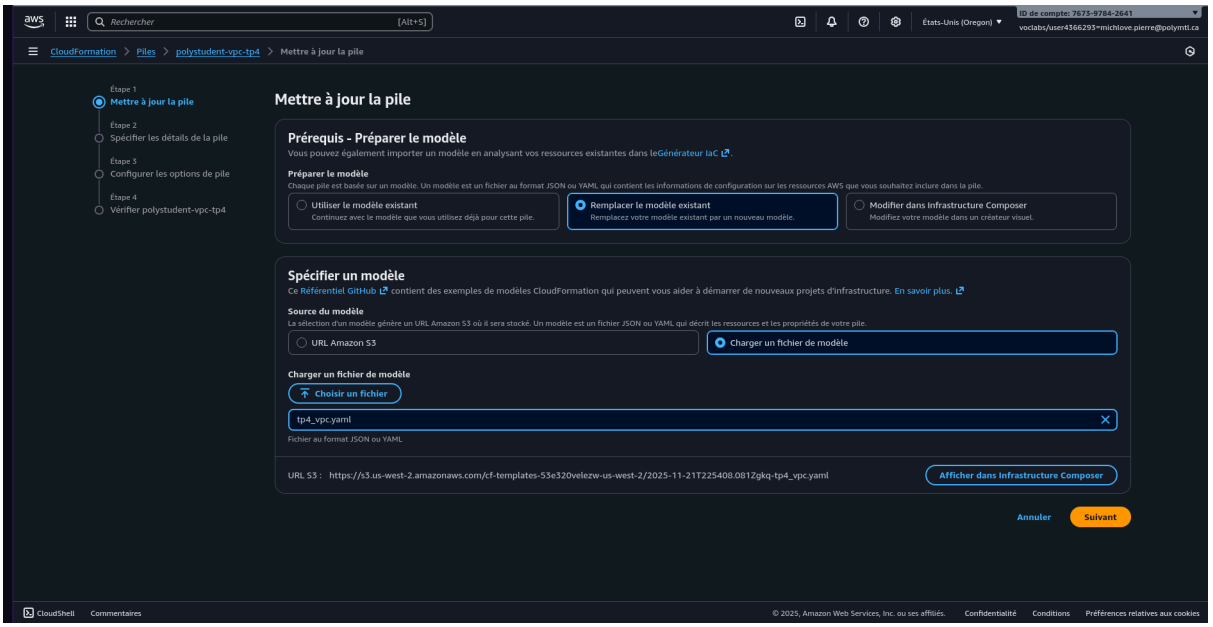


Figure 42: Mise à jour de la pile et chargement du fichier yaml obtenu à l'aide du script

