INF8102: Cloud Security

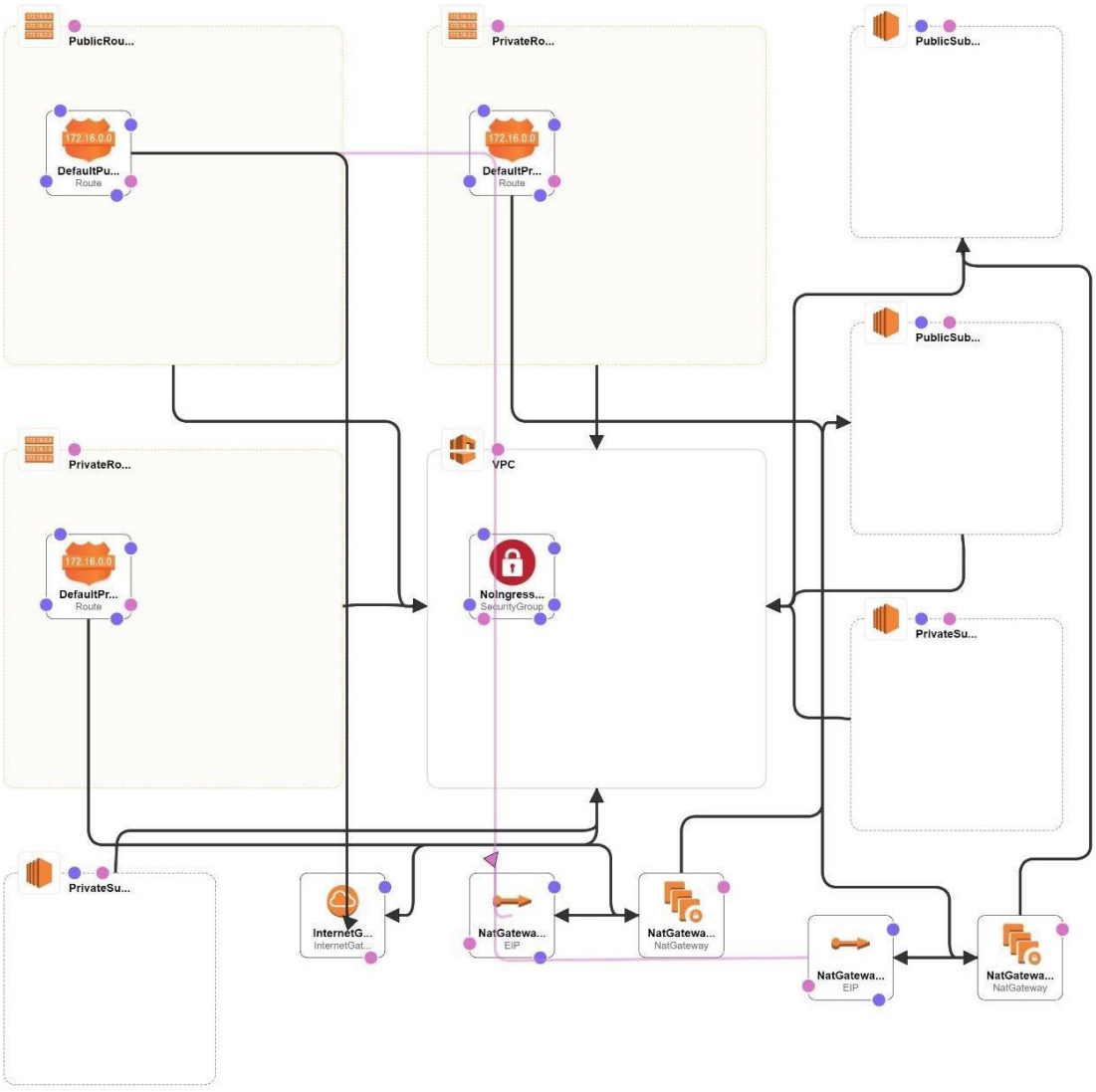
TP 4: Infrastructure as Code Security

Session: Fall 2024

Département de Génie Informatique and Génie Logiciel

Infrastructure as Code (IaC) Security

Let's consider the previous VPC infrastructure *polystudentlab-vpc* in TP 3 consisting of two availability zones AZ1 and AZ2. Each availability zone has a security group that controls the ingress network traffic and NAT gateways for outbound internet access, with routes from private subnets set to use it.



