



DELANNOY  
ROMAIN

24 FÉVRIER - 4 AVRIL 2025

# RAPPORT DE STAGE 2025

DAMARTEX, DevSecOps Team

damartex  
GROUP

## Table des matières

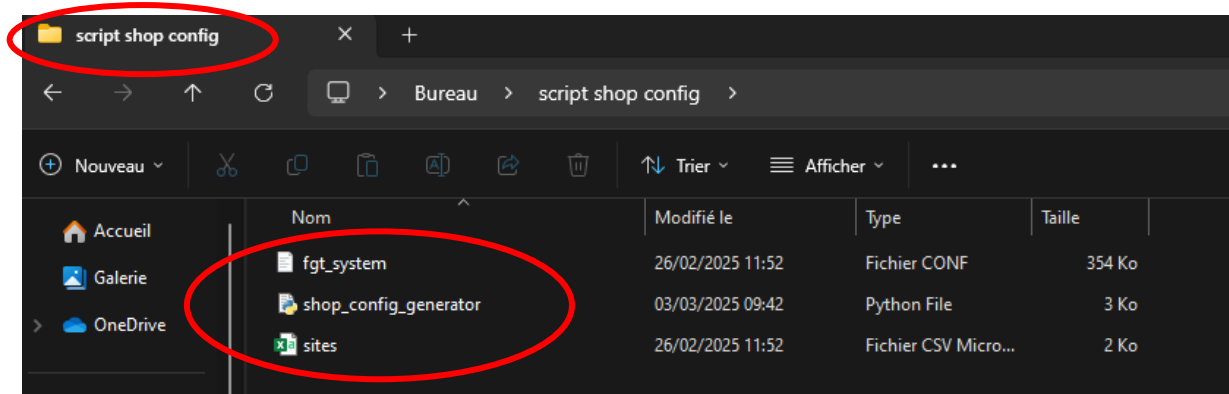
Mission 1 : Industrialiser la configuration de 300 firewalls	3
COMPETENCES .....	9
Mission 2 : Script bruno en js avec call API .....	11
COMPETENCES .....	16
Mission 3 : Migration de listener vers AWS .....	17
COMPETENCES .....	17
Mission 4 : Intégrer et gérer des tâches/services ECS (Elastic Container Service) d'AWS.....	20
COMPETENCES .....	22

# Mission 1 : Industrialiser la configuration de 300 firewalls

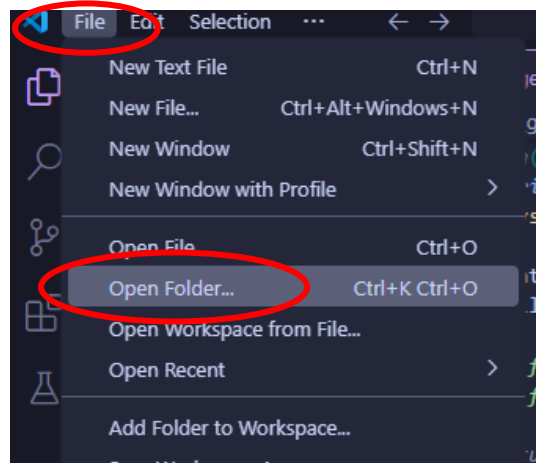
Le but était de créer un script en PYTHON qui génère les fichiers de configuration Forti de 173 magasins

## Utilisation du script shop config

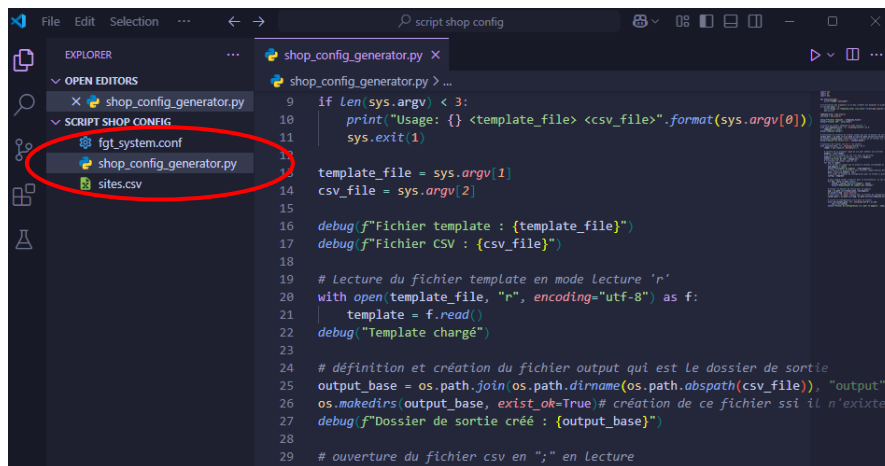
Pour faire fonctionner le script, il faut avoir le fichier **template** (par exemple : fgt\_system) et le fichier **CSV** (par exemple : sites) au bon endroit dans le dossier "**script shop config**", où se trouve également le fichier **shop\_config\_generator.py**, qui est le script.



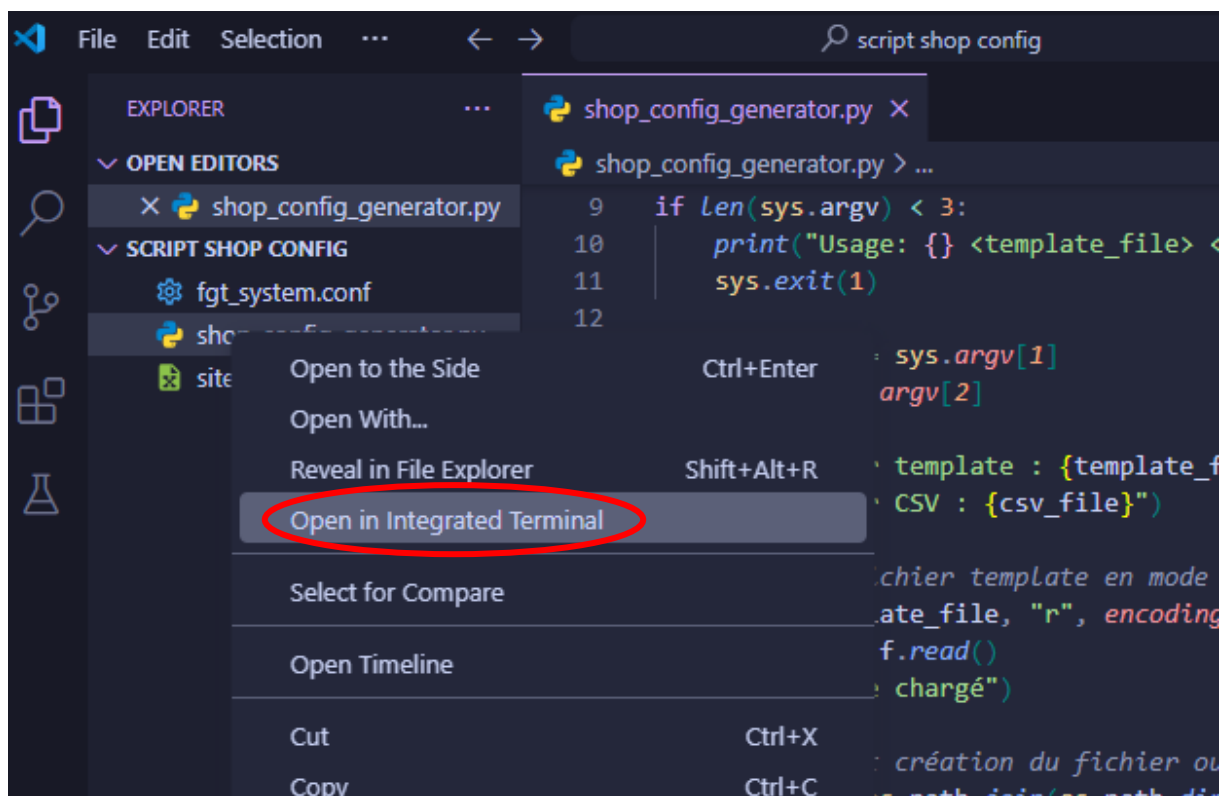
Une fois les fichiers placés correctement, il faut ouvrir le dossier "script shop config" dans VS Code en utilisant "Open Folder".