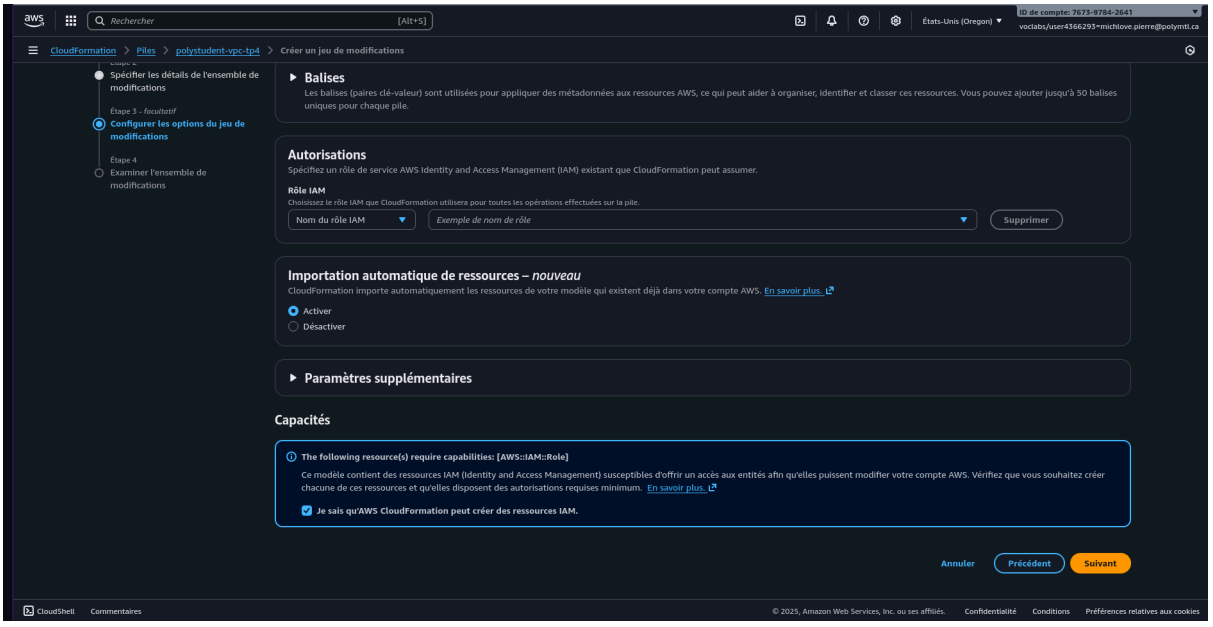


Figure 43: Démontrer que nous avons coché la section capacités

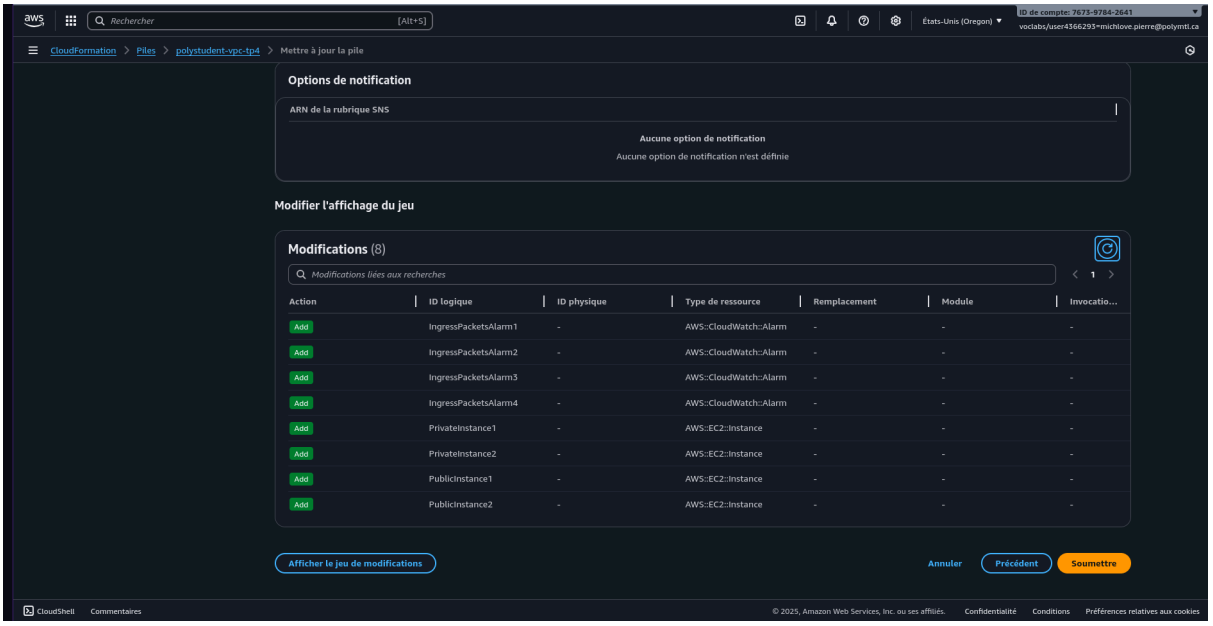


Figure 44: Démontrer que les instances EC2 et les alarmes font partie de la section mis à jour

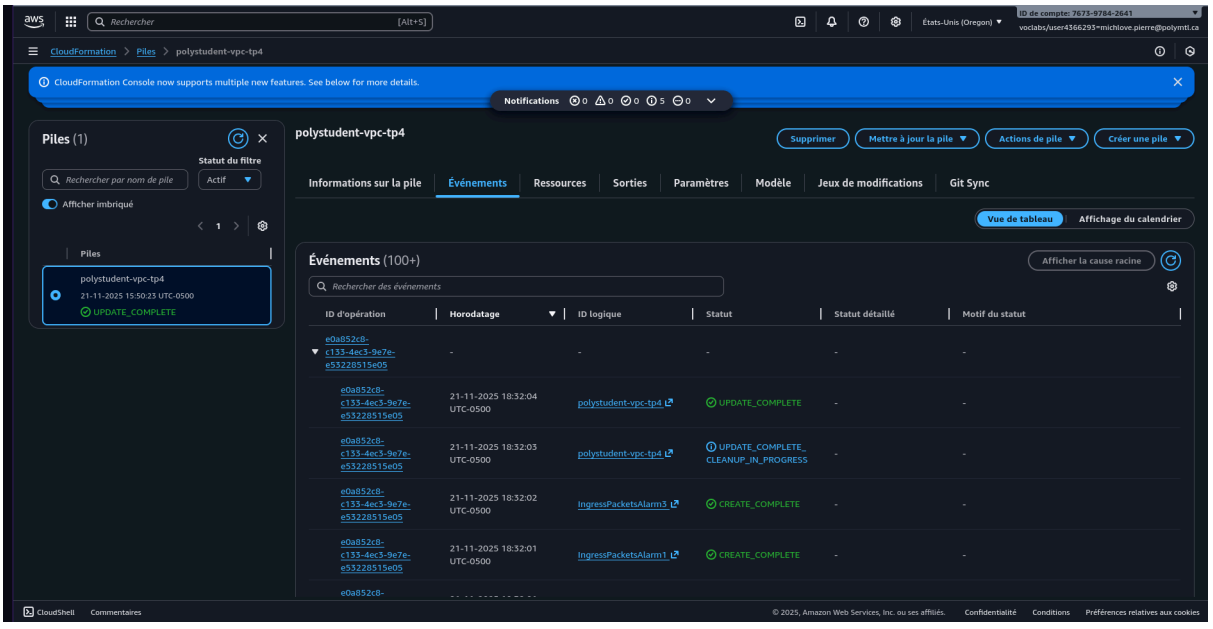
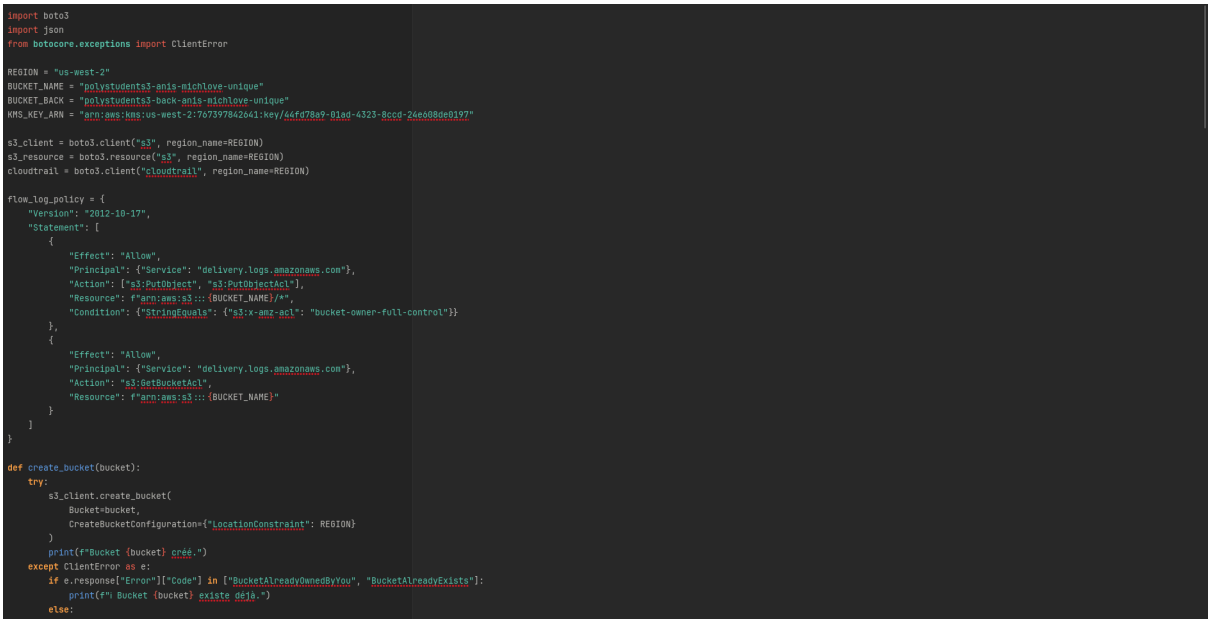