VPC Security

Let's build the architecture in TP 3 using CloudFormation. Create a file named *vpc.yaml* and add the following line to create the VPC with network address *10.0.0.0/16*.

N.B: You can rename the VPC *polystudent-vpc* to *polystudent-vpc1* if the name already exists.

```
  GNU nano 5.9                          test.yaml                          Modified
#   Create a VPC with:
#       2 Public Subnets
#       2 Private Subnets
#   An Internet Gateway (with routes to it for Public Subnets)
#   A NAT Gateway for outbound access (with routes from Private Subnets set to use it)

Description:  This deploys a VPC, with a pair of public and private subnets spread
  across two Availability Zones. It deploys an internet gateway, with a default
  route on the public subnets. It deploys a pair of NAT gateways (one in each AZ),
  and default routes for them in the private subnets.

Parameters:
  EnvironmentName:
    Description: environment is prefixed to resource names
    Type: String

  VpcCIDR:
    Description: VPC polystudent-vpc
    Type: String
    Default: 10.0.0.0/16

Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCIDR
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: !Ref EnvironmentName

Outputs:
  VPC:
    Description: A reference to the created VPC
    Value: !Ref VPC
```

Fig. 1: VPC creation – part 1

Now, let's create private and public subnets (AZ1, AZ2) per availability zone. In the block *Parameters* of the YAML file (see Fig. 1), add the following lines

```
  PublicSubnet1CIDR:
    Description: public subnet in Availability Zone 1
    Type: String
    Default: 10.0.0.0/24

  PublicSubnet2CIDR:
    Description: public subnet in Availability Zone 2
    Type: String
    Default: 10.0.16.0/24

  PrivateSubnet1CIDR:
    Description: private subnet in Availability Zone 1
    Type: String
    Default: 10.0.128.0/24

  PrivateSubnet2CIDR:
    Description: private subnet in Availability Zone 2
    Type: String
    Default: 10.0.144.0/24
```

Fig. 2: VPC creation – part 2

In block *Resources* (see Fig. 1), add the following lines to create each public and private subnet following the same process in Fig.2,

Fig. 3: VPC creation – part 3

In the block *Outputs* (see Fig. 1), kindly add the following lines to render private and public subnets in Cloudformation

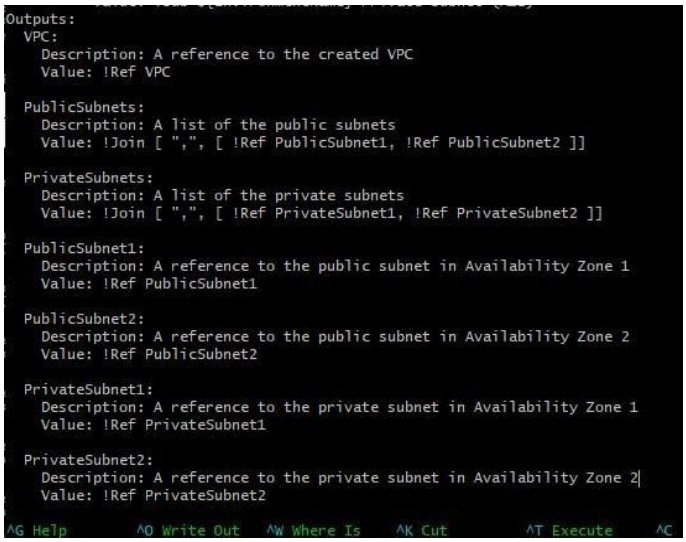


Fig. 4: VPC creation – part 4

Next, let's create an internet gateway and attach it to *polystudent-vpc*. In the block *Resources* (see Fig. 1), add the following lines

```
InternetGateway:
  Type: AWS::EC2::InternetGateway
  Properties:
    Tags:
      - Key: Name
        Value: !Ref EnvironmentName

InternetGatewayAttachment:
  Type: AWS::EC2::VPCGatewayAttachment
  Properties:
    InternetGatewayId: !Ref InternetGateway
    VpcId: !Ref VPC
```

