TP 4: Infrastructure as a code

INF8102 : Sécurité infonuagique
Département des génies logiciel et informatique

TALABONG TEMGOUA WILLY HERMAN 2351362
LAMKINSI AMINE 2078728

Polytechnique Montréal
Novembre 2025

lien Github : https://github.com/willytalabong-commits/INF8102-TP4-G9

1. Génération de l’infrastructure avec Python

1.1 Objectif de la question

L’objectif de cette première question du TP4 est de reproduire l’architecture VPC présentée dans la figure 1-9 (VPC multi-AZ avec subnets publics/privés, NAT Gateways, Internet Gateway, route tables et security group), puis de générer automatiquement cette infrastructure à l’aide d’un script Python, en utilisant une bibliothèque d’Infrastructure-as-Code (IaC).

Dans le cadre de ce travail, j’ai choisi d’utiliser la bibliothèque Troposphere, qui permet de générer des templates AWS CloudFormation en Python, combinée avec boto3 pour déployer le stack dans AWS.

Ce choix s'inscrit dans les principes infonuagique vus au cours :

- Automatisation de l’infrastructure (IaC)
- Reproductibilité de l’environnement
- Sécurité par construction (définition centralisée des SG, routes, NAT, subnets)

1.2 Démarche et étapes réalisées

Étape 1 : Configuration de l’environnement Python

Un environnement virtuel Python a été créé, puis les dépendances suivantes ont été installées :

- `troposphere` (génération de templates CloudFormation)
- `boto3` (interaction AWS)
- `botocore`

Commande utilisée :

```
python -m venv venv
source venv/bin/activate
pip install troposphere boto3
```

Cela permet de travailler dans un environnement isolé et reproductible.

Étape 2 : Définition de l'architecture dans Troposphere

J'ai ensuite développé un script Python nommé **`deploiment_vpc_iac.py`**.
Ce script génère automatiquement une template CloudFormation représentant l'architecture demandée :

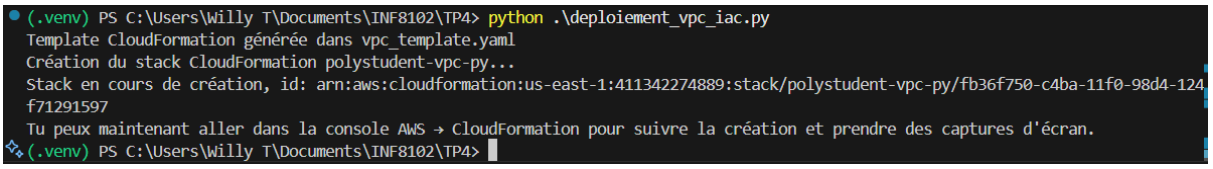
- **VPC `10.0.0.0/16`**
- **4 subnets** (public/privé dans AZ1 et AZ2)
- **1 Internet Gateway** (IGW)
- **2 NAT Gateways** (une dans chaque zone de disponibilité)
- **1 route table publique**, associée aux deux subnets publics
- **2 route tables privées**, chacune associée à un subnet privé
- **1 Security Group** conforme aux ports définis dans la figure 1-9
    - ports : 22, 80, 443, 53, 3306, 5432, 1433, 3389, 1514, 9200-9300

Toutes ces ressources sont définies programmaticalement, ce qui garantit une cohérence et une réutilisabilité totale de l'infrastructure.

Étape 3 : Génération locale de la template CloudFormation

L'exécution du script génère automatiquement un fichier : `vpc_template.yaml`

Ce fichier constitue la preuve IaC et peut être réinjecté dans CloudFormation à tout moment pour reconstruire la même architecture.



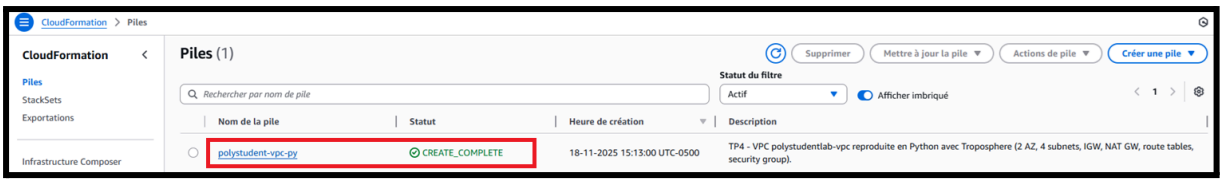
Étape 4 : Déploiement automatique avec boto3 + CloudFormation

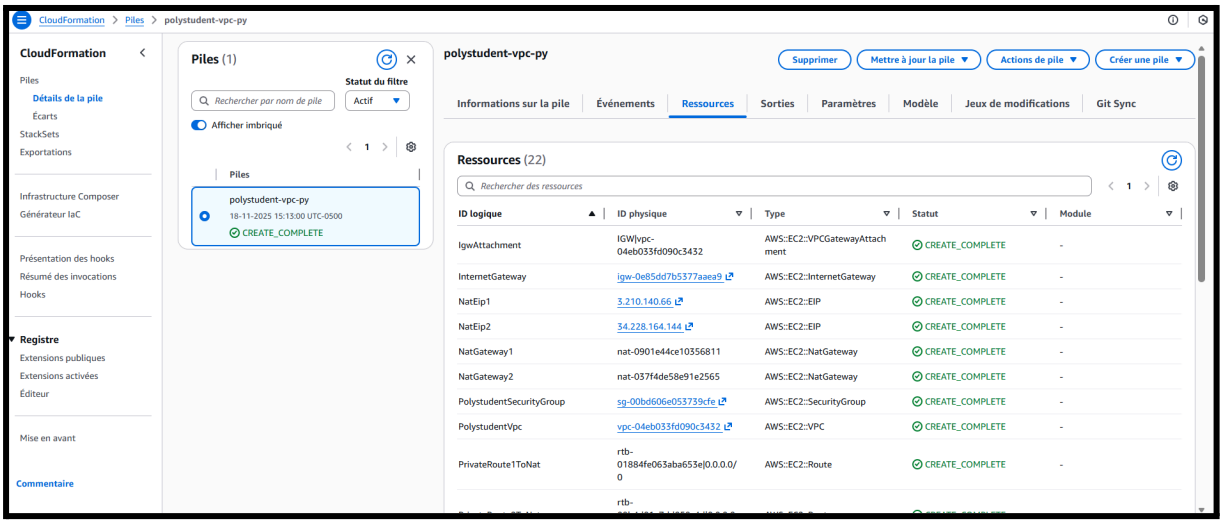
Le script Python déclenche ensuite la création du stack sur AWS : `python deploiment_vpc_iac.py`

Le script utilise la fonction boto3 :

```
cf.create_stack(
    StackName="polystudent-vpc-py",
    TemplateBody=template_body,
    Capabilities=["CAPABILITY_NAMED_IAM"]
)
```

Cette commande déclenche la création automatique de toutes les ressources.





Étape 5 : Validation du déploiement (preuve)

Une fois le stack créé, j'ai validé la présence de l'ensemble des ressources dans la console AWS.

Les éléments suivants ont été vérifiés :

✔ VPC créée

- CIDR : `10.0.0.0/16`

✔ Subnets

- PublicSubnet1 (AZ1) – `10.0.0.0/24`
- PrivateSubnet1 (AZ1) – `10.0.1.0/24`
- PublicSubnet2 (AZ2) – `10.0.2.0/24`
- PrivateSubnet2 (AZ2) – `10.0.3.0/24`