Une fois le dossier ouvert, tu verras tous les fichiers du dossier sur la gauche.



Ensuite, fais un clic droit sur le nom du script shop\_config\_generator.py et sélectionne "Open in Integrated Terminal" pour exécuter le script



Un terminal s'ouvrira en bas du script. Il faudra alors taper la commande suivante dans le terminal :

```
python shop_config_generator.py <template_file> <csv_file>
```

Remplace <template\_file> et <csv\_file> par le chemin et le nom des fichiers respectifs

C'est un exemple pour les fichiers de MAGFR

<template\_file> = .\FR\MAGFR\fgt\_system.conf et <csv\_file> = .\FR\MAGFR\MAGFR.csv

La commande finale sera donc :

```
python shop_config_generator.py .\FR\MAGFR\fgt_system.conf .\FR\MAGFR\MAGFR.csv
```

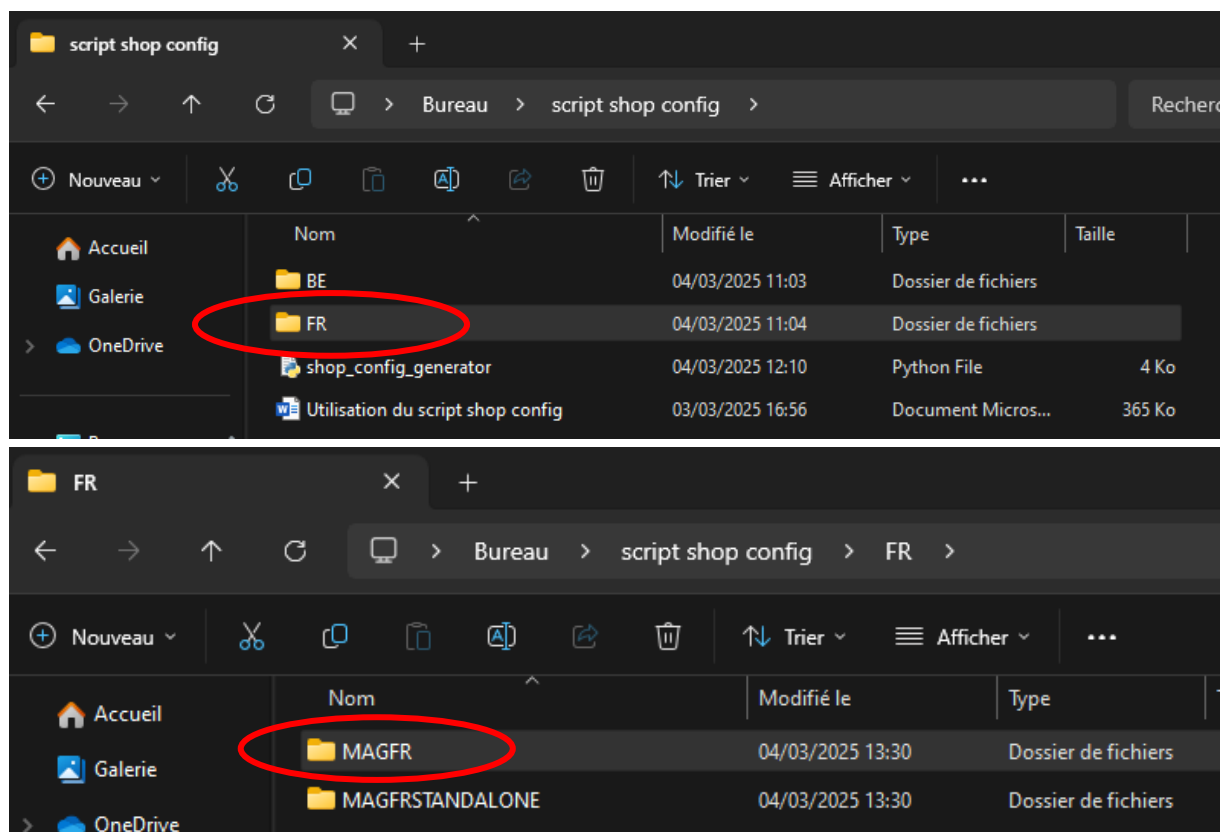
Le script sera exécuté si, dans le terminal, la phrase suivante s'affiche :