Fig. 5: VPC creation – part 5

Now, let's add a NAT Gateway per availability zone to enable internet access in private subnets. In the block *Resources* (see Fig. 1), add the following lines

```
  NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc

  NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc

  NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway1EIP.AllocationId
      SubnetId: !Ref PublicSubnet1

  NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway2EIP.AllocationId
      SubnetId: !Ref PublicSubnet2
```

Fig. 6: VPC creation – part 6

Next, let's create a routing table and attached it to public subnets. In the block *Resources* (see Fig. 1), add the following lines

```
  DefaultPublicRoute:
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1

  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execu
```

Fig. 7: VPC creation – part 7

Now, let's create two routing tables and attach them to private subnets. Each route is added from private subnets to NAT gateways for outbound internet access. In the block *Resources* (see Fig. 1), add the following lines

```
  PrivateRouteTable1:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Routes (AZ1)

  DefaultPrivateRoute1:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1

  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1

  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Routes (AZ2)

  DefaultPrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2
```

Fig. 8: VPC creation – part 8

Next, let's create a security group *polystudent-sg*. Add new rules to *polystudent-sg* that authorize specific ports such as SSH (22), HTTP (80), HTTPS (443), DNS (tcp/udp 53), MSSQL (1433), PostgreSQL (5432), MySQL (3306), RDP (3389), OSSEC (1514) and ElasticSearch (9200-9300).

```
IngressSecurityGroup:
   Type: AWS::EC2::SecurityGroup
   Properties:
     GroupName: "polystudent-sg"
     GroupDescription: "Security group allows SSH, HTTP, HTTPS, MSSQL,etc..."
     VpcId: !Ref VPC
     SecurityGroupIngress:
       - IpProtocol: tcp
         FromPort: 22
         ToPort: 22
         CidrIp: 0.0.0.0/0
       #Add more rules for HTTP, HTTPS, MSSQL, etc....
```

Fig. 9: VPC creation – part 9

Now, open AWS Cloudformation and load the YAML file in the stack to automatically generate a secure VPC infrastructure. You will obtain the following result,

Ressources (22)

Rechercher des ressources

ID logique	ID physique	Type	Statut	Moti
DefaultPrivateRoute1	AWSte-Defau-1N0P9QEY62BZU	AWS::EC2::Route	CREATE_COMPLETE	-
DefaultPrivateRoute2	AWSte-Defau-I42637JGSN3M	AWS::EC2::Route	CREATE_COMPLETE	-
DefaultPublicRoute	AWSte-Defau-14CZXN1QWRF2Q	AWS::EC2::Route	CREATE_COMPLETE	-
InternetGateway	igw-0aae9c89946a985ed	AWS::EC2::InternetGateway	CREATE_COMPLETE	-
InternetGatewayAttachment	AWSte-Inter-ZDZG8VG06W38	AWS::EC2::VPCGatewayAttachment	CREATE_COMPLETE	-
NatGateway1	nat-03293bfb4d0b159f0	AWS::EC2::NatGateway	CREATE_COMPLETE	-
NatGateway1EIP	34.231.216.169	AWS::EC2::EIP	CREATE_COMPLETE	-
NatGateway2	nat-0d473bcd696614285	AWS::EC2::NatGateway	CREATE_COMPLETE	-
NatGateway2EIP	44.208.110.207	AWS::EC2::EIP	CREATE_COMPLETE	-
NoIngressSecurityGroup	sg-048d35439dccc3423	AWS::EC2::SecurityGroup	CREATE_COMPLETE	-
PrivateRouteTable1	rtb-0577e0f1bb19ab09e	AWS::EC2::RouteTable	CREATE_COMPLETE	-
PrivateRouteTable2	rtb-058654341507ee1e2	AWS::EC2::RouteTable	CREATE_COMPLETE	-
PrivateSubnet1	subnet-0d9c1673525c11ce4	AWS::EC2::Subnet	CREATE_COMPLETE	-
PrivateSubnet1RouteTableAssociation	rtbassoc-0cf03d234010d7179	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE	-
PrivateSubnet2	subnet-0d8c4ab539df5df30	AWS::EC2::Subnet	CREATE_COMPLETE	-