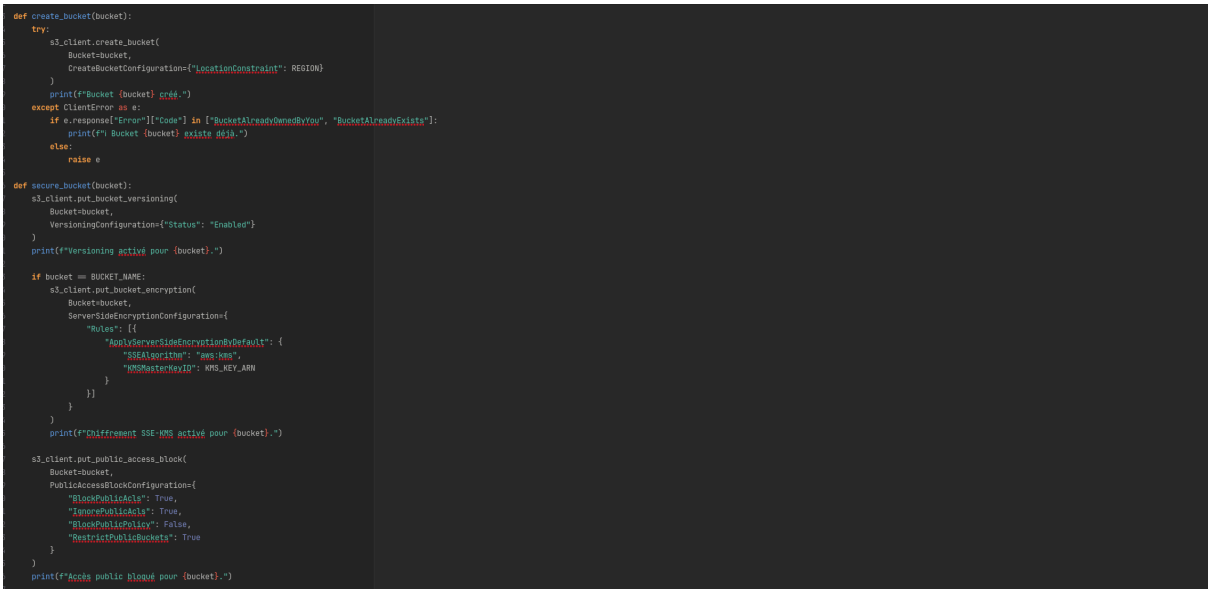
Figure 45: Démontrer que la pile à bien été mise à jour.