✔ Route Tables

- Public RT -> IGW
- Private RT1 -> NAT GW1
- Private RT2 -> NAT GW2

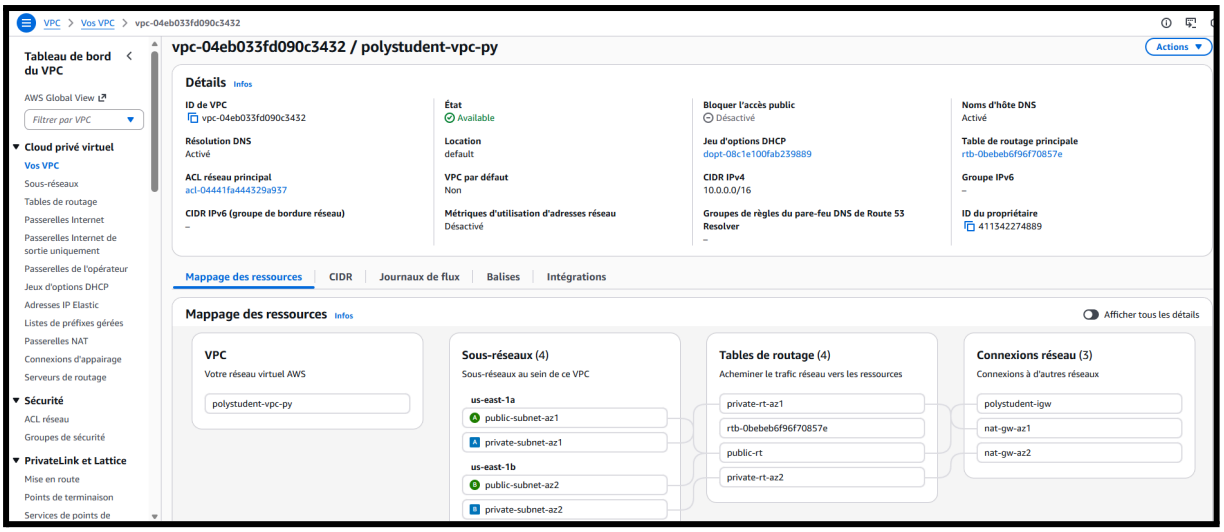
✔ Internet Gateway attachée à la VPC

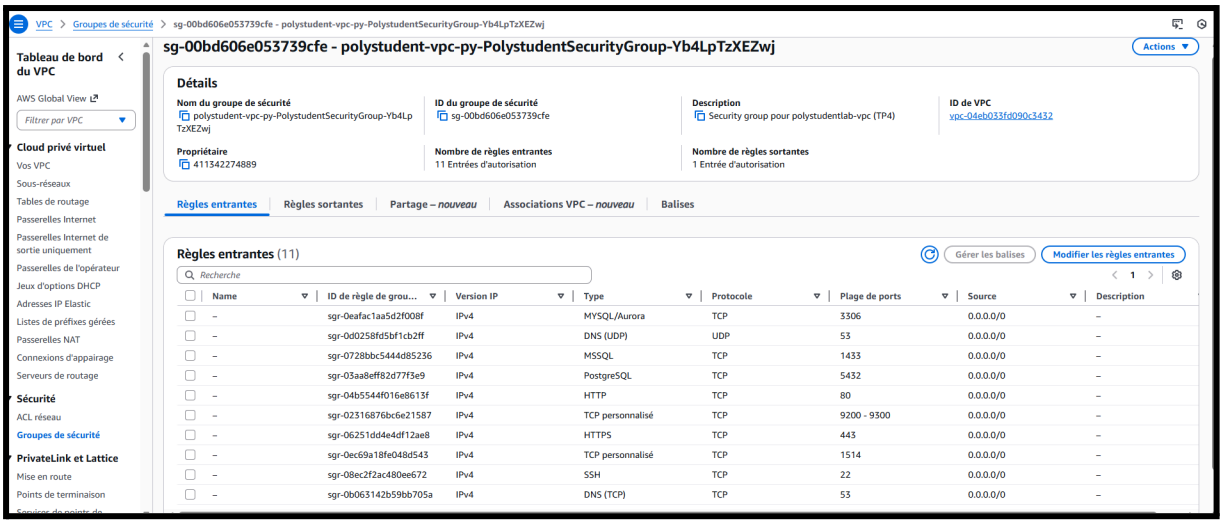
✔ NAT Gateways

- NAT AZ1
- NAT AZ2

✔ Security Group polystudent-sg

Avec les ports :
22, 80, 443, 53 TCP/UDP, 3306, 5432, 1433, 3389, 1514, 9200-9300





1.3 Code complet utilisé

Le code complet utilisé pour générer et déployer l'architecture est disponible dans le fichier : `deploiment_vpc_iac.py`

Il contient :

- la génération de la template CloudFormation via Troposphere
- la sauvegarde en YAML
- le déploiement automatique avec boto3

Ce script constitue la preuve du travail automatisé et reproductible demandé dans la question 1.

Cette première question m'a permis de :

- Transformer une architecture réseau AWS en un modèle IaC Python,
- Automatiser la création d'un environnement cloud complet,
- Appliquer les bonnes pratiques infonuagiques d'infrastructure reproductible,
- Tester l'infrastructure dans CloudFormation et fournir les preuves requises.

L'ensemble prouve que l'infrastructure a bien été générée par Python + Troposphere + boto3, conformément aux exigences du TP4.

2. Reproduction du bucket S3 et génération de l'infrastructure IaC en utilisant Python

2.1 Objectif de la question

L'objectif de cette deuxième question du TP4 est de reproduire en Infrastructure-as-Code (IaC) le bucket S3 *polystudents3* présenté dans la Figure 11. Ce bucket constitue un élément central de la solution cloud sécurisée étudiée dans le cours, notamment pour la journalisation et le stockage sécurisé.

Dans le cadre de ce TP, j'ai utilisé Troposphere (pour générer un template CloudFormation en Python) et boto3 (pour déployer automatiquement ce template dans AWS).

Le bucket a été adapté à mon groupe sous le nom : `polystudents3-group9`

2.2 Développement du service IaC en Python

Le travail a été réalisé dans un script nommé : **`deploiement_s3_iac.py`**

Ce script effectue les étapes suivantes :

1. Génération de la template CloudFormation

Le script utilise **Troposphere** pour :

- Définir un paramètre `KmsKeyArn`

- Créer un bucket avec :
    - `BucketName = polystudents3-group9`
    - `AccessControl = Private`
    - `VersioningConfiguration = Enabled`
    - `BucketEncryption = SSE-KMS`
    - `KMSMasterKeyID = arn:aws:kms:us-east-1:411342274889:key/a39fe93a-6789-433f-80fe-ef2d26cbf21a`
- Ajouter une DeletionPolicy = Retain

La template est automatiquement sauvegardée localement dans : `s3_template.yaml`

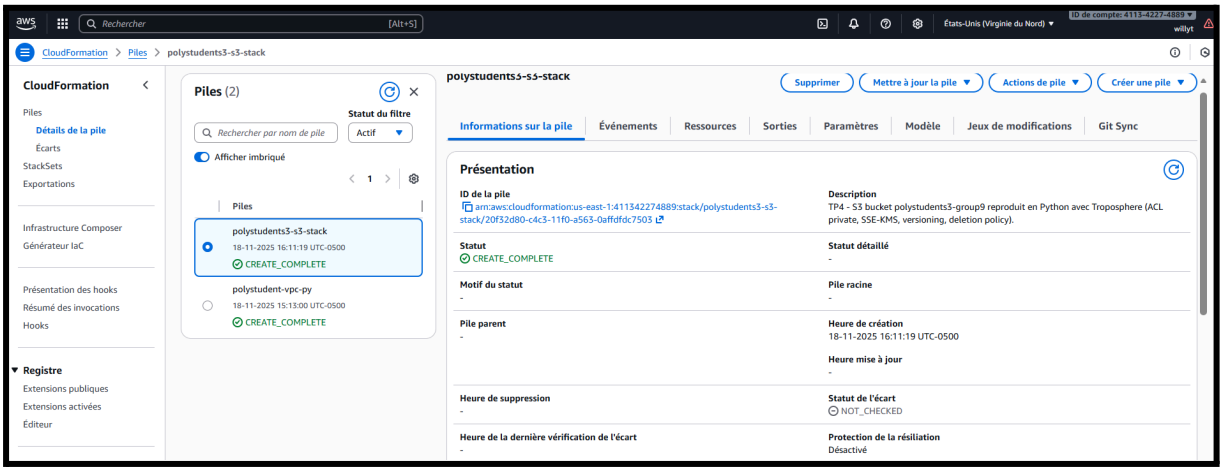
```
(.venv) PS C:\Users\Willy T\Documents\INF8102\TP4> python .\deploiement_s3_iac.py
Template S3 CloudFormation générée dans s3_template.yaml
Création du stack CloudFormation polystudents3-s3-stack...
Stack en cours de création, id : arn:aws:cloudformation:us-east-1:411342274889:stack/polystudents3-s3-stack/20f32d80-c4c3-11f0-a563-0affdfdc7503
Va dans la console AWS → CloudFormation pour vérifier l'état du stack et prendre des captures d'écran comme preuve pour le TP.
(.venv) PS C:\Users\Willy T\Documents\INF8102\TP4>
```