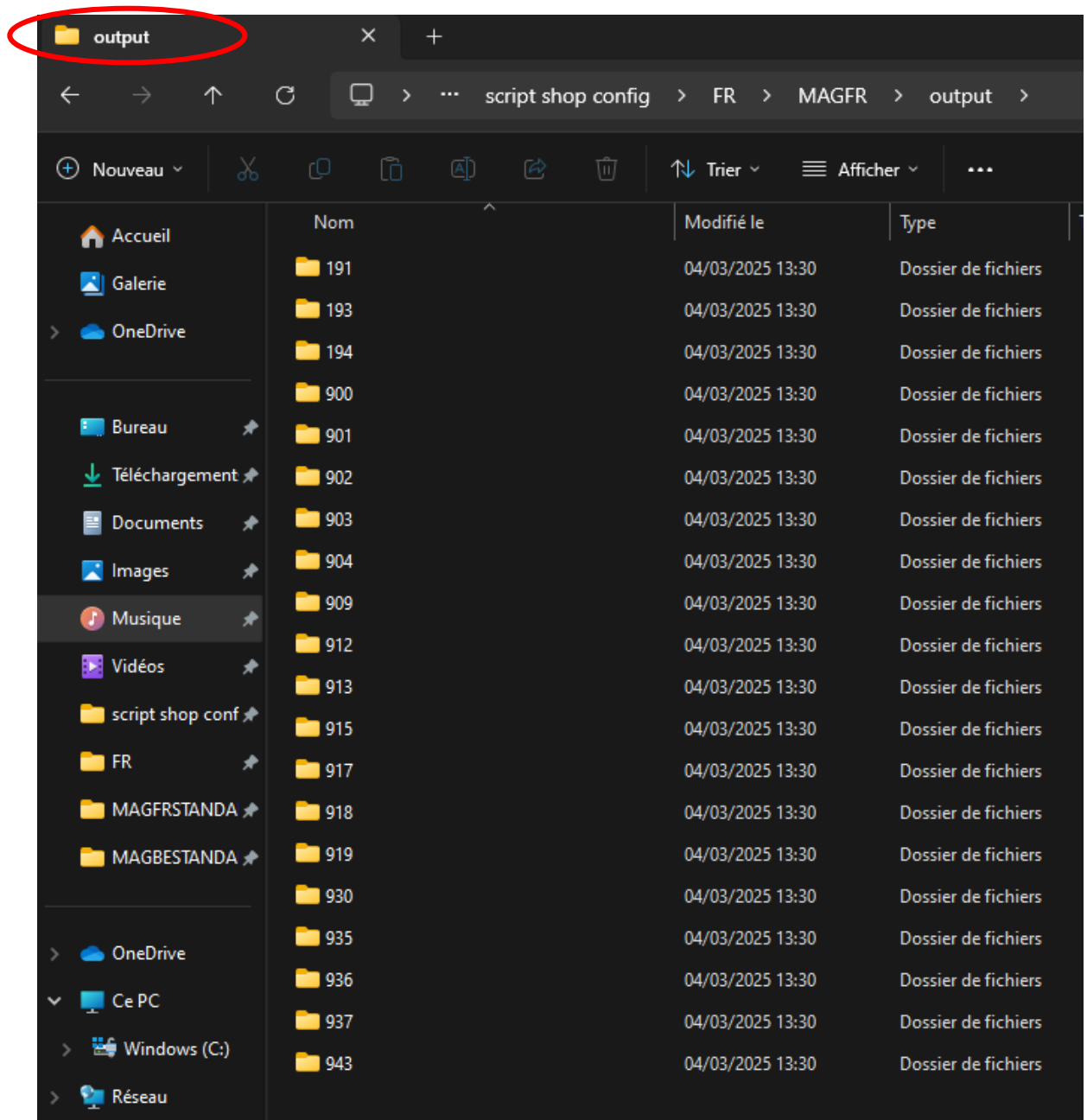
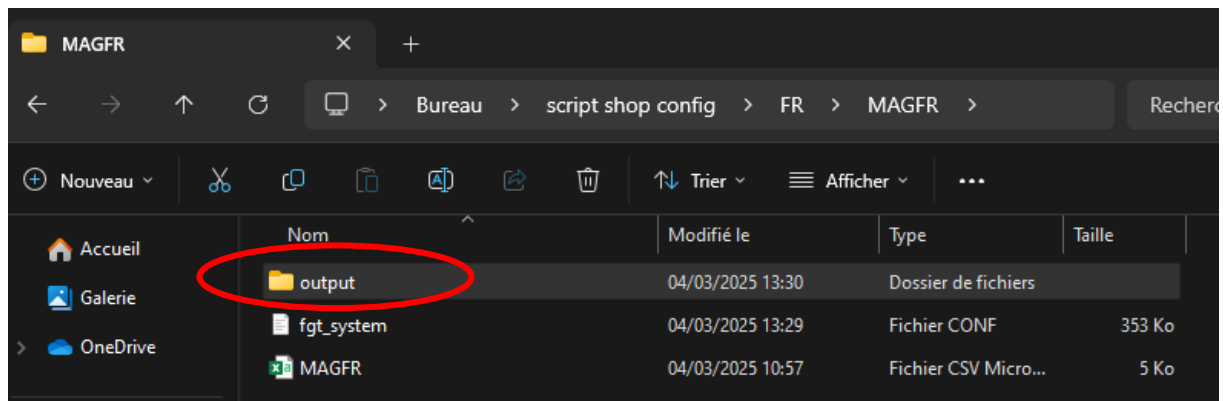
***Le dossier output est créé et le script est terminé.***

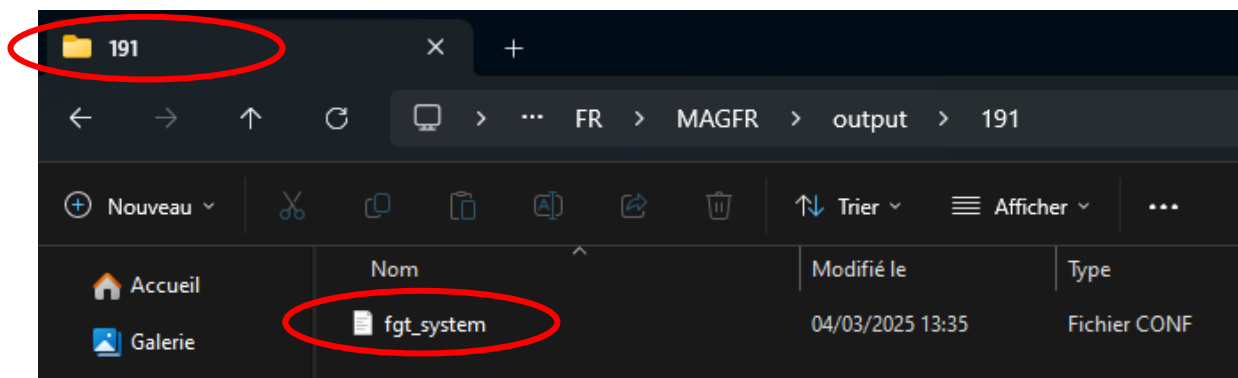
```
PS C:\Users\Administrateur\Desktop\script shop config> python shop_config_generator.py .\FR\MAGFR\fgt_system.conf .\FR\MAGFR\MAGFR.csv
>>
le dossier output est créé et le script est terminé
PS C:\Users\Administrateur\Desktop\script shop config> 
```

Le dossier output apparaîtra donc dans le dossier

.\script shop config\FR\MAGFR :







*S'il faut le détail de tout ce que fait le script lorsque tu l'exécute*

*Il faut rajouter le mot « --verbose » dans la ligne de commande dans le terminal*

```
python shop_config_generator.py .\FR\MAGFR\fgt_system.conf
.\FR\MAGFR\MAGFR.csv --verbose
```

```
PS C:\Users\Administrateur\Desktop\script shop config> python shop_config_generator.py .\FR\MAGFR\fgt_system.conf .\FR\MAGFR\MAGFR.csv --verbose
>>
DEBUG: Fichier template : .\FR\MAGFR\fgt_system.conf
DEBUG: Fichier CSV : .\FR\MAGFR\MAGFR.csv
DEBUG: Template chargé
DEBUG: Dossier de sortie créé : C:\Users\Administrateur\Desktop\script shop config\FR\MAGFR\output
DEBUG: En-têtes du CSV : ['Code Site', 'ID_MAG', 'N_FW', 'MAG_Y', 'LO_ID', 'HA_PRIORITY', 'PASS', 'HIST_IP', 'HIST_MASK', 'HIST_GW', 'HIST_IP_START', 'HIST_IP_END']
DEBUG: Traitement du magasin : 900
DEBUG: Remplacement de [Code Site] par 900
DEBUG: Remplacement de [ID_MAG] par 900
DEBUG: Remplacement de [N_FW] par 1
DEBUG: Remplacement de [MAG_Y] par 4
DEBUG: Remplacement de [LO_ID] par 102
DEBUG: Remplacement de [HA_PRIORITY] par 250
DEBUG: Remplacement de [PASS] par MAGFR900!
DEBUG: Remplacement de [HIST_IP] par 10.33.214.32
DEBUG: Remplacement de [HIST_MASK] par 255.255.255.224
```