3.3



```python
import boto3
import json
from botocore.exceptions import ClientError

REGION = "us-west-2"
BUCKET_NAME = "polystudents3-anis-michlove-unique"
BUCKET_BACK = "polystudents3-back-anis-michlove-unique"
KMS_KEY_ARN = "arn:aws:kms:us-west-2:707397842641:key/44fd78a9-01ad-4323-8ccd-24e608de0197"

s3_client = boto3.client("s3", region_name=REGION)
s3_resource = boto3.resource("s3", region_name=REGION)
cloudtrail = boto3.client("cloudtrail", region_name=REGION)

flow_log_policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {"Service": "delivery.logs.amazonaws.com"},
            "Action": ["s3:PutObject", "s3:PutObjectAcl"],
            "Resource": f"arn:aws:s3:::{BUCKET_NAME}/*",
            "Condition": {"StringEquals": {"s3:x-amz-acl": "bucket-owner-full-control"}}
        },
        {
            "Effect": "Allow",
            "Principal": {"Service": "delivery.logs.amazonaws.com"},
            "Action": "s3:GetBucketAcl",
            "Resource": f"arn:aws:s3:::{BUCKET_NAME}"
        }
    ]
}

def create_bucket(bucket):
    try:
        s3_client.create_bucket(
            Bucket=bucket,
            CreateBucketConfiguration={"LocationConstraint": REGION}
        )
        print(f"Bucket {bucket} créé.")
    except ClientError as e:
        if e.response["Error"]["Code"] in ["BucketAlreadyOwnedByYou", "BucketAlreadyExists"]:
            print(f"⚠ Bucket {bucket} existe déjà.")
        else:
```

Figure 46: Mise à jour du script generate_s3_bucket_with_bucket_back.py pour créer le nouveau bucket



```python
def create_bucket(bucket):
    try:
        s3_client.create_bucket(
            Bucket=bucket,
            CreateBucketConfiguration={"LocationConstraint": REGION}
        )
        print(f"Bucket {bucket} créé.")
    except ClientError as e:
        if e.response["Error"]["Code"] in ["BucketAlreadyOwnedByYou", "BucketAlreadyExists"]:
            print(f"⚠ Bucket {bucket} existe déjà.")
        else:
            raise e

def secure_bucket(bucket):
    s3_client.put_bucket_versioning(
        Bucket=bucket,
        VersioningConfiguration={"Status": "Enabled"}
    )
    print(f"Versioning activé pour {bucket}.")

    if bucket == BUCKET_NAME:
        s3_client.put_bucket_encryption(
            Bucket=bucket,
            ServerSideEncryptionConfiguration={
                "Rules": [{
                    "ApplyServerSideEncryptionByDefault": {
                        "SSEAlgorithm": "aws:kms",
                        "KMSMasterKeyID": KMS_KEY_ARN
                    }
                }]
            }
        )
        print(f"Chiffrement SSE-KMS activé pour {bucket}.")

    s3_client.put_public_access_block(
        Bucket=bucket,
        PublicAccessBlockConfiguration={
            "BlockPublicAcls": True,
            "IgnorePublicAcls": True,
            "BlockPublicPolicy": False,
            "RestrictPublicBuckets": True
        }
    )
    print(f"Accès public bloqué pour {bucket}.")
```

Figure 47: Mise à jour du script generate_s3_bucket_with_bucket_back.py pour ajuster les accès acces au bucket

```python
def replicate_objects():
    print(f"\nRéplication manuelle {BUCKET_NAME} → {BUCKET_BACK}...")

    src_bucket = s3_resource.Bucket(BUCKET_NAME)
    dest_bucket = s3_resource.Bucket(BUCKET_BACK)

    for obj in src_bucket.objects.all():
        copy_source = {"Bucket": BUCKET_NAME, "Key": obj.key}
        dest_bucket.copy(copy_source, obj.key)
        print(f"Copié : {obj.key}")

    print("Réplication terminée.\n")

def apply_cloudtrail_bucket_policy():
    policy = {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Sid": "AWSCloudTrailAclCheck",
                "Effect": "Allow",
                "Principal": {"Service": "cloudtrail.amazonaws.com"},
                "Action": "s3:GetBucketAcl",
                "Resource": f"arn:aws:s3:::{BUCKET_BACK}"
            },
            {
                "Sid": "AWSCloudTrailWrite",
                "Effect": "Allow",
                "Principal": {"Service": "cloudtrail.amazonaws.com"},
                "Action": "s3:PutObject",
                "Resource": f"arn:aws:s3:::{BUCKET_BACK}/AWSLogs/767397842641/*",
                "Condition": {
                    "StringEquals": {
                        "s3:x-amz-acl": "bucket-owner-full-control"
                    }
                }
            }
        ]
    }

    s3_client.put_bucket_policy(
        Bucket=BUCKET_BACK,
        Policy=json.dumps(policy)
    )
    print(f"Politique CloudTrail appliquée au bucket {BUCKET_BACK}")
```

Figure 48 : Mise à jour du script generate_s3_bucket_with_bucket_back.py pour répliquer le bucket S3 vers le bucket S3 polystudents3-back et pour activer CloudTrail

```python
def setup_cloudtrail():
    print("Activation CloudTrail pour audit des objets S3...")

    trail_name = "S3ObjectTrail"

    try:
        cloudtrail.create_trail(
            Name=trail_name,
            S3BucketName=BUCKET_BACK,
            IsMultiRegionTrail=False
        )
        print("Trail créé.")

    except ClientError as e:
        if "TrailAlreadyExistsException" in str(e):
            print("ℹ Trail existe déjà.")
        else:
            raise e

    cloudtrail.put_event_selectors(
        TrailName=trail_name,
        EventSelectors=[
            {
                "ReadWriteType": "WriteOnly",
                "IncludeManagementEvents": False,
                "DataResources": [{
                    "Type": "AWS::S3::Object",
                    "Values": [f"arn:aws:s3:::{BUCKET_NAME}/*"]
                }]
            }
        ]
    )

    cloudtrail.start_logging(Name=trail_name)
    print("CloudTrail activé pour surveiller modifications/suppressions.\n")
```