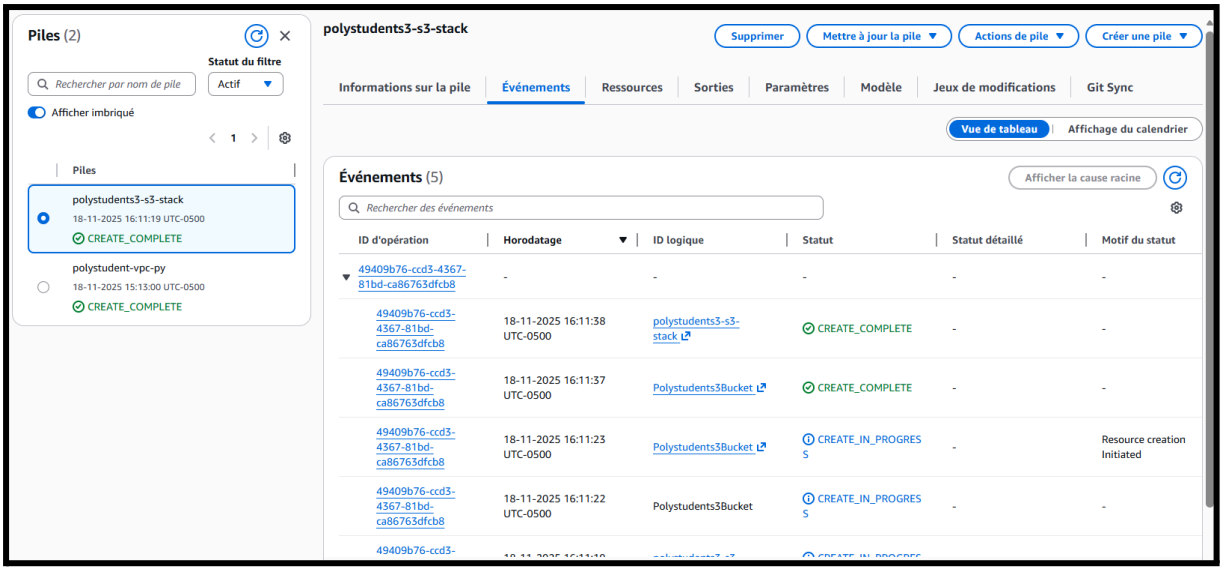
2. Déploiement du stack CloudFormation avec boto3

Le script appelle ensuite :

```
cf.create_stack(
    StackName="polystudents3-s3-stack",
    TemplateBody=template_body,
    Parameters=[...]
)
```

Ce qui déclenche la création du bucket géré entièrement comme IaC.



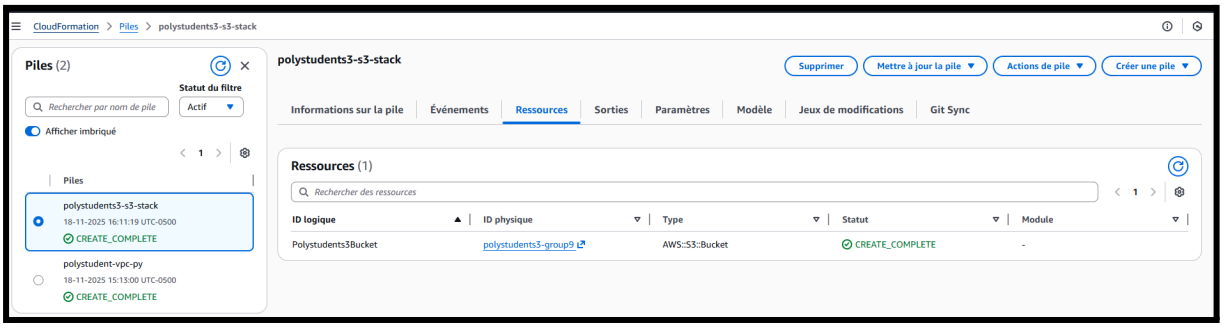


2.4 Validation et preuve de fonctionnement

Après exécution du script, les éléments suivants ont été vérifiés dans AWS :

Stack CloudFormation créé avec succès

- Nom : **polystudents3-s3-stack**
- Statut : **CREATE_COMPLETE**
- Ressource créée : `Polystudents3Bucket`



Cette question m'a permis de :

- Reproduire un service AWS S3 sécurisé à l'aide d'Infrastructure-as-Code,
- Automatiser entièrement la création du bucket avec Python,
- Appliquer les bonnes pratiques cloud vues en cours :
  - chiffrement KMS,
  - access control strict,
  - conservation des données,
- Valider le déploiement via CloudFormation.

Le script `deploiement_s3_iac.py` ainsi que la template `s3_template.yaml` démontrent que la solution a bien été générée et testée automatiquement, conformément aux exigences du TP4.

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)

L'objectif de cette section est de modifier le code IaC de la VPC (Question 1) afin d'activer les VPC Flow Logs, en configurant la VPC pour qu'elle envoie uniquement les paquets rejetés (REJECT) vers le bucket S3 sécurisé créé à la Question 2, soit : **`polystudents3-group9`**

La solution doit être développée exclusivement en Python, testée et accompagnée de preuves de déploiement.

Pour satisfaire cette exigence, j'ai intégré la ressource AWS suivante dans une template CloudFormation générée via Troposphere : `AWS::EC2::FlowLog`

Cette ressource permet de capturer le trafic réseau d'une VPC et d'exporter les logs dans un service cible (S3, CloudWatch Logs, etc.).
Dans notre cas, la configuration imposée est :

- **ResourceType : VPC**
- **TrafficType : REJECT**
- **LogDestinationType : S3**
- **LogDestination : arn:aws:s3:::polystudents3-group9**

L'ensemble a été automatisé dans un script Python nommé : **`deploiement_vpc_flowlogs_iac.py`**

Le script réalise les opérations suivantes :

1. **Définition de la VPC** (mêmes paramètres que la Question 1).
2. **Création d'un subnet public**, d'une route table et d'une Internet Gateway (infrastructure minimale nécessaire au test).

3. **Ajout de la ressource `AWS::EC2::FlowLog`** pour activer la journalisation du trafic REJECT vers le bucket S3.
4. **Génération automatique** d'un fichier IaC : **`vpc_flowlogs_template.yaml`**
5. **Déploiement via boto3** d'un stack CloudFormation : **`polystudent-vpc-flowlogs-stack`**

Ce stack met en place la VPC et active les Flow Logs sur celle-ci.

```
(.venv) PS C:\Users\Willy T\Documents\INF8102\TP4> python .\deploiement_vpc_flowlogs_iac.py
Template VPC + Flow Logs générée dans vpc_flowlogs_template.yaml
Création du stack CloudFormation polystudent-vpc-flowlogs-stack...
Stack en cours de création, id : arn:aws:cloudformation:us-east-1:411342274889:stack/polystudent-vpc-flowlogs-stack/f40241d0-c4c7-11f0-9180-0e6c5a76bbb9
Va dans la console AWS → CloudFormation pour vérifier l'état du stack et prends des captures d'écran pour le rapport (stack + VPC Flow Logs + S3).
(.venv) PS C:\Users\Willy T\Documents\INF8102\TP4>
```

Après exécution du script :

python deploiement_vpc_flowlogs_iac.py

le stack CloudFormation est créé avec succès.

✔ 1. Vérification dans CloudFormation

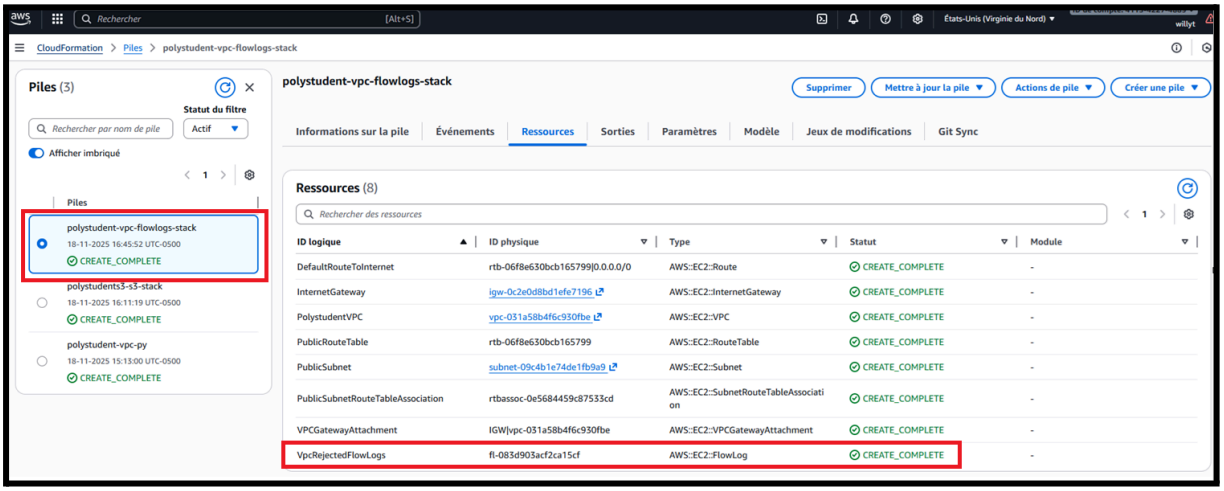
Le stack : **polystudent-vpc-flowlogs-stack**

apparaît avec le statut :

- **CREATE_COMPLETE**

et contient notamment la ressource :

- **VpcRejectedFlowLogs – AWS::EC2::FlowLog**

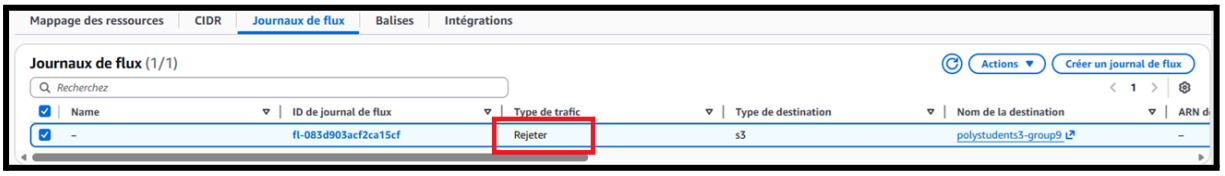


✔ 2. Vérification dans la console VPC

Dans la VPC créée, onglet Flow logs, une ligne apparaît avec :

- **Filter : REJECT**
- **Destination type : S3**
- **Bucket : polystudents3-group9**

Cela confirme que les paquets rejetés seront exportés automatiquement vers le bucket configuré.



