ÉPREUVE E5
CONTRÔLE EN COURS DE FORMATION

Support et mise à disposition de service informatique

DELANNOY Romain
BTS SIO Lycée Gaston Berger

PRESENTATION DE L'ENTREPRISE WEPA

- **Création** : 1948
- **Siège social** : Arnsberg, Allemagne
- **Secteur** : Industrie du papier hygiénique et sanitaire (papier toilette, essuie-tout, mouchoirs, etc.)
- **Effectif** : Environ **4 000 collaborateurs**
- **Sites de production** : 13 usines en Europe (Allemagne, France, Italie, Pays-Bas, Royaume-Uni, Pologne)
- **Chiffre d'affaires** : Environ **1,6 milliard d'euros** (dernières données connues)
- **Classement européen** : 3e plus grand producteur de papier hygiénique en Europe
- **Production annuelle** : Plus de **850 000 tonnes** de papier sanitaire
- **Engagement durable** : Environ **40 %** des produits sont fabriqués à partir de fibres recyclées

REALISATION

Mission 1 : Installation de nouveau VeloCloud en coordination avec l'Allemagne

Mission 2 : Réorganisation des baies de brassage

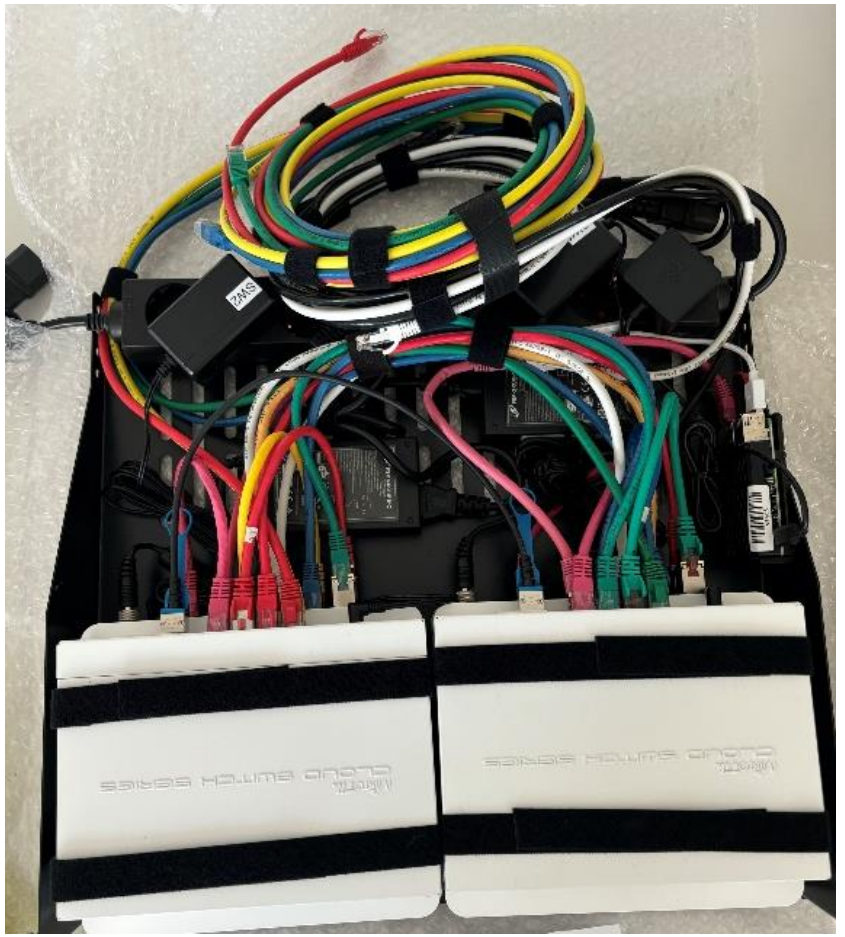
Mission 3 : Formatage de PC / création de session et affiliation à des groupes

Mission 4 : Intervention sur machine dans l'usine

MISSION 1 : INSTALLATION DE NOUVEAU VELOCLOUD EN COORDINATION AVEC L'ALLEMAGNE

L'installation de VeloCloud consiste à mettre en place une solution SD-WAN (réseau étendu défini par logiciel) de VMware.