Figure 49: Setup du cloudtrail afin de l'activer

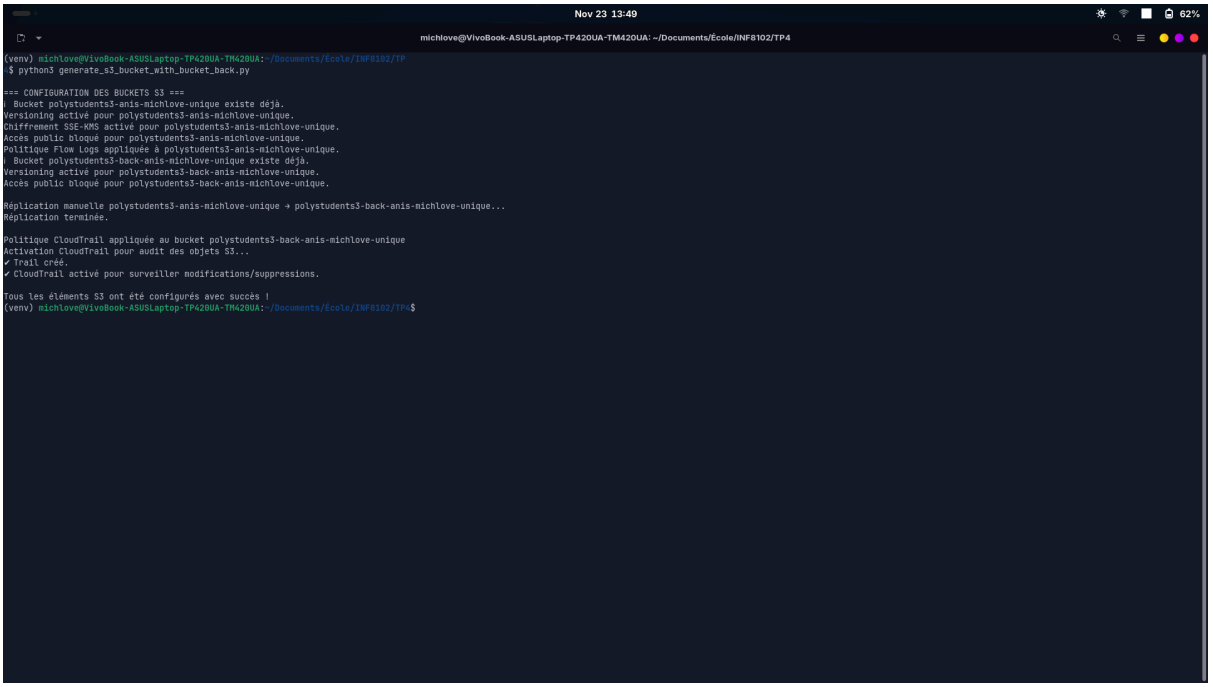


Figure 50: Exécution du script generate_s3_bucket_with_bucket_back.py

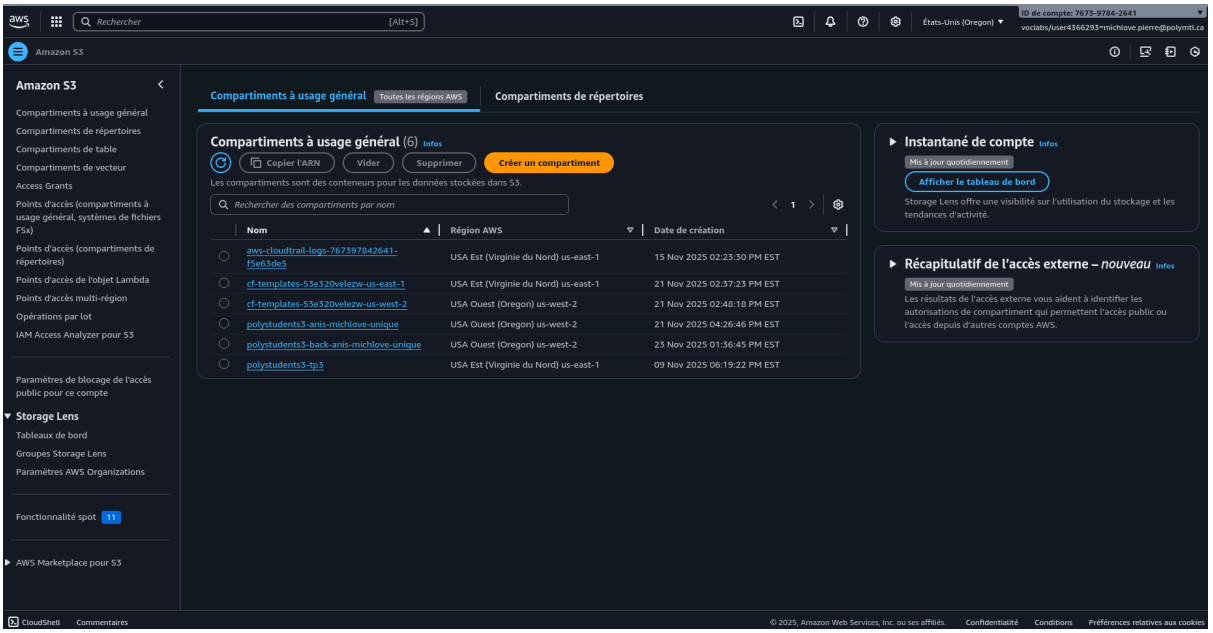


Figure 51: Démontrer que le bucket S3 back à bien été créer