EC2 Security

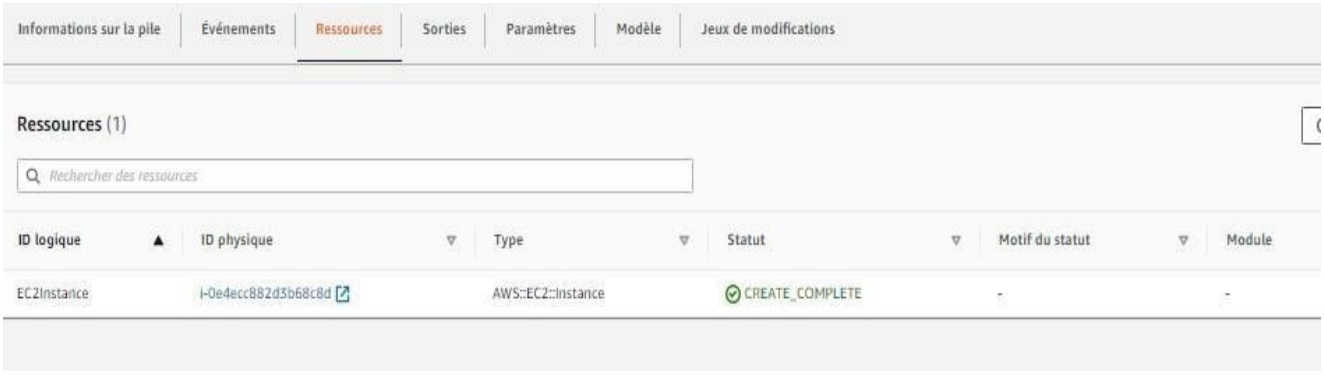
This section shows a basic example of how to secure an EC2 instance with a security group and a key pair for secure remote access. The keypair *polystudent-keypair* created in the previous TPs can be reused as well as the security group *polystusdent-sg*. You can also use an existing public subnet ID. Create a file named *ec2.json* and add the following line to create the EC2 instance

```
#   Run an secure EC2 Instance with:
#       - security group
#       - IAM role

{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "Deploy a secure EC2 instance on the public subnet of AZ1 ",
  "Resources": {
  "EC2Instance": {
    "Type" : "AWS::EC2::Instance",
    "Properties" : {
       "KeyName": "polystudent-keypair",
       "InstanceType": "t2.micro",
       "ImageId": "ami-0ff8a91507f77f867",
       "AvailabilityZone": "ca-central-1a",
       "SecurityGroupIds" : ["polystudent-sg"],
       "SubnetId": "subnet-03b7e0e782de6ee1c",
       "IamInstanceProfile" : "LabRole",
       "BlockDeviceMappings" : [
          {
            "DeviceName" : "/dev/sda1",
            "Ebs" : {
              "DeleteOnTermination" : "false",
              "VolumeSize" : "80"
            }
          }
        ]
      }
    }
  }
}
```

Fig. 10: EC2 creation

Now, open AWS Cloudformation and load your YAML file (see Fig. 10) in the stack to automatically generate a secure EC2 instance



S3 Security

This section shows a basic example of how to secure an S3 bucket using a deletion policy, private access control, encryption at rest, and versioning for backup.

Create a file named *s3.json* and add the following line to create an S3 bucket *polystudents3* with KMS key *polystudent-kms1*. ACL is set to private, and versioning is enabled to allow replication. You must also change the KMS Key ID with the proper ARN.