✔ 3. Vérification dans S3

Dans le bucket **polystudents3-group9**, un dossier avec des préfixes du type :

`AWSLogs/411342274889/vpcflowlogs/…` se crée dès que du trafic REJECT est généré.

Cette question m'a permis d'intégrer de manière automatisée les VPC Flow Logs à une VPC AWS en utilisant exclusivement du code Python. Le déploiement IaC assure une solution reproductible, sécurisée et conforme aux exigences de l'énoncé :

- capture uniquement des paquets rejetés,
- export vers un bucket S3 sécurisé,
- déploiement automatisé via Troposphere + boto3.

Les captures d'écran fournies démontrent que la configuration a été correctement déployée et testée.

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)

Cette question consiste à étendre le code IaC de la VPC afin de déployer automatiquement :

- 4 instances EC2 :

    - 2 publiques (AZ1 et AZ2)
    - 2 privées (AZ1 et AZ2)
- Chacune associée au rôle IAM LabRole via un Instance Profile
- Une alarme CloudWatch par instance sur la métrique :
    - NetworkPacketsIn
    - Seuil moyen : 1000 paquets/sec
    - Période : 60 secondes

L'ensemble doit être réalisé uniquement en Python (Troposphere + boto3) et testé dans AWS.

Démarche et conception IaC

Afin de satisfaire les exigences, j'ai développé un script IaC nommé :
deploiement_ec2_alarms_iac.py

Ce script repose sur les bibliothèques Troposphere (génération de template CloudFormation) et boto3 (déploiement automatique).
Il réalise les opérations suivantes :

a) Création de la VPC et des subnets

La VPC générée contient :

- 2 subnets publics (AZ1 : 10.0.0.0/24, AZ2 : 10.0.1.0/24)
- 2 subnets privés (AZ1 : 10.0.2.0/24, AZ2 : 10.0.3.0/24)
- Une Internet Gateway et une route table publique

Ces composants permettent d'exécuter les instances dans leur zone respective.

b) Définition du rôle IAM pour les instances

Une ressource InstanceProfile est ajoutée :

Roles: ["LabRole"]
InstanceProfileName: "LabInstanceProfile"

Cela attache automatiquement le rôle IAM LabRole à toutes les instances EC2 créées, comme demandé.

c) Déploiement des 4 instances EC2

Le script crée les instances suivantes :

Instance	Sous-réseau	Type	IAM Role

public-instance-az1	PublicSubnetAz1	t3.micro	LabRole
public-instance-az2	PublicSubnetAz2	t3.micro	LabRole
private-instance-az1	PrivateSubnetAz1	t3.micro	LabRole
private-instance-az2	PrivateSubnetAz2	t3.micro	LabRole

Chaque instance utilise la KeyPair fournie dans le script : **polystudent-pairs**

Le template est exporté dans un fichier local : **ec2_alarms_template.yaml**

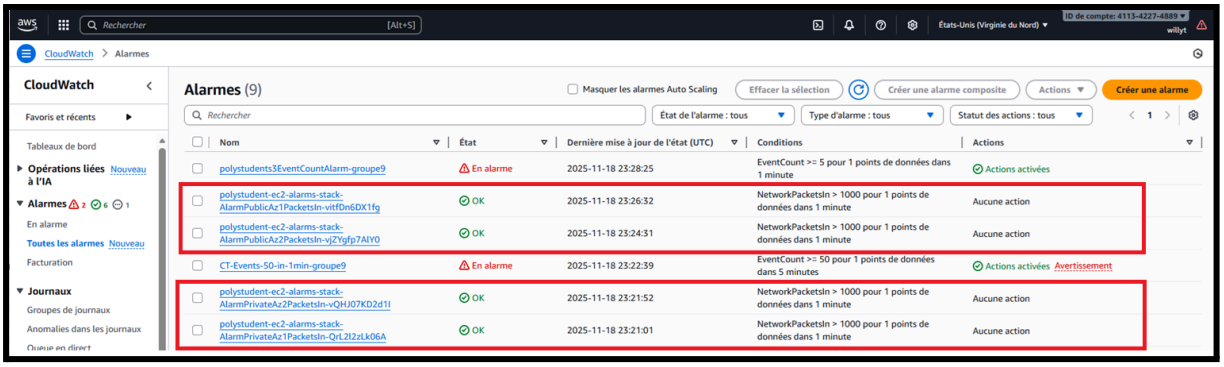
```
(.venv) PS C:\Users\Willy T\Documents\INF8102\TP4> python .\deploiement_ec2_alarms_iac.py
Template EC2 + alarmes générée dans ec2_alarms_template.yaml
Création du stack CloudFormation polystudent-ec2-alarms-stack...
Stack en cours de création, id : arn:aws:cloudformation:us-east-1:411342274889:stack/polystudent-ec2-alarms-stack/2d868b10-c4cc-11
f0-bce3-12d4856459a1
Va dans la console AWS → CloudFormation pour vérifier l'état du stack, puis dans EC2 (instances + rôle IAM LabRole) et dans CloudW
atch (alarmes).
(.venv) PS C:\Users\Willy T\Documents\INF8102\TP4>
```

4. Ajout des alarmes CloudWatch

Une alarme CloudWatch est déployée pour chaque instance EC2 sur la métrique réseau :

- **Namespace** : AWS/EC2
- **MetricName** : NetworkPacketsIn
- **Statistic** : Average
- **Threshold** : 1000 pkts/sec
- **EvaluationPeriods** : 1 (période de 60 sec)
- **ComparisonOperator** : GreaterThanThreshold

Ce mécanisme permet d'alerter en cas de trafic entrant anormalement élevé.



Déploiement et preuve

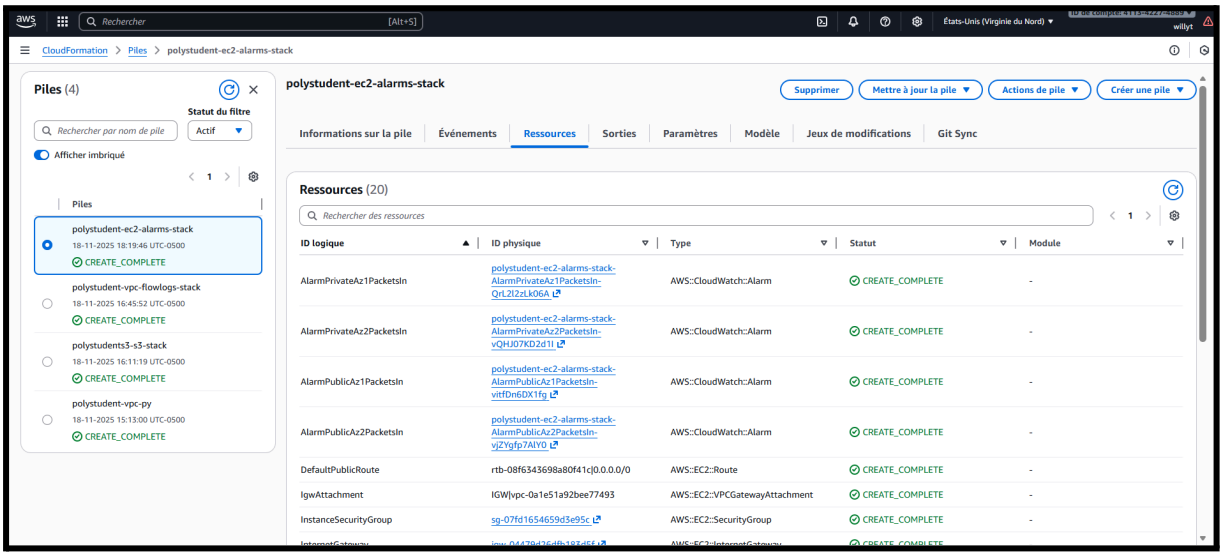
Le script est exécuté avec : python deploiement_ec2_alarms_iac.py