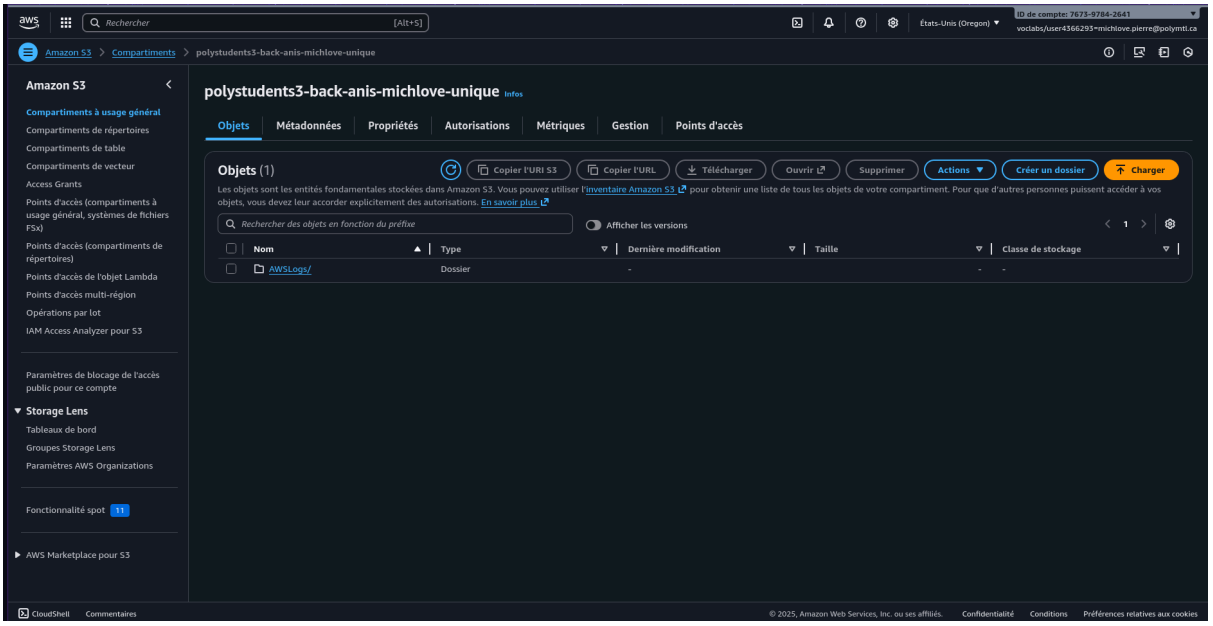


Figure 52: Démontrer que le cloudtrail est bien activé dans le bucket S3 back

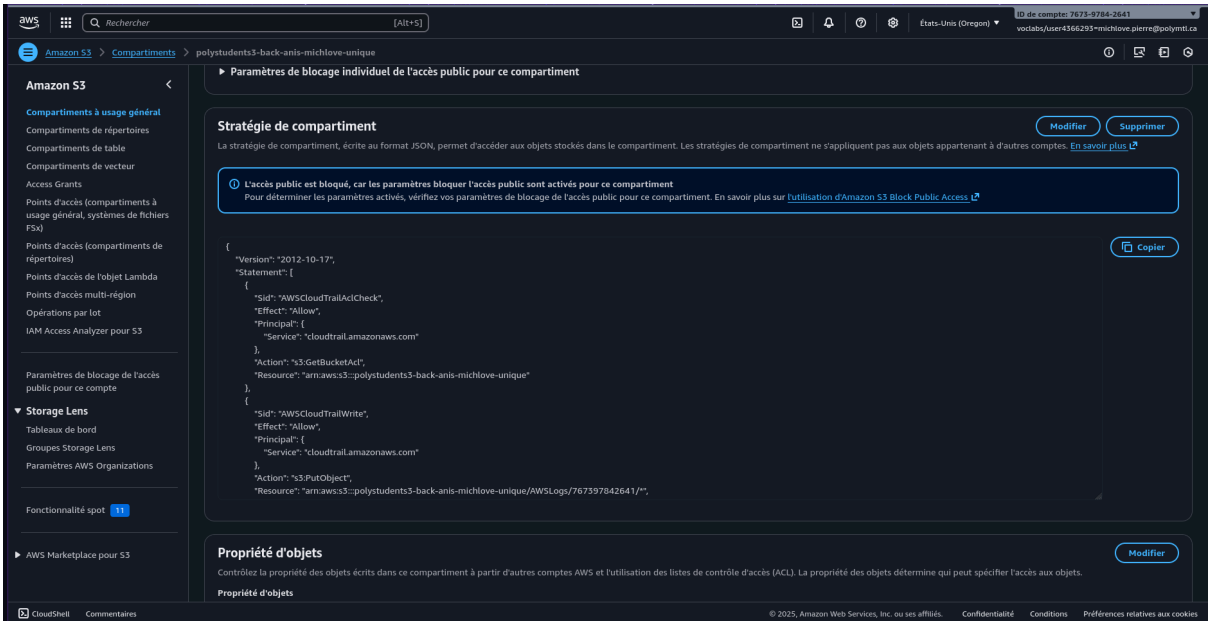


Figure 52: Preuve que le cloudtrail est bien activé dans le bucket S3 back

4.