## CODE du Script :

```
import csv
import os
import argparse

parser = argparse.ArgumentParser(description = "Genere des fichiers à partir d'un template et d'un CSV")
parser.add_argument("template", help="fichier template")
parser.add_argument("csv", help="fichier CSV")
parser.add_argument('-v', '--verbose', action='store_true', help='mode debug')

args = parser.parse_args()

def debug(message):
    if args.verbose:
        print(f"DEBUG: {message}")

if not os.path.isfile(args.template):
    print(f"Erreur : Le fichier '{args.template}' n'existe pas.")
    exit(1)

if not os.path.isfile(args.csv):
    print(f"Erreur : Le fichier '{args.csv}' n'existe pas.")
    exit(1)

debug(f"Fichier template : {args.template}")
debug(f"Fichier CSV : {args.csv}")

# Lecture du fichier template en mode lecture 'r'
with open(args.template, "r", encoding="utf-8") as f:
    template = f.read()
debug("Template chargé")

# définition et création du fichier output qui est le dossier de sortie
output_base = os.path.join(os.path.dirname(os.path.abspath(args.csv)), "output")
os.makedirs(output_base, exist_ok=True) # création de ce fichier ssi il n'existe pas
debug(f"Dossier de sortie créé : {output_base}")

# ouverture du fichier csv en ";" en lecture
with open(args.csv, "r", encoding="utf-8") as f:
    reader = csv.reader(f, delimiter=";")

    # lecture de la premiere ligne du csv pour obtenir les en-têtes
    headers = next(reader)
    # on retire les espaces et les "[" pour les en-têtes
    headers = [h.strip().strip("[") for h in headers]
    debug(f"En-têtes du CSV : {headers}")
    # traitement de chaque ligne du csv
    for row in reader:
        # on prends en compte que la première colonne corresponde au nom du magasin
        nom_magasin = row[0]
        debug(f"Traitement du magasin : {nom_magasin}")
        # création d'un dictionnaire pour associer chaque données des lignes du CSV au noms des colonnes
        data = dict(zip(headers, row))
        # on prend en exemple de configuration pour le fichier a générer, le fichier template que l'on donne en argument
        config = template

        # pour chaque valeur récupérer dans le dictionnaire, on les ajoutent aux emplacements correspondant dans la config (le template)
        for key, value in data.items():
            config = config.replace(f"{{{key}}}", value)
            debug(f"Remplacement de {{{key}}} par {value}")

        # Définir le dossier de sortie pour ce magasin
        mag = os.path.join(output_base, nom_magasin)
        os.makedirs(mag, exist_ok=True)
        # Construire le chemin complet pour le fichier de configuration généré en conservant le nom du fichier template d'origine
        output_path = os.path.join(mag, os.path.basename(args.template))

        # Écrire la configuration créée dans le fichier
        with open(output_path, "w", encoding="utf-8") as out:
            out.write(config)
        debug(f"Fichier de configuration créé pour le magasin : {nom_magasin}")

print("le dossier output est créé et le script est terminé")
```



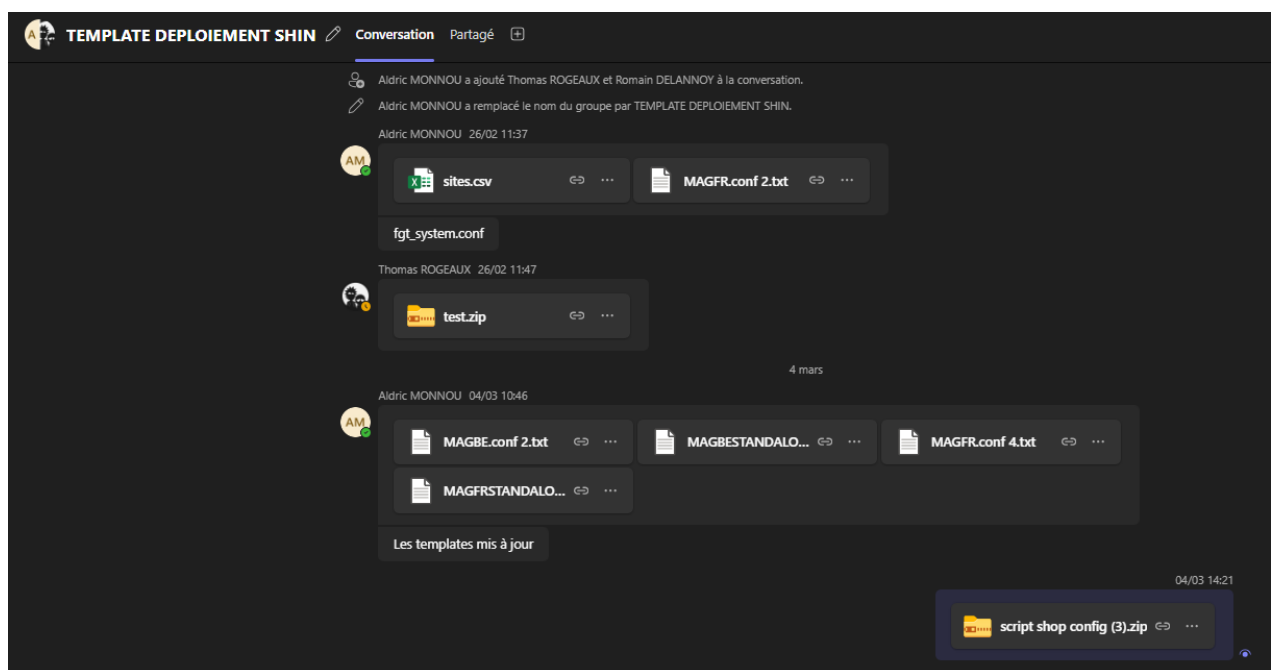
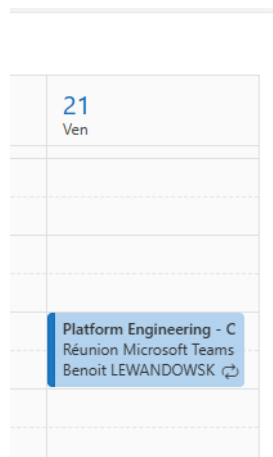
# COMPETENCES

## 1.2 Répondre aux incidents et aux demandes d'assistance et d'évolution

Le script Python permet de traiter en quelques secondes des demandes d'évolution qui, manuellement, auraient pris plusieurs minutes voire heures, améliorant ainsi la qualité et la rapidité du support.

## 1.4 Travailler en mode projet

Ce script a été réalisé pour l'équipe réseau j'ai donc travaillé et collaboré avec l'équipe réseau via teams. De plus, Nous avons réunion tous les vendredis a 10H pour suivre les avancements de projet



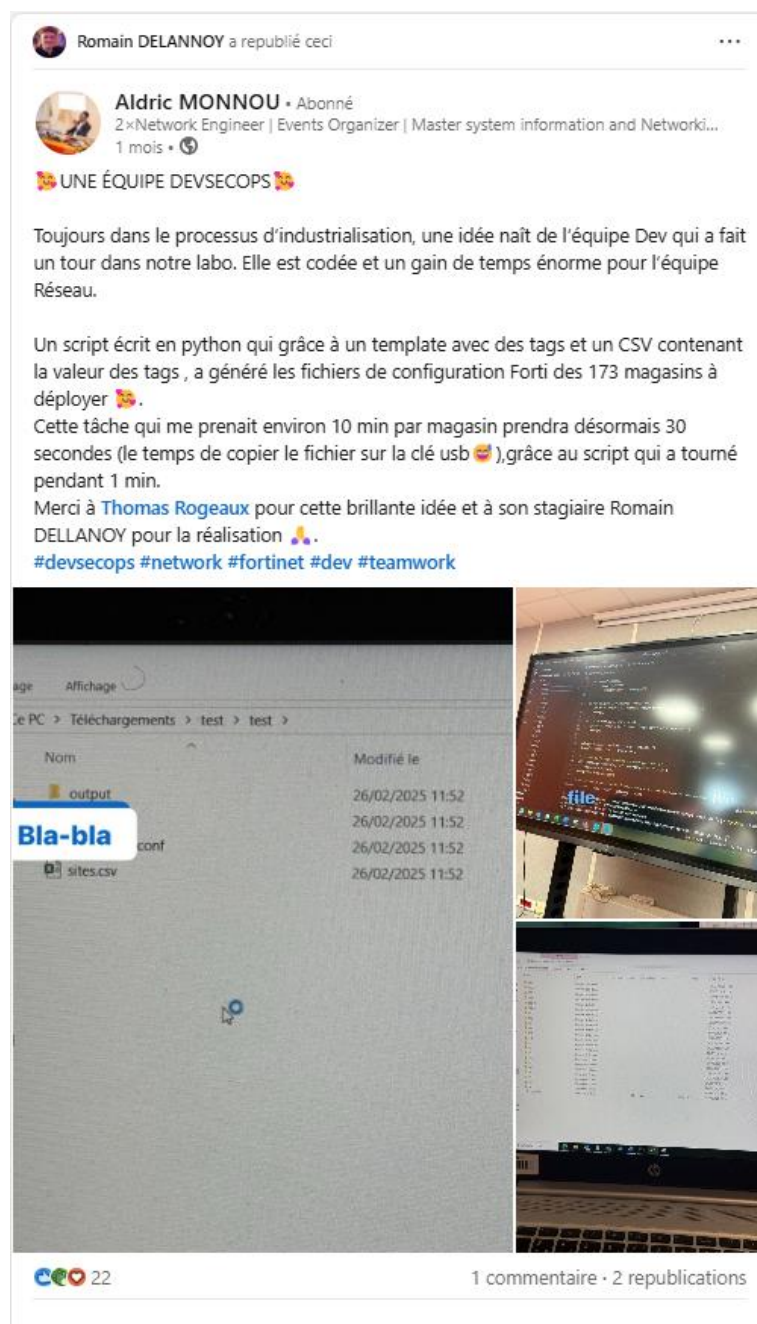
## 1.5 Mettre à disposition des utilisateurs un service informatique