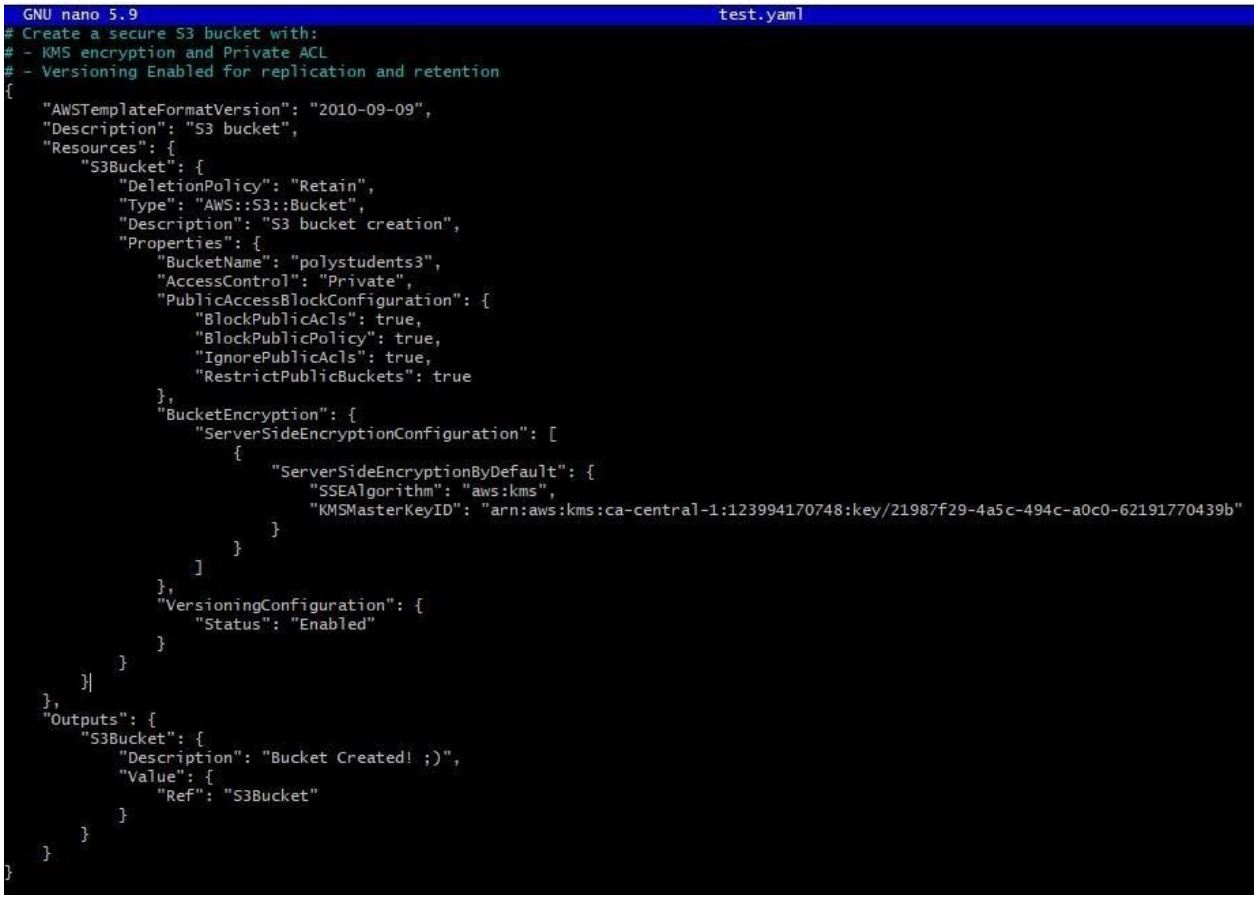
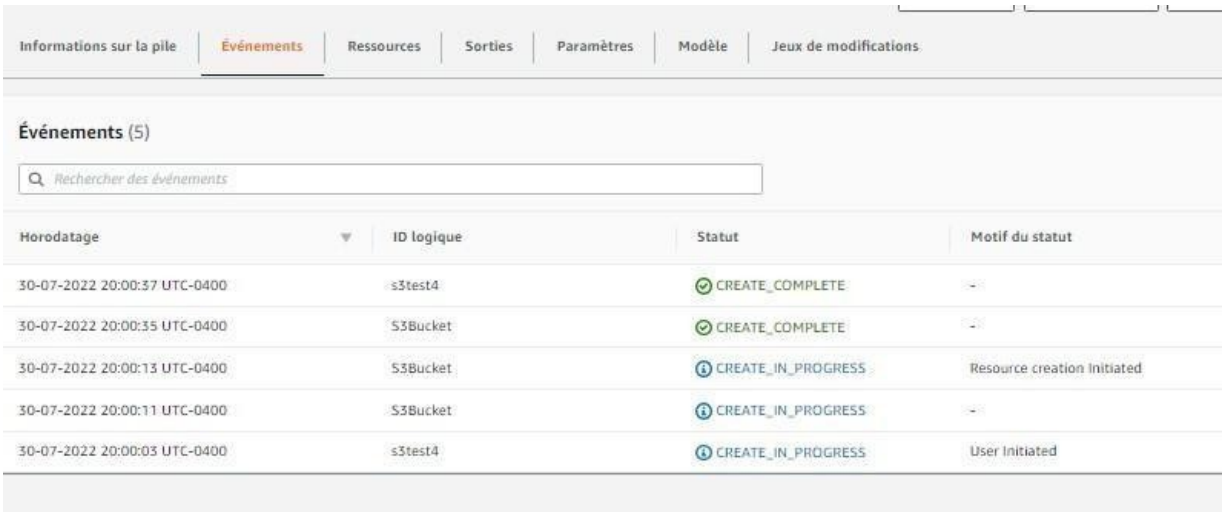