Figure 53: Installation de trivy

Figure 54: Vérification de la version de trivy

4.1

```
michlove@VivoBook-ASUSLaptop-TP420UA-TM420UA:~/Documents/Ecole/INF8102/TP4/TP4_I
NF8102$ trivy fs . --scanners vuln,misconfig,secret --severity MEDIUM,HIGH,CRITICAL --format json --output scan_report.json
2025-12-11T00:36:34-05:00       INFO    [vuln] Vulnerability scanning is enabled
2025-12-11T00:36:34-05:00       INFO    [misconfig] Misconfiguration scanning is enabled
2025-12-11T00:36:35-05:00       INFO    [secret] Secret scanning is enabled
2025-12-11T00:36:35-05:00       INFO    [secret] If your scanning is slow, please try '--scanners vuln,misconfig' to disable secret scanning
2025-12-11T00:36:35-05:00       INFO    [secret] Please see https://trivy.dev/v0.67/docs/scanner/secret#recommendation for faster secret detection
2025-12-11T00:36:35-05:00       WARN    [cloudformation parser] Missing parameter values        file_path="tp4_vpc.yaml" parameters="EnvironmentName"
2025-12-11T00:36:35-05:00       INFO    Number of language-specific files       num=0
2025-12-11T00:36:35-05:00       INFO    Detected config files   num=1
```

Figure 55: Exécution de trivy sur le dossier du tp afin de générer le rapport

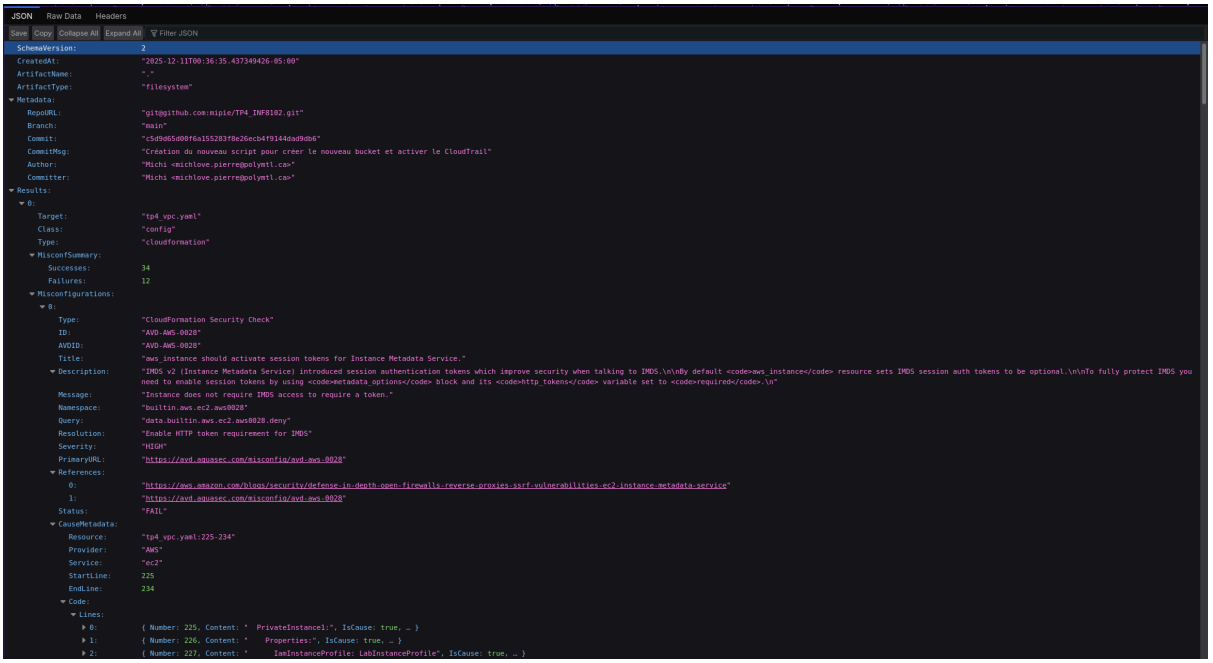


Figure 56: Preuve que le rapport à bien été généré

4.2

```
michlove@VivoBook-ASUSLaptop-TP420UA-TM420UA:~/Documents/Ecole/INF8102/TP4/TP4_INF8102$ jq '[.Results[].Vulnerabilities // [] | .[] | objects | select(.Severity? == "HIGH") | {Description: (.Description // .Message // "N/A"), Severity, CV
SSv3: (.CVSS?.nvd?.V3Score // .CVSS?.redhat?.V3Score // "N/A")}]' scan_report.json > cve_json
michlove@VivoBook-ASUSLaptop-TP420UA-TM420UA:~/Documents/Ecole/INF8102/TP4/TP4_INF8102$
```