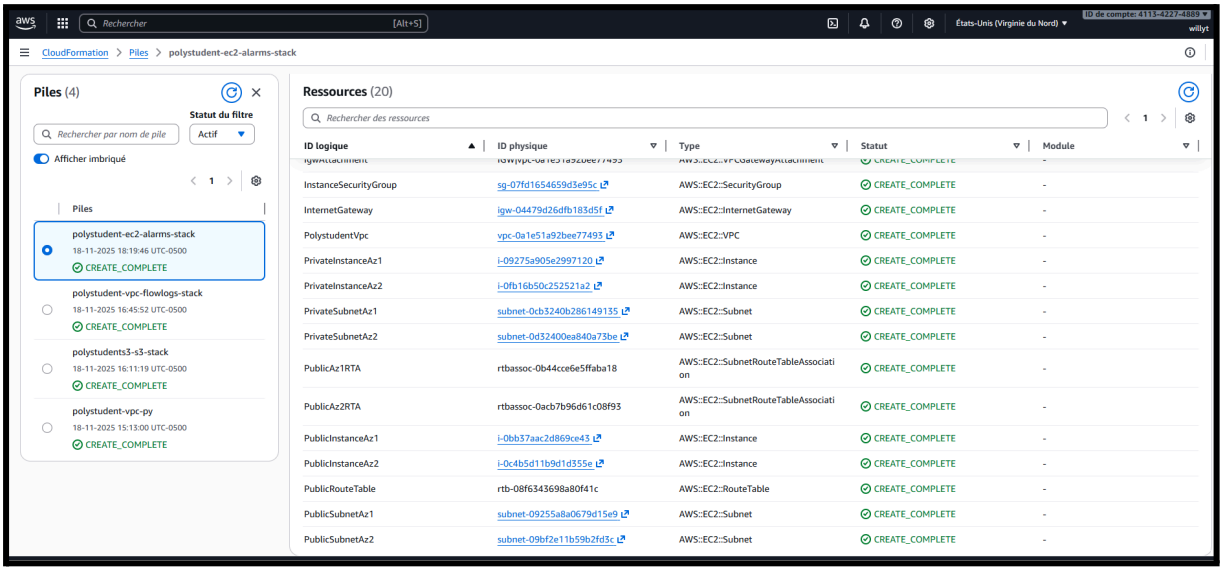
Il déclenche la création du stack : **polystudent-ec2-alarms-stack**

Les validations suivantes ont été effectuées :

✔ 1. CloudFormation

- Stack présent en **CREATE_COMPLETE**
- 4 ressources EC2 + 4 alarmes CloudWatch + InstanceProfile

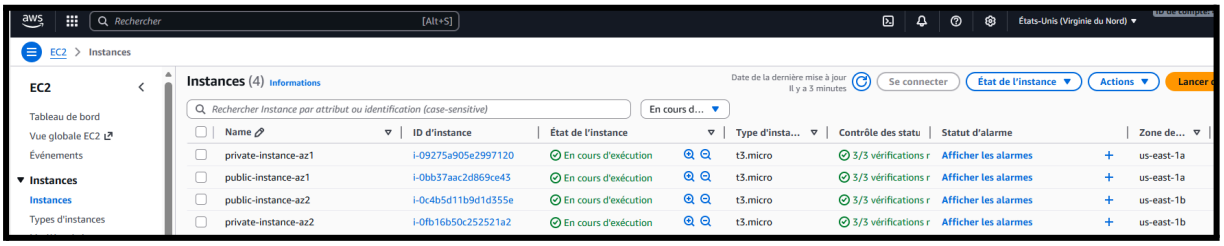


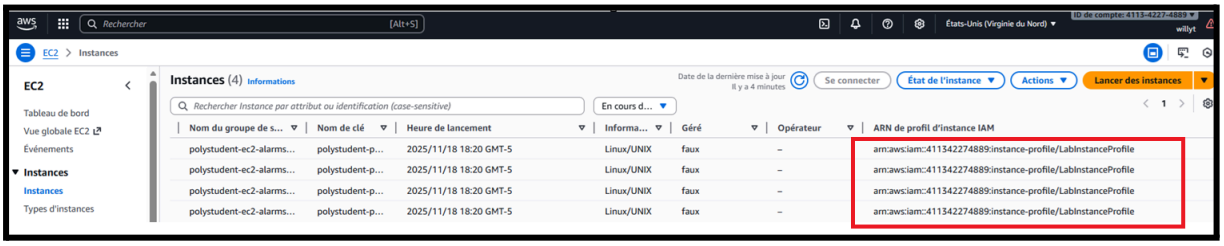


✔ 2. Instances EC2

Dans la console EC2, les 4 instances apparaissent :

- 2 en subnets publics
- 2 en subnets privés
- IAM Role = LabRole visible dans l'onglet *IAM Role* de chaque instance



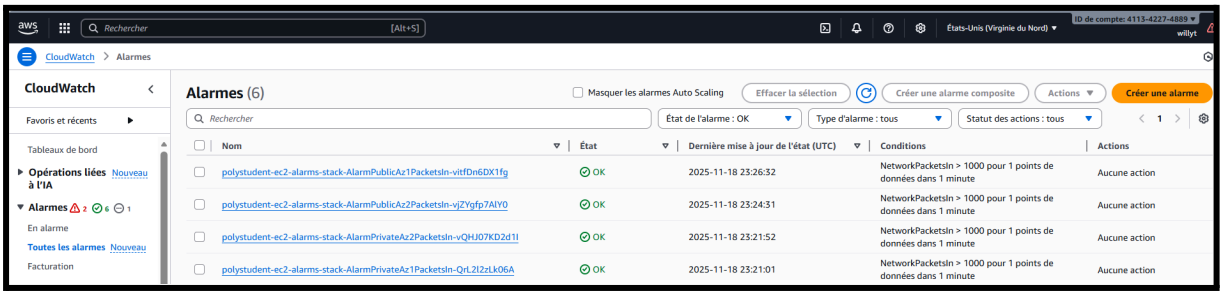


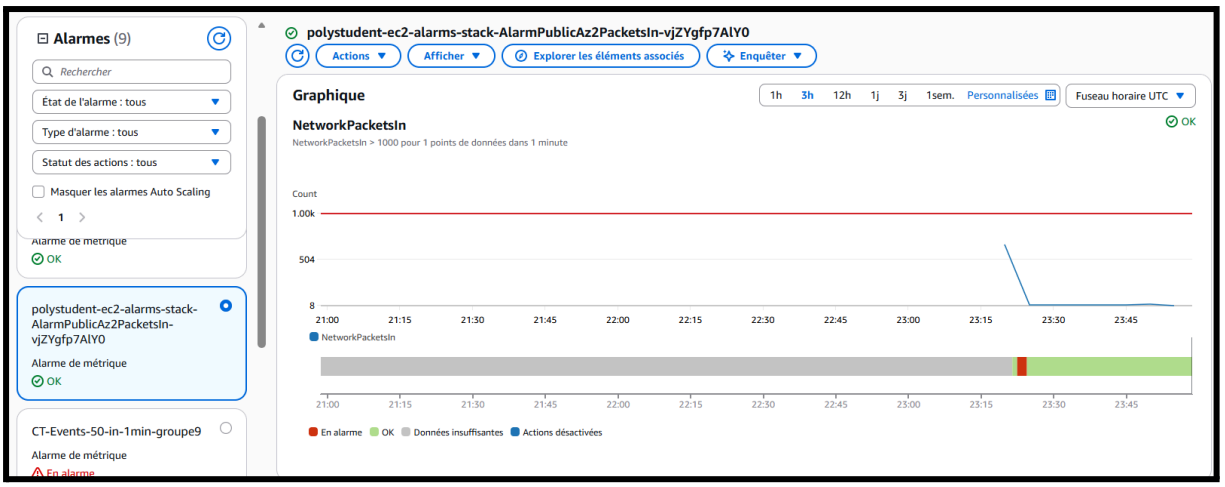
✔ 3. Alarmes CloudWatch

Les 4 alarmes ont été correctement créées :

- AlarmPublicAz1PacketsIn
- AlarmPublicAz2PacketsIn
- AlarmPrivateAz1PacketsIn
- AlarmPrivateAz2PacketsIn

Toutes configurées à **1000 pkts/sec**.





Cette question m'a permis d'automatiser une architecture multi-AZ complète comprenant :

- 4 instances EC2 distribuées en public/privé,
- un rôle IAM attaché à toutes les instances (LabRole),
- un mécanisme avancé de supervision CloudWatch basé sur la surveillance réseau.

L'intégration du tout dans un script IaC Python offre une solution reproductible, sécurisée et conforme aux pratiques infonuagiques vues en cours.

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.