Le script constitue un service opérationnel : génération automatique de la config, livraison intégrée au processus de déploiement des magasins, interface simple (fichier CSV ou interface web légère) pour que les exploitants réseau saisissent leurs paramètres et obtiennent directement le fichier prêt à charger.

## 1.6 Organiser son développement professionnel

Suite à la réalisation du script l'équipe réseau ma remercié via un post LinkedIn

Cela pourrait correspondre à **Gérer son identité professionnelle**



## Mission 2 : Script bruno en js avec call API

Générer et stocker un token pour les API permet de sécuriser les échanges. Le token, est utilisé pour l'authentification, identifie l'utilisateur et autorise l'accès aux API sans mot de passe à chaque requête.

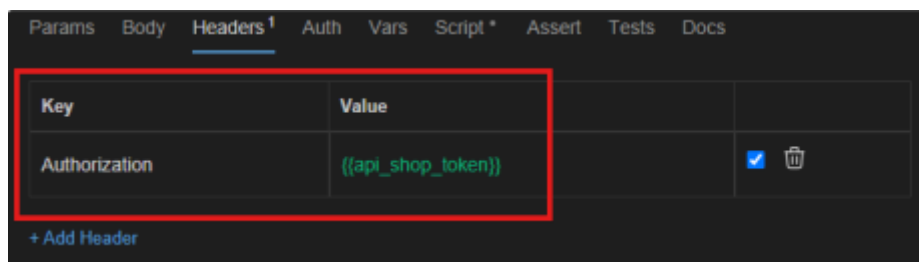
Le fait de le stocker en variable le token va faciliter l'accessibilité des API tout en gardant une sécurisation sur les données.

Cela permet le renouvellement automatique du token pour faciliter l'usage et l'accès aux applications en gardant cela sécurisé.

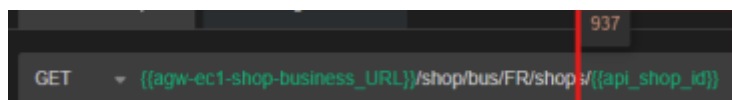
### Mise en œuvre

Création des variables a utiliser pour la génération et le stockage du token

'api\_shop\_token' : cette variable stock le token et l'utilisé dans la value de la key 'Authorization' pour s'identifier

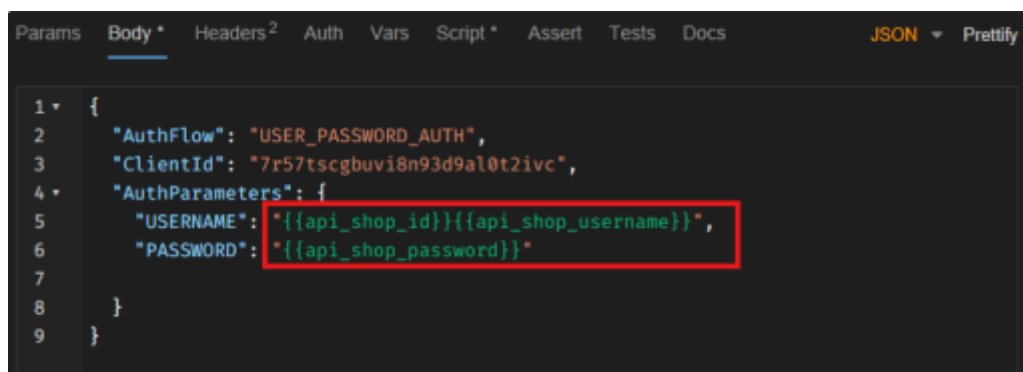


'api\_shop\_id' : cette variable stock l'id du magasin et est réemployé dans les URL et dans le USERNAME de l'authentification

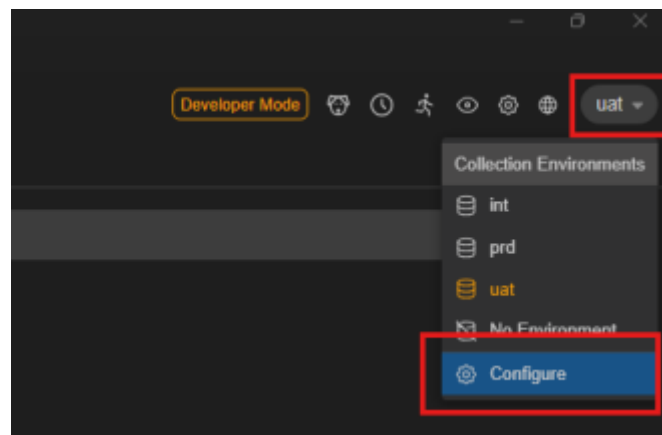


'api\_shop\_username' : cette variable stock l'username et est utiliser associé a l'api\_shop\_id pour y créer l'USERNAME de l'authentification