Figure 57: Extraction des cve du rapport json

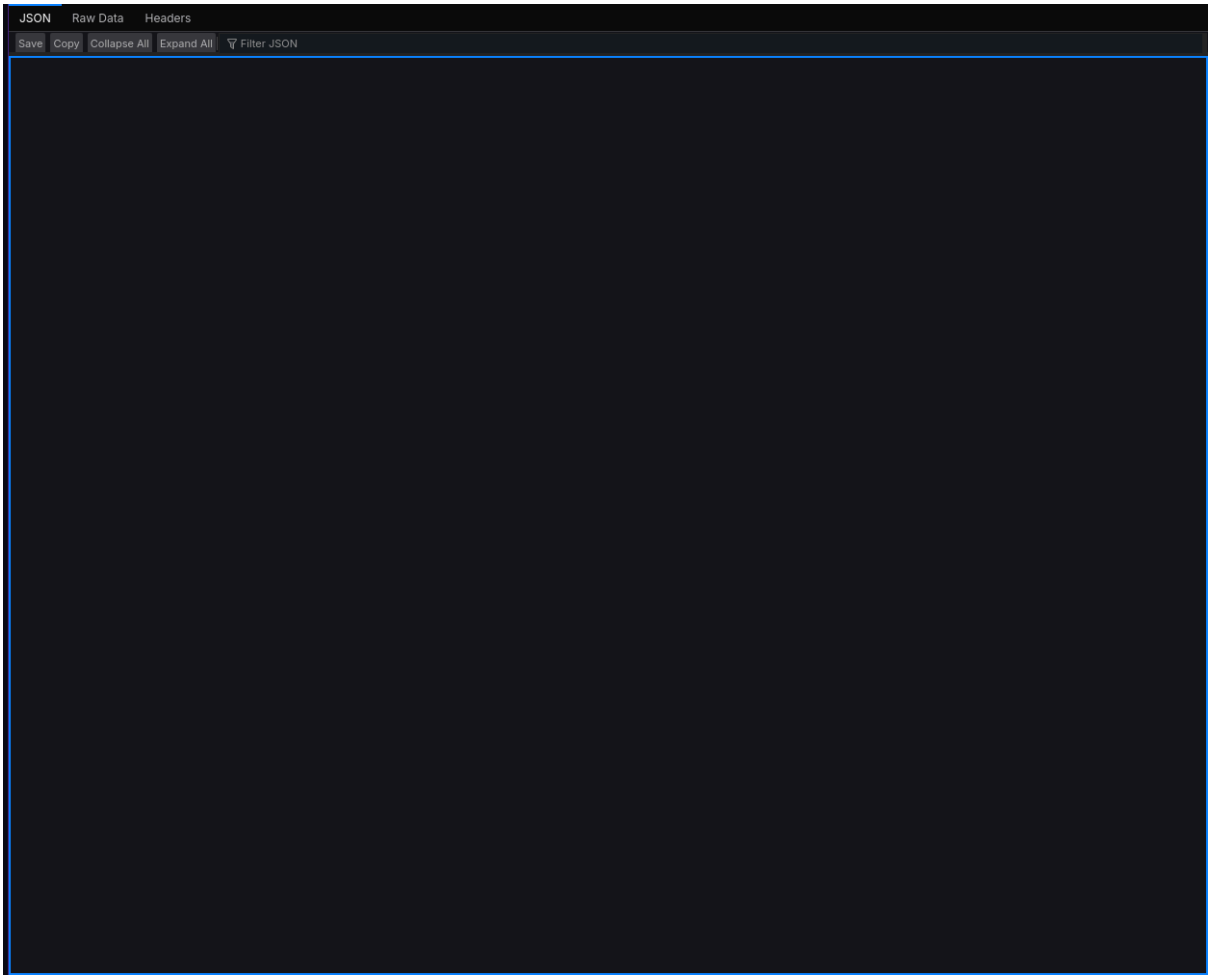


Figure 58: Preuve que le fichier cve.json a bien été créer

4.3

Selon les résultats obtenus, quatre vulnérabilités ont été détectées dans notre code IaC : AVD-AWS-0028, AVD-AWS-0107, AVD-AWS-0131 et AVD-AWS-0164. Les paragraphes suivants présentent, pour chacune de ces vulnérabilités, une mesure de sécurité appropriée permettant d'en assurer la mitigation.

En premier lieu, au niveau de la vulnérabilité AVD-AWS-0028, la solution serait d'activer les tokens de session en utilisant le paramètre *metadata_options* et en modifiant sa variable *http_tokens* à l'état *required*.

En deuxième lieu, au niveau de la vulnérabilité AVD-AWS-0107, la solution serait d'utiliser une unique adresse IP autorisée *x.x.x.x/x* sur la variable *CidrIp* et d'éviter l'utilisation de 0.0.0.0/0. Cette limitation assure qu'uniquement un utilisateur de confiance puisse atteindre les ports critiques tels que SSH (22) et RDP (3389).

En troisième lieu, au niveau de la vulnérabilité AVD-AWS-0131, la solution serait d'activer le chiffrement des volumes EBS pour chaque instance EC2. Afin d'y arriver, il faut ajouter le

paramètre *BlockDeviceMappings* en modifiant sa variable *Encrypted* à l'état *true*. Ainsi, au repos, les données des instances seraient encryptées et sécurisées.

En quatrième lieu, au niveau de la vulnérabilité AVD-AWS-0164, la solution serait de modifier le paramètre *MapPublicIpOnLaunch* à l'état *false*. Ainsi, cela permettrait d'éviter que le VPC attribue automatiquement aux instances des adresses IP publiques par défaut.

Voici le lien vers notre répo: https://github.com/mipie/TP4_INF8102.git

Références

https://avd.aquasec.com/misconfig/aws/ec2/avd-aws-0028/
https://avd.aquasec.com/misconfig/aws/ec2/avd-aws-0107/
https://avd.aquasec.com/misconfig/aws/ec2/avd-aws-0131/
https://avd.aquasec.com/misconfig/aws/ec2/avd-aws-0164/
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIEncryption.html
https://support.icompaas.com/support/solutions/articles/62000233593-ensure-vpc-subnets-do-not-assign-public-ip-by-default/