Cette question vise à compléter la configuration du bucket S3 polystudents3-group9 en y ajoutant deux mécanismes essentiels de résilience et de traçabilité :

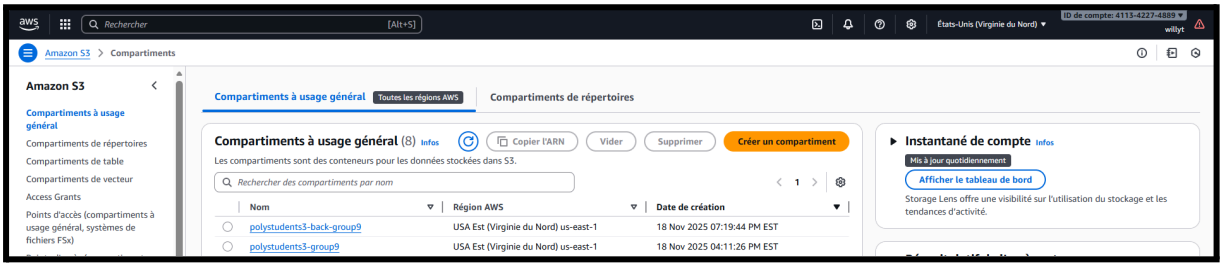
1. Activer la réplication inter-bucket du contenu vers un bucket de secours polystudents3-back-group9.

2. Activer CloudTrail pour journaliser toutes les opérations de modification et de suppression d'objets (PutObject, DeleteObject) sur le bucket source.

L'ensemble doit être réalisé via Python (boto3), testé et accompagné de preuves.

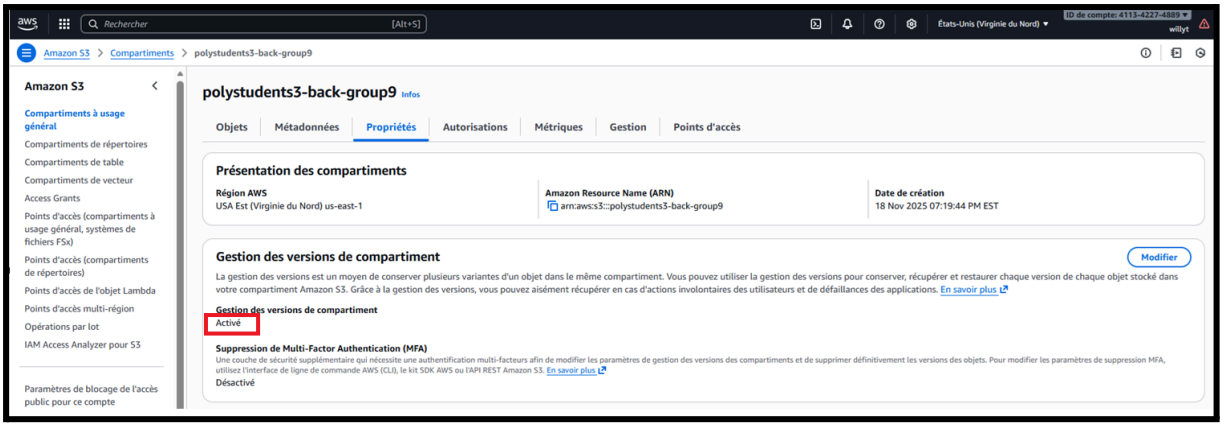
Un script Python nommé **deploiement_s3_replication_cloudtrail.py** a été développé afin d'automatiser l'intégralité de la configuration. Ce script exécute successivement les opérations suivantes :

```
(.venv) PS C:\Users\Willy T\Documents\INF8102\TP4> python .\deploiement_s3_replication_cloudtrail.py
Rôle polystudents3-replication-role-group9 existe déjà, récupération de l'ARN ...
Attachement de la policy de réplication au rôle polystudents3-replication-role-group9 ...
Configuration de la réplication sur le bucket source polystudents3-group9 vers polystudents3-back-group9 ...
Réplication configurée.
Application de la bucket policy CloudTrail sur polystudents3-back-group9 ...
Bucket policy CloudTrail appliquée.
Création du trail CloudTrail polystudents3-trail-group9 ...
Configuration des data events CloudTrail pour journaliser les écritures (Put/Delete) sur les objets de polystudents3-group9 ...
Démarrage du logging CloudTrail ...
CloudTrail est maintenant actif pour le bucket source.
=== Déploiement terminé. Vérifie S3 (Replication), le bucket de backup et CloudTrail (Trails & Event history). ===
(.venv) PS C:\Users\Willy T\Documents\INF8102\TP4>
```



a) Vérification et configuration des buckets S3

- Vérification de l'existence du bucket source polystudents3-group9.
- Création du bucket de destination polystudents3-back-group9 si nécessaire.
- Activation du versioning sur les deux buckets, condition indispensable à la réplication S3.



b) Création du rôle IAM de réplication

Un rôle IAM nommé polystudents3-replication-role-group9 est créé pour permettre à S3 :

- de lire les versions d'objets dans le bucket source ;
- de répliquer les objets dans le bucket de destination.

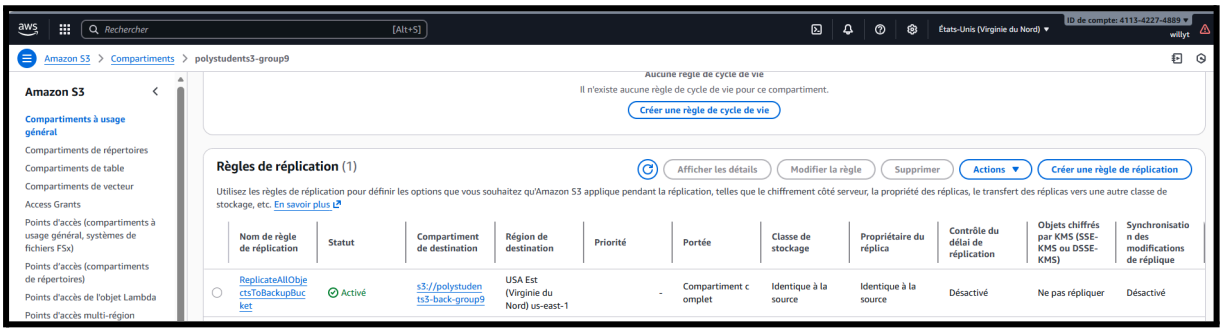
Une politique inline est attachée au rôle afin d'autoriser toutes les actions requises pour la réplication (s3:GetObjectVersion, s3:ReplicateObject, s3:ReplicateDelete, etc.).

c) Activation de la réplication S3

Une règle de réplication complète est appliquée sur le bucket polystudents3-group9 :

- ID de la règle : **ReplicateAllObjectsToBackupBucket**
- Statut : **Enabled**
- Préfixe : vide (tous les objets)
- Destination : arn:aws:s3:::polystudents3-back-group9

À partir de ce moment, toute création, modification ou suppression d'objet dans le bucket source est automatiquement répliquée vers le bucket de destination.



d) Application d'une bucket policy CloudTrail

CloudTrail exige une bucket policy spécifique pour pouvoir écrire des fichiers de logs dans un bucket S3.
Le script applique automatiquement cette policy au bucket polystudents3-back-group9, incluant :

- autorisation s3:GetBucketAcl pour CloudTrail ;
- autorisation s3:PutObject sous le préfixe AWSLogs/411342274889/* ;
- contrainte s3:x-amz-acl = bucket-owner-full-control.

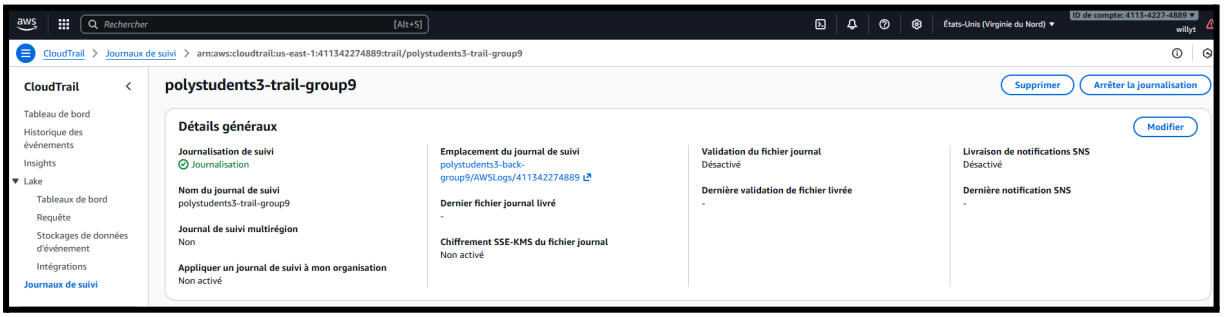
e) Activation de CloudTrail et configuration des Data Events

Un trail CloudTrail nommé **polystudents3-trail-group9** est créé et configuré pour enregistrer les **opérations d'écriture (WriteOnly)** sur les objets du bucket polystudents3-group9.