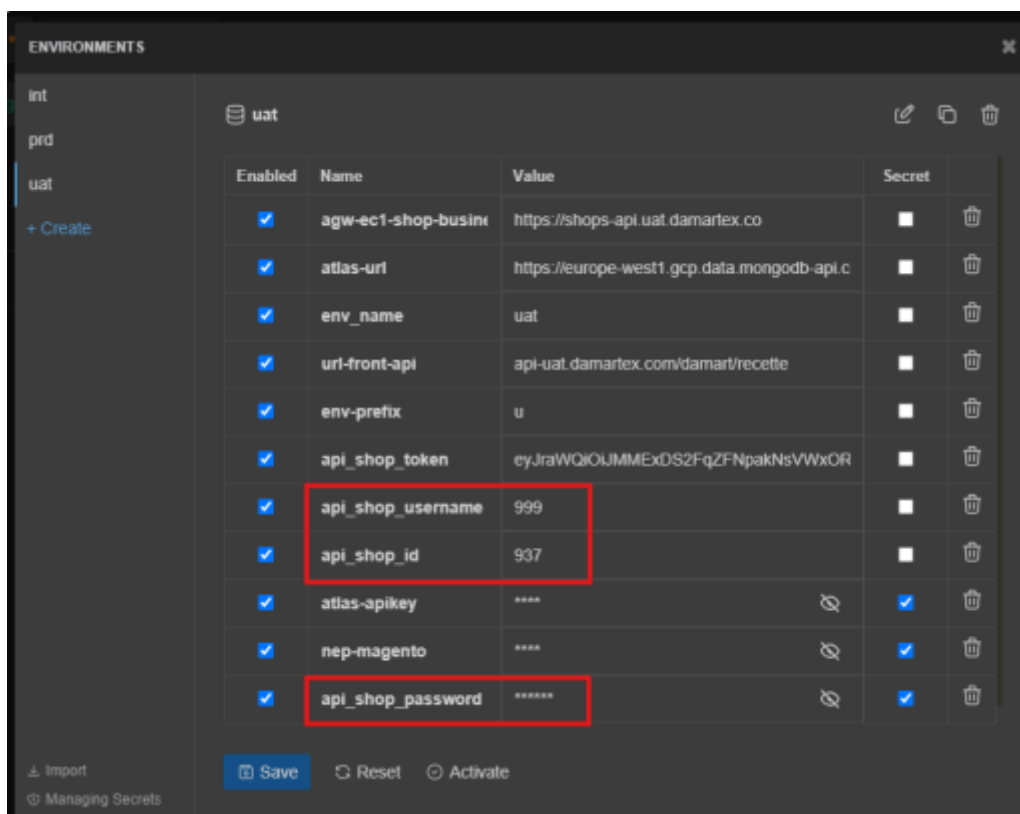
'api\_shop\_password' : cette variable stock le mot de passe de l'utilisateur et est utilisé pour le PASSWORD de l'authentification



Lors de chaque utilisation, il faut commencer par compléter les valeurs des variables citées ci-dessus via *configure*



Paramètres des variables dans le menu déroulant de l'environnement



### Réalisations possibles : Pre Request ou Post Response

Pre Request :

C'est un script à exécuter dans la Pre Request de chaque API

```

Untitled-1

const axios = require('axios');

const USERNAME = bru.getEnvVar('api_shop_username');
const PASSWORD = bru.getEnvVar('api_shop_password');
const clientId = '7r57tscgbuvi8n93d9al0t2ivc';

axios({
  method: 'POST',
  url: 'https://cognito-idp.eu-central-1.amazonaws.com/',
  headers: {
    'Content-Type': 'application/x-amz-json-1.1',
    'X-Amz-Target': 'AWSCognitoIdentityProviderService.InitiateAuth'
  },
  data: {
    AuthFlow: 'USER_PASSWORD_AUTH',
    ClientId: clientId,
    AuthParameters: {
      USERNAME,
      PASSWORD
    }
  }
})

.then(response => {
  const authResult = response.data.AuthenticationResult;
  console.log('Authentification réussie !');

  const token = authResult.AccessToken
  console.log(token)
  bru.setEnvVar('api_shop_token', token);

  console.log(bru.getEnvVar('api_shop_token'));
  console.log('Enregistrement réussie !');
})
.catch(error => {
  console.error('Échec :', error.response?.data || error.message);
});

```

Ce script est donc fonctionnel mais trop long de copier-coller ce script dans le Pre Request de chaque API.

### Alternative, plus simple et rapide a utiliser :

#### Post Response :

Une commande d'une ligne à exécuter dans le Post Response de l'API d'authentification du magasin

```
bru.setEnvVar("api_shop_token", res.body.AuthenticationResult.IdToken);
```

Le code en Pre Request complète une opération d'authentification en utilisant une requête HTTP, récupère un token d'accès ( AccessToken ) et le stocke. Le code Post Response suppose que le token a déjà été obtenu et le stocke simplement dans une

variable d'environnement. La différence est dans l'interaction avec un service d'authentification dans le premier cas et dans le stockage d'un token déjà obtenu dans le second.

## **Fonctionnement**

### **Chargement de la bibliothèque axios**

```
1  const axios = require('axios');
```

axios : C'est une bibliothèque JavaScript populaire utilisée pour effectuer des requêtes HTTP. Dans ce code, elle est utilisée pour envoyer une requête HTTP POST au service afin de s'authentifier.

### **Récupération des variables d'environnement USERNAME et PASSWORD**

```
3  const USERNAME = bru.getEnvVar('api_shop_username');
4  const PASSWORD = bru.getEnvVar('api_shop_password');
```

bru.getEnvVar() : Cette fonction permet de récupérer les valeurs des variables d'environnement stockées sous les noms 'api\_shop\_username' et 'api\_shop\_password'. Ces valeurs représentent respectivement le nom d'utilisateur et le mot de passe utilisés pour l'authentification auprès de l'API.

### **Définition du clientId**

```
5  const clientId = '7r57tscgbuvi8n93d9al0t2ivc';
```

Cela identifie l'application qui tente d'effectuer l'authentification, en lui attribuant des autorisations et un accès approprié.

### **Envoi de la requête HTTP avec axios**

```
7  axios({
8    method: 'POST',
9    url: 'https://cognito-idp.eu-central-1.amazonaws.com/',
10   headers: {
11     'Content-Type': 'application/x-amz-json-1.1',
12     'X-Amz-Target': 'AWSCognitoIdentityProviderService.InitiateAuth'
13   },
14   data: {
15     AuthFlow: 'USER_PASSWORD_AUTH',
16     ClientId: clientId,
17     AuthParameters: {
18       USERNAME,
19       PASSWORD
20     }
21   }
22 })
```

### **Requête HTTP POST :**

Une requête est envoyée à l'URL <https://cognito-idp.eu-central-1.amazonaws.com/> pour démarrer le processus d'authentification.

#### Headers :

'Content-Type': 'application/x-amz-json-1.1' : Indique que les données envoyées sont au format JSON.

'X-Amz-Target': 'AWSCognitoIdentityProviderService.InitiateAuth' : Spécifie l'action d'authentification à effectuer.

#### Data :

AuthFlow: 'USER\_PASSWORD\_AUTH' : Ce paramètre spécifie le flux d'authentification, ici l'authentification avec nom d'utilisateur et mot de passe.

ClientId et AuthParameters : Contient les paramètres nécessaires pour l'authentification, à savoir l'ID client et le USERNAME et le PASSWORD .

#### Traitement de la réponse avec .then()

```
24 * .then(response => {
25     const authResult = response.data.AuthenticationResult;
26     console.log('Authentification réussie !');
27
28     const token = authResult.AccessToken
29     console.log(token)
30     bru.setEnvVar('api_shop_token', token);
31
32     console.log(bru.getEnvVar('api_shop_token'));
33     console.log('Enregistrement réussie !');
34
35 })
```

.then() : La méthode .then() est utilisée pour traiter la réponse une fois que la requête HTTP a réussi. Elle reçoit un objet response contenant les données retournées.

response.data.AuthenticationResult : Cela contient les résultats d'authentification, notamment le AccessToken , qui est un token d'accès permettant de valider l'utilisateur pour des appels API futurs.

bru.setEnvVar('api\_shop\_token', token) : Le AccessToken récupéré est stocké dans une variable d'environnement avec le nom "api\_shop\_token" . Cela permet de l'utiliser ultérieurement pour authentifier les requêtes API.

```
36 * .catch(error => {
37     console.error('Échec :', error.response?.data || error.message);
38 });
```

.catch() : Cette méthode permet de capturer les erreurs si quelque chose échoue l'erreur est capturée et affichée.

error.response?.data : Permet d'afficher les données d'erreur.