Fig. 11: S3 creation

Replace the *KMSMasterKeyID* with the arn of the KMS key *polystudent-kms1*. Now, load the JSON file (see Fig. 11) in the stack to generate the S3 bucket,



IaC Code Security

This section shows how to check security issues in the code using vulnerability scanning tools such as Trivy and TFsec.

Create a folder *polylab* and copy your IaC configuration files (JSON, YAML) to the directory. Run the following command to perform a vulnerability scan on the IaC source code

```
trivy fs --security-checks vuln,secret,config  polylab/
```

N.B: please, check how to install Trivy in TP 3

Exercise (100 pts)

1. Reproduce the VPC example below (see Fig. 1-9) and generates the infrastructure using Python. (15 pts)

   N.B: You can use libraries such as boto3, troposphere, or cdktf. You can also rename existing names to avoid conflicts. The code must be tested with a proof.

2. Reproduce the S3 bucket *polystudens3* in the example below (see Fig. 11) and generates the IaC service using Python. (10 pts)

   N.B: You can use Python libraries such as boto3, troposphere, or cdktf. You can also rename existing names to avoid conflicts. The code must be tested with a proof.

3. Let us consider the IaC code of the VPC example below.

   3.1. Modify the code in the VPC example to support VPC flow logs. Note that only rejected packets can be captured and sent to the S3 bucket *polystudens3* (see Question 2). (10 pts)

   N.B: You can only do it either in Bash or Python. The code must be tested with a proof.

   3.2. The generated VPC has 2 public instances on AZ1 and AZ2, and 2 private instances on AZ1 and AZ2. Update the code of EC2 instances in the VPC example to support the IAM role *LabRole*, and a CloudWatch alarm that controls the ingress number of packets on *all instances*. The average threshold is 1000 pkts/sec. (30 pts)

   N.B: You can only do it either in Bash or Python. The code must be tested with a proof.

   3.3. Update the code of the S3 bucket *polystudens3* in Question 2 so that

   (1) the bucket is replicated to a destination S3 bucket *polystudents3-back* (10 pts)
   (2) Cloudtrail is enabled to log object modification/deletion activities on the bucket (10 pts)

   N.B: You can only do it either in Bash or Python. The code must be tested with a proof.

4. Use Trivy to perform a vulnerability scanning on your IaC code
   4.1. Generate scan report of vulnerabilities with medium, high, and critical severity (5 pts)
   4.2. Extract Description, CVSSv3, and high severity using the *jq* command and stores it in the file *cve.json* (5 pts)
   4.3. Describe 5 security measures to mitigate vulnerabilities in the IaC code (5 pts)
   N.B: To install Trivy and use it, please check TP 3. The scan must be tested with a proof.