Configuration des Data Events :

- Type = AWS::S3::Object
- Values = arn:aws:s3:::polystudents3-group9/
- ReadWriteType = WriteOnly

Le logging CloudTrail est ensuite démarré.

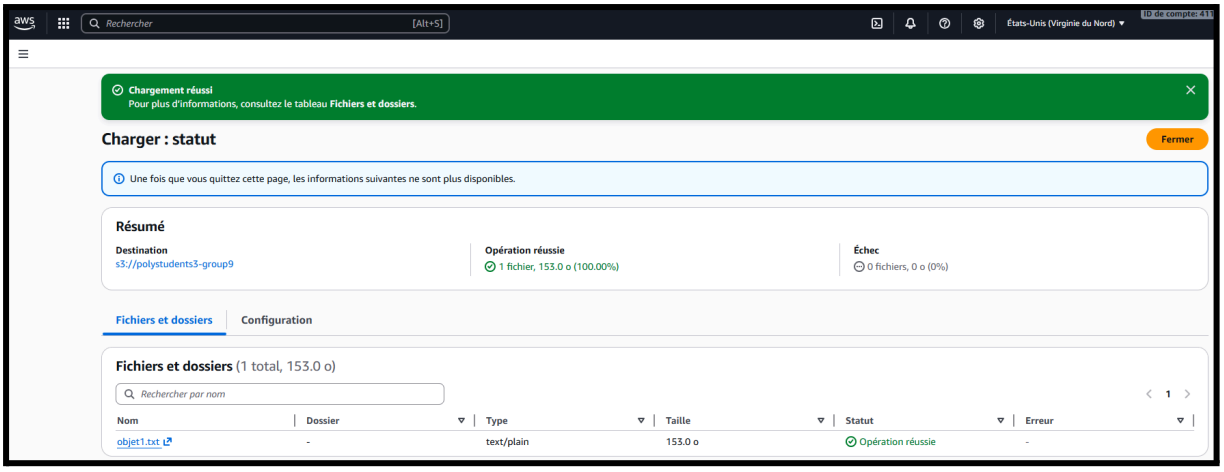


Preuves de bon fonctionnement

Les validations suivantes ont été effectuées :

✔ Réplication S3 opérationnelle

Après upload d'un objet dans le bucket source, celui-ci apparaît automatiquement dans : polystudents3-back-group9

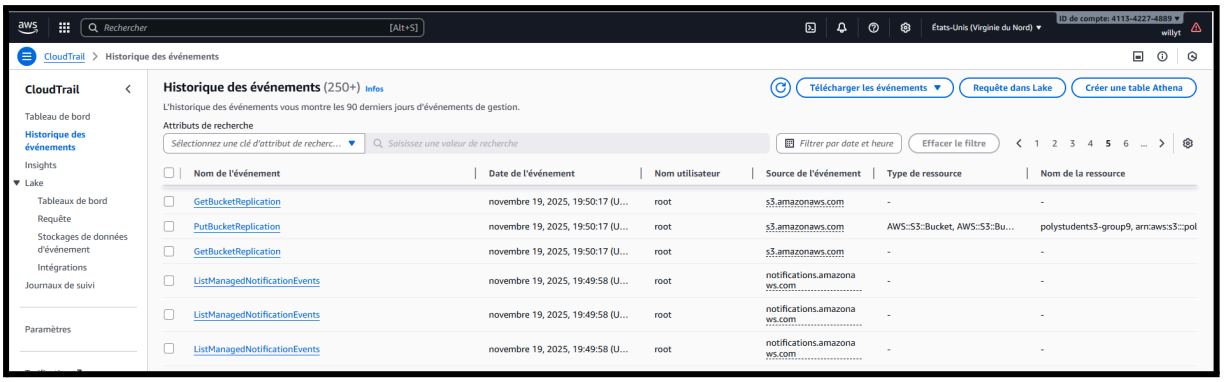


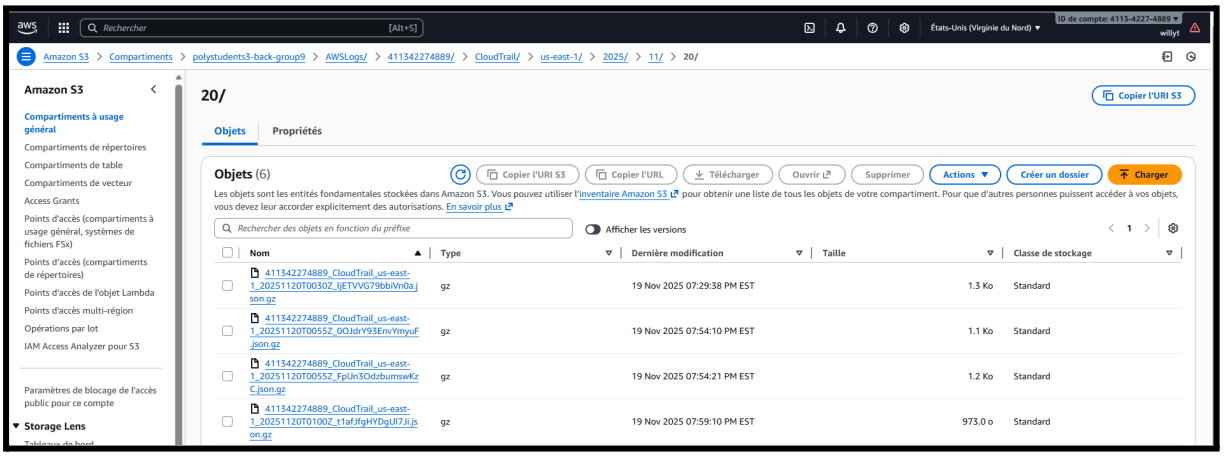
✔ CloudTrail journalise les opérations S3

Dans **CloudTrail -> Event history**, un filtre sur :

- Resource Type : AWS::S3::Object
- Event Name : PutObject / DeleteObject
- Resource : polystudents3-group9

affiche correctement les événements générés lors de la modification des objets.





Cette question m'a permis de mettre en œuvre deux contrôles essentiels à la sécurité dans AWS :

- **Réplication automatique S3**, améliorant la résilience et la continuité des opérations.
- **CloudTrail Data Events**, assurant une traçabilité complète des opérations sur les objets sensibles.

Le script Python fournit une solution automatisée, reproductible et conforme aux bonnes pratiques répondant ainsi aux exigences du TP4.

4. Scan with Trivy

4.1 Generate scan report of vulnerabilities with medium, high, and critical severity (5 pts)

Dans cette section, l'objectif est d'effectuer un scan de sécurité sur l'ensemble du code IaC généré dans les questions précédentes (VPC, EC2, S3, Flow Logs) afin d'identifier les vulnérabilités potentielles présentes dans les templates CloudFormation.
Pour ce faire, nous avons utilisé l'outil Trivy, comme recommandé dans le TP3, en mode filesystem scan (fs).
Les fichiers IaC à analyser ont été préalablement regroupés dans un dossier dédié nommé polylab/, comprenant les templates suivants :

- vpc_template.yaml
- s3_template.yaml
- vpc_flowlogs_template.yaml
- ec2_alarms_template.yaml

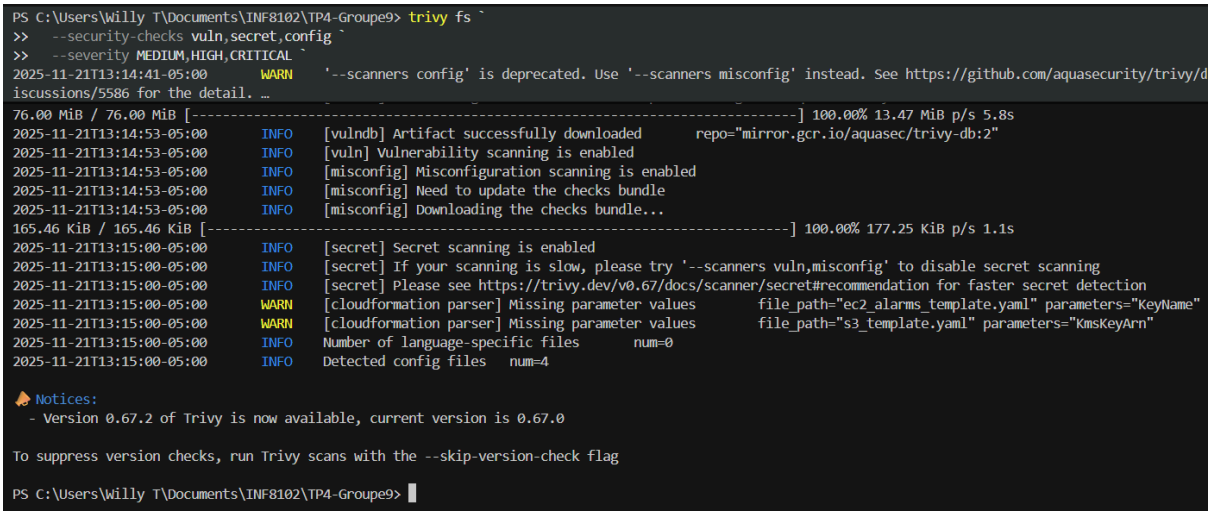
L'analyse est lancée depuis la racine du projet TP4 via la commande suivante :

```
trivy fs --security-checks vuln,secret,config `
    --severity MEDIUM,HIGH,CRITICAL `
    -f json `
    -o scans/trivy_report.json `
    .\polylab\
```

Cette commande permet :

- d'effectuer un scan complet des vulnérabilités, mauvaises configurations et secrets,
- de filtrer uniquement les sévérités MEDIUM, HIGH et CRITICAL, comme demandé,
- d'exporter le rapport final au format JSON dans le fichier scans/trivy_report.json.