Commentaire du code Post Response :

```
1 bru.setEnvVar("api_shop_token",res.body.AuthenticationResult.IdToken);
```

bru.setEnvVar() : Dans Bruno, il existe cette méthode pour définir des variables d'environnement. Elle permet de stocker des valeurs (dans ce cas, un token)

```
1 bru.setEnvVar("api_shop_token",res.body.AuthenticationResult.IdToken);
```

"api\_shop\_token" : Il s'agit de la variable d'environnement, qui contient le token d'authentification nécessaire pour accéder à l'API du magasin.

```
1 bru.setEnvVar("api_shop_token",res.body.AuthenticationResult.IdToken);
```

res.body.AuthenticationResult.IdToken : C'est la valeur de la variable d'environnement. Ici, tu récupères un Idtoken de la réponse d'authentification, et tu le stockes dans la variable d'environnement "api\_shop\_token" . Le token d'identification (IdToken) contient des informations sur l'utilisateur authentifié.

## COMPETENCES

### 1.1 Gérer le patrimoine informatique :

En intégrant un système d'authentification et en automatisant le renouvellement des tokens, il contribue à la gestion de la sécurité des services internes (comme l'authentification aux API), un aspect crucial de la gestion du patrimoine informatique.

### 1.3 Développer la présence en ligne de l'organisation :

Le développement et la gestion des API sécurisées sont essentiels pour garantir une expérience utilisateur fluide et fiable dans les applications en ligne, ce qui est un facteur clé pour développer la présence en ligne d'une organisation.

### 1.5 Mettre à disposition des utilisateurs un service informatique :

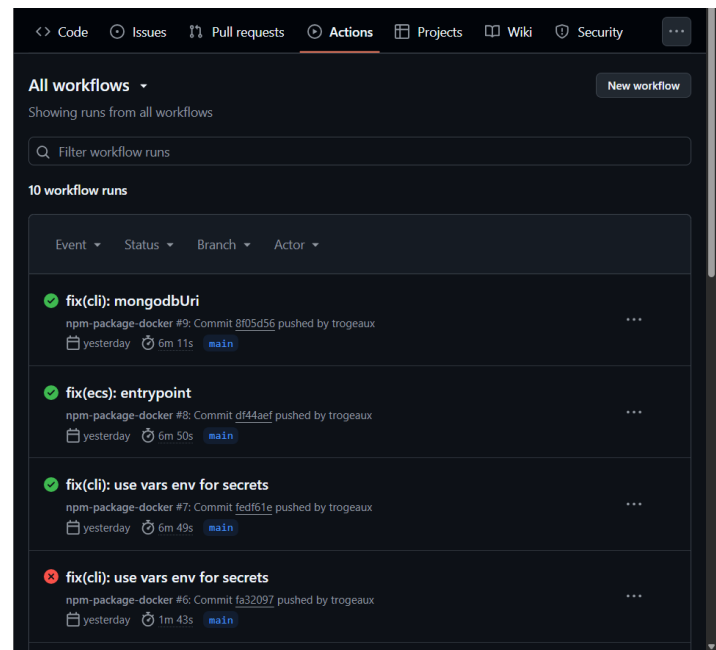
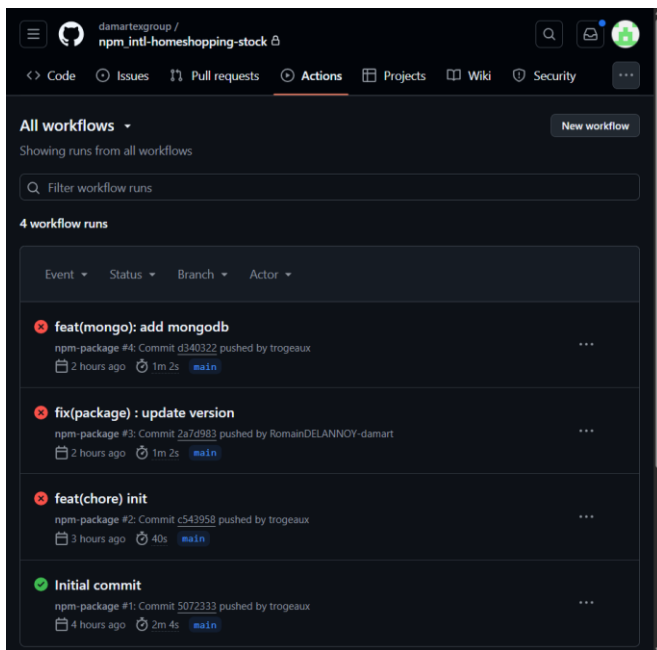
L'objectif principal du projet est de rendre accessible un service sécurisé aux utilisateurs, en leur permettant d'interagir avec des API de manière transparente, sans avoir à saisir de mots de passe à chaque requête.

En générant et en stockant des tokens d'authentification, je facilite l'accès sécurisé aux services tout en maintenant un haut niveau de sécurité, ce qui contribue directement à l'amélioration des services offerts aux utilisateurs finaux.



## Mission 3 : Migration de listener vers AWS

Migrer un listener de Google vers AWS signifie déplacer un service qui reçoit des données ou des événements depuis Google (comme Google Cloud ou Pub/Sub) vers Amazon Web Services (AWS). Cela implique de créer des services équivalents sur AWS (comme SNS ou SQS) pour recevoir ces événements et de modifier les configurations de l'application pour qu'elle envoie les données vers AWS au lieu de Google.



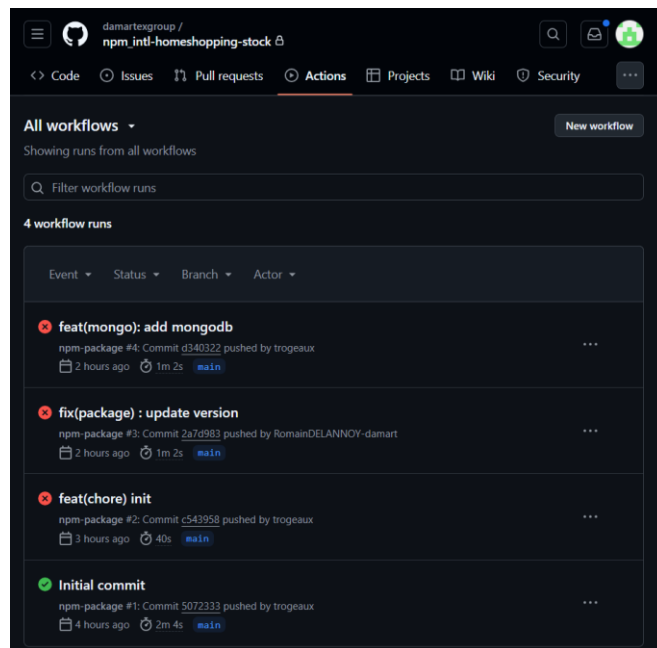
## COMPETENCES

### 1.1 Gérer le patrimoine informatique :

La migration implique de gérer et d'optimiser le listener Google en le déplaçant vers AWS. Cela inclut la gestion des ressources informatiques et la mise en place de solutions efficaces pour remplacer ou améliorer les services existants.

### 1.2 Répondre aux incidents et aux demandes d'assistance et d'évolution :

Pendant ou après la migration, des incidents peuvent survenir, comme des erreurs de configuration ou des problèmes de performance.




### 1.3 Développer la présence en ligne de l'organisation :

Migrer vers AWS fait partie d'une stratégie pour améliorer la performance des services en ligne de DAMARTEX. Cela peut avoir un impact direct sur la présence numérique de l'entreprise, notamment en garantissant une meilleure gestion des événements ou des requêtes.

## 1.4 Travailler en mode projet :

Cela implique la collaboration entre différentes équipes (comme les équipes développement, infrastructure et support) et un suivi des progrès, typique du travail en mode projet. Nous avons réunion tous les vendredis a 10H pour suivre les avancements de projet

21  
Ven

Platform Engineering - C  
Réunion Microsoft Teams  
Benoit LEWANDOWSK 

## **1.5 Mettre à disposition des utilisateurs un service informatique :**

En migrant le listener, on s'assure que le service reste disponible et performant pour répondre aux besoins des utilisateurs, même sur la nouvelle plateforme AWS.

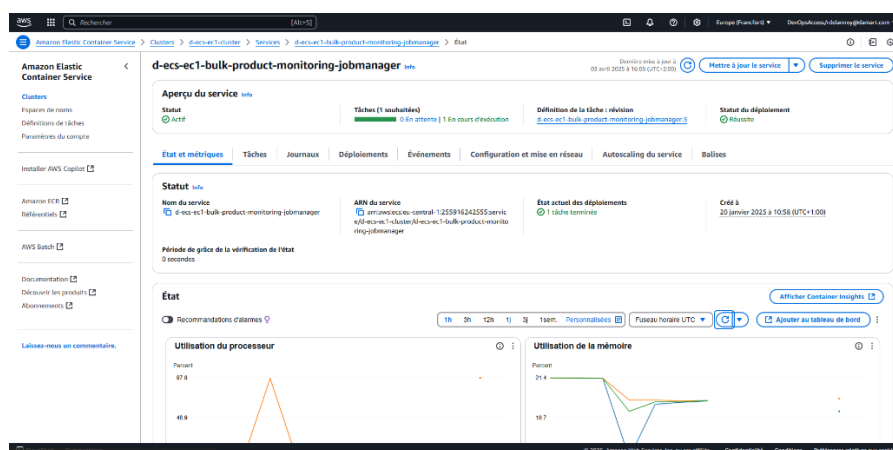
# Mission 4 : Intégrer et gérer des tâches/services ECS (Elastic Container Service) d'AWS

Le code permet de gérer des services ECS (Elastic Container Service) d'AWS. Il offre trois fonctionnalités principales :

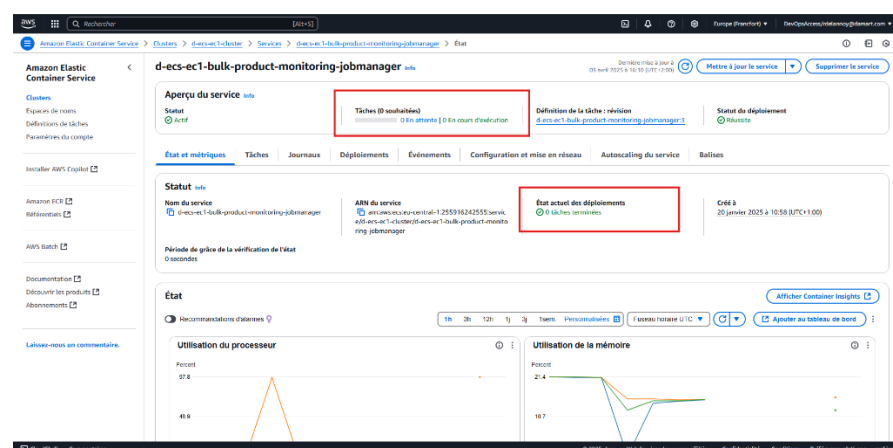
1. **Arrêter un service ECS** : en sauvegardant le nombre de tâches actives et en mettant ce nombre à zéro.
2. **Redémarrer un service ECS** : en restaurant le nombre de tâches précédemment sauvegardé.
3. **Vérifier l'état du service** : en suivant son statut (actif ou inactif) et en attendant son arrêt complet.

Le tout permet d'automatiser l'arrêt, le redémarrage et le suivi de services ECS tout en conservant les configurations initiales.

## AVANT le STOP



## APRES le STOP



Le code est le suivant :

```

from boto3 import Session
import time

boto_sess = Session(profile_name="damart-dev")
ecs_client = boto_sess.client('ecs')
ssm = boto_sess.client('ssm')

def stopEcsTask(cluster_name, service_name):
    response = ecs_client.describe_services(
        cluster=cluster_name,
        services=[service_name]
    )
    print(response)
    nr_tasks = response["services"][0]["desiredCount"]
    print(nr_tasks)

    ssm.put_parameter(
        Name="/ecs/{}/{}/desired_count".format(cluster_name, service_name),
        Value=str(nr_tasks),
        Type='String',
        Overwrite=True, # Overwrite a pour valeur True pour écraser la valeur existante et remplacé par le nombre actuel de tâches
    )
    print(response)

    response = ecs_client.update_service(
        cluster=cluster_name,
        service=service_name,
        desiredCount=0,
    )
    print(response)
    return response

def startEcsTask(cluster_name, service_name):
    try:
        response = ssm.get_parameter(
            Name=f"/ecs/{cluster_name}/{service_name}/desired_count"
        )

        nr_tasks = int(response["Parameter"]["Value"])
        print(f"Restoring desired count: {nr_tasks} for {service_name}")

        if nr_tasks > 0:
            update_response = ecs_client.update_service(
                cluster=cluster_name,
                service=service_name,
                desiredCount=nr_tasks
            )
            print(f"{service_name} started with {nr_tasks} tasks.")
            return update_response
        else:
            print(f"Cannot start {service_name}")
            return None
    except Exception as e:
        print(f"Error starting ECS task: {e}")
        return None

def getEcsTaskState(cluster_name, service_name):
    response = ecs_client.describe_services(
        cluster=cluster_name,
        services=[service_name]
    )
    if 'services' in response:
        service = response['services'][0]
        print(f"Service Status: {service['status']}")
    else:
        print(f"no services {service_name} found.")

def taskStatus(cluster_name, service_name):
    while True:
        task_state = getEcsTaskState(cluster_name, service_name)

        if task_state:
            print(task_state)
            if task_state == 'INACTIVE':
                print("The task has been stopped.")
                break
            time.sleep(5)

print("avant stop")
getEcsTaskState("d-ecs-ec1-cluster", "d-ecs-ec1-bulk-product-monitoring-jobmanager")
stopEcsTask("d-ecs-ec1-cluster", "d-ecs-ec1-bulk-product-monitoring-jobmanager")
print("après stop")
getEcsTaskState("d-ecs-ec1-cluster", "d-ecs-ec1-bulk-product-monitoring-jobmanager")
startEcsTask("d-ecs-ec1-cluster", "d-ecs-ec1-bulk-product-monitoring-jobmanager")
print("après start")
getEcsTaskState("d-ecs-ec1-cluster", "d-ecs-ec1-bulk-product-monitoring-jobmanager")
taskStatus("d-ecs-ec1-cluster", "d-ecs-ec1-bulk-product-monitoring-jobmanager")

```

## COMPETENCES

### **1.1 Gérer le patrimoine informatique :**

Le projet implique la gestion des services ECS, qui font partie de l'infrastructure cloud de l'organisation. Cela nécessite de maintenir et gérer les tâches et services ECS pour assurer la disponibilité et la performance des applications.

### **1.3 Développer la présence en ligne de l'organisation :**

En automatisant la gestion des services ECS, ce projet soutient la continuité de service des applications en ligne, ce qui contribue indirectement à la présence en ligne stable de l'organisation.

### **1.5 Mettre à disposition des utilisateurs un service informatique :**

En garantissant le bon fonctionnement et la disponibilité des services ECS, ce projet permet de mettre à disposition des utilisateurs des services informatiques fiables et performants.