L'exécution de Trivy (reproduite dans la capture d'écran ci dessous) confirme que : le scan s'est déroulé avec succès, aucun secret n'a été détecté, plusieurs vulnérabilités de configuration (« Misconfigurations ») ont été identifiées, principalement liées aux bonnes pratiques AWS (security groups, configuration EC2, protections S3, etc.).
Ce rapport constitue la preuve exigée et sera utilisé dans la section suivante pour extraire les vulnérabilités de sévérité HIGH.



4.2 Extract Description, CVSSv3, and high severity using the jq command and stores it in the file cve.json (5 pts)

Pour cette étape, l'objectif était d'extraire depuis le rapport Trivy uniquement les vulnérabilités de sévérité HIGH, comme demandé dans l'énoncé.
Pour faciliter le filtrage et rendre la commande plus lisible, j'ai placé le script jq dans un fichier séparé nommé filter.jq.
Le contenu du fichier filter.jq est le suivant :

```
.Results[]?
| .Vulnerabilities? // []
| map(select(.Severity == "HIGH"))
| map({
    ID: .VulnerabilityID,
    Title: .Title,
    Description: .Description,
    Severity: .Severity,
    CVSSv3: (.CVSS? | to_entries[]?.value?.V3Vector // null)
})
```

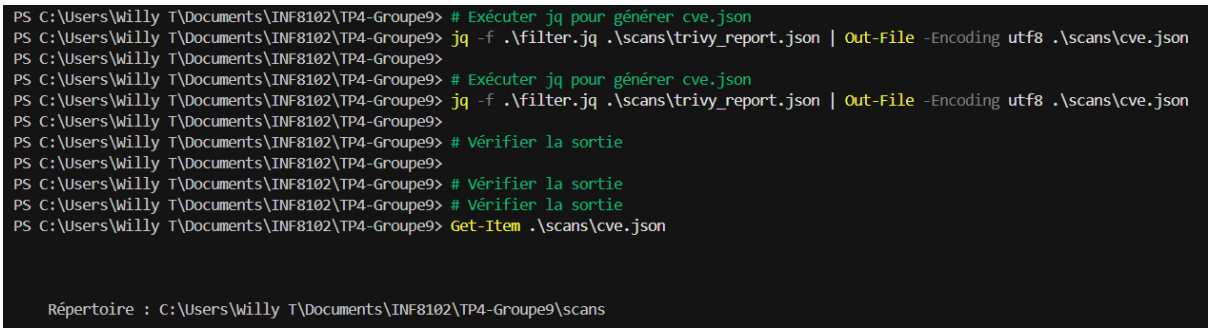
L'extraction a ensuite été exécutée via PowerShell à l'aide de la commande suivante :

```
jq -f .\filter.jq .\scans\trivy_report.json | Out-File -Encoding utf8 .\scans\cve.json
```

Cette commande permet :

- de charger le script de filtrage filter.jq,
- d'appliquer ce filtre au rapport généré par Trivy,
- d'extraire uniquement les vulnérabilités de sévérité *HIGH*,
- de produire un fichier de sortie nommé cve.json dans le dossier scans/.

Une fois l'exécution terminée, j'ai vérifié la présence du fichier avec :
Get-Item .\scans\cve.json
La capture d'écran ci-dessous (voir page XX) présente la sortie complète du terminal lors de l'exécution de jq, constituant la preuve exigée par le TP.
À noter : dans mon cas, Trivy n'a pas identifié de vulnérabilités CVE de type HIGH dans le code IaC, ce qui est normal pour des templates CloudFormation.
Les seules alertes disponibles concernent principalement des misconfigurations, qui ne sont pas inclues dans cve.json car ce fichier est réservé aux CVE logicielles.



4.3 Describe 5 security measures to mitigate vulnerabilities in the IaC code (5 pts)

Le fichier cve.json est vide car le scan Trivy n'a détecté aucune vulnérabilité CVE dans les composants logiciels.
En revanche, le rapport trivy_report.json contient plusieurs vulnérabilités de configuration (Misconfigurations) de sévérité HIGH et CRITICAL, issues exclusivement des templates CloudFormation analysés.
Les cinq mesures de mitigation proposées se basent directement sur ces misconfigurations IaC.

À la suite du scan Trivy réalisé sur les différents templates CloudFormation du projet (VPC, EC2, S3, Flow Logs), plusieurs faiblesses critiques et majeures ont été détectées. Afin de renforcer la posture de sécurité de l'infrastructure déployée via IaC, les cinq mesures d'atténuation suivantes sont proposées :

1. Activer IMDSv2 sur toutes les instances EC2

Le rapport Trivy met en évidence plusieurs ressources EC2 où l'Instance Metadata Service (IMDS) n'exige pas l'utilisation de jetons (AVD-AWS-0028 – sévérité HIGH). Pour se prémunir des attaques SSRF visant le vol des credentials IAM, il est recommandé d'imposer IMDSv2 via le bloc suivant :

```
MetadataOptions:
        HttpTokens: required
        HttpEndpoint: enabled
```

Cette configuration garantit que seules les requêtes authentifiées peuvent accéder aux métadonnées sensibles de l'instance.

2. Restreindre les règles de sécurité des Security Groups

Le scan a révélé plusieurs règles d'ingress et d'egress exposées publiquement avec CidrIp: 0.0.0.0/0, notamment sur les ports SSH/RDP et sur les flux sortants (AVD-AWS-0104 — CRITICAL, AVD-AWS-0107 — HIGH).

Pour réduire l'exposition aux attaques réseau :

- limiter les flux à un CIDR interne (ex.: 10.0.0.0/16)
- ou à une IP spécifique (bastion host)
- ou à un autre Security Group.

Exemple recommandé : CidrIp: 10.0.0.0/16

Cela applique le principe du moindre privilège et élimine les accès indiscriminés depuis Internet.

3. Chiffrer les volumes EBS et les disques racine des instances EC2

Plusieurs ressources EC2 présentent des volumes non chiffrés (AVD-AWS-0131 – HIGH).

Afin d'assurer la protection des données au repos et de respecter les bonnes pratiques AWS ainsi que les exigences légales (ex. : Loi 25), les volumes doivent être chiffrés :

```
BlockDeviceMappings:
  - DeviceName: /dev/xvda
    Ebs:
      Encrypted: true
      KmsKeyId: !Ref MyKmsKeyArn
```

Ce contrôle réduit le risque de fuite d'information en cas de compromission de disque.

4. Activer les mécanismes S3 Block Public Access

Le template S3 analysé ne comporte pas de configuration empêchant les accès publics accidentels (AVD-AWS-0086/0087/0091/0093 — HIGH).

Il est impératif d'activer les quatre protections natives AWS :

```
PublicAccessBlockConfiguration:
  BlockPublicAcls: true
```

BlockPublicPolicy: true
IgnorePublicAcls: true
RestrictPublicBuckets: true

Cette configuration garantit qu'aucune ACL ou politique publique ne puisse exposer involontairement le bucket et les données sensibles qu'il contient.

5. Désactiver l'attribution automatique d'adresses IP publiques dans les subnets

Plusieurs subnets utilisent MapPublicIpOnLaunch: true, ce qui expose les instances à une adresse publique dès leur création (AVD-AWS-0164 – HIGH). Afin de réduire la surface d'attaque et de forcer le passage par des points d'entrée contrôlés (ex. : ALB, NAT Gateway, Bastion), il est recommandé de désactiver cette option : **MapPublicIpOnLaunch: false**

Cette mesure empêche la création d'instances involontairement accessibles sur Internet.

Ces cinq mesures permettent de corriger les principales vulnérabilités identifiées par le scan IaC, tout en alignant l'infrastructure sur les meilleures pratiques AWS. Leur mise en œuvre améliore significativement la sécurité globale de l'environnement cloud déployé via CloudFormation.

NB : lien Github : https://github.com/willytalabong-commits/INF8102-TP4-G9