Pour effectuer cela nous étions en coordination avec l'Allemagne pour débrancher l'ancien VeloCloud qui était déjà installé dans les baies de l'entreprise



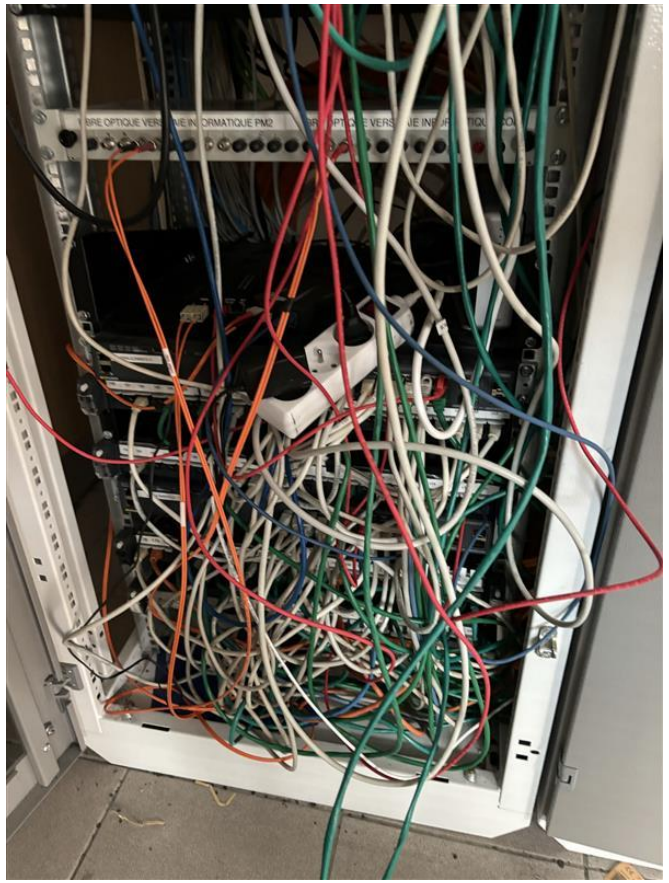

Cloud Router Switch
MikroTik
SW01
Cloud Router Switch
MikroTik
SW02
SW1
VCE1
VCE2
SW2

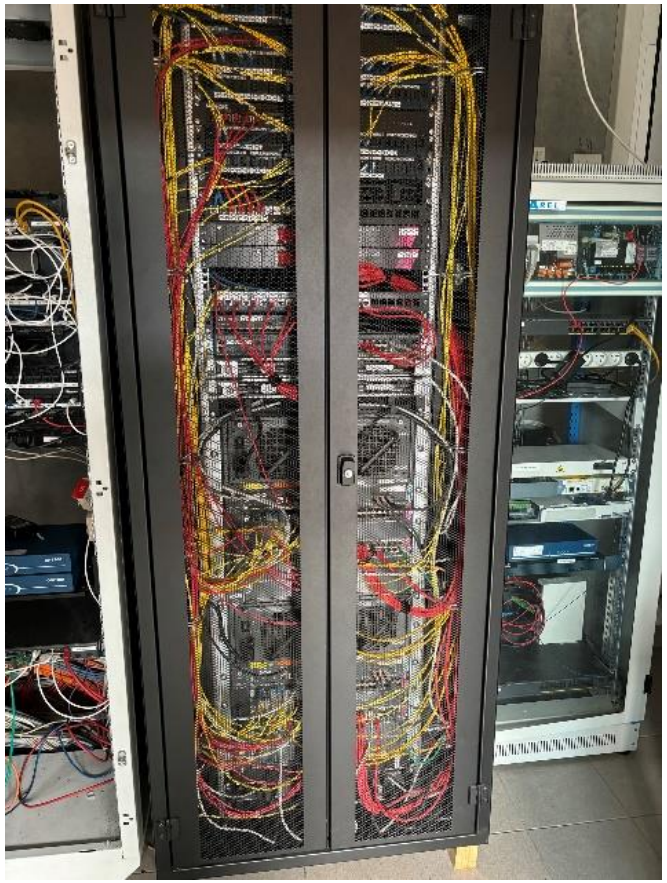

SD WAN
SD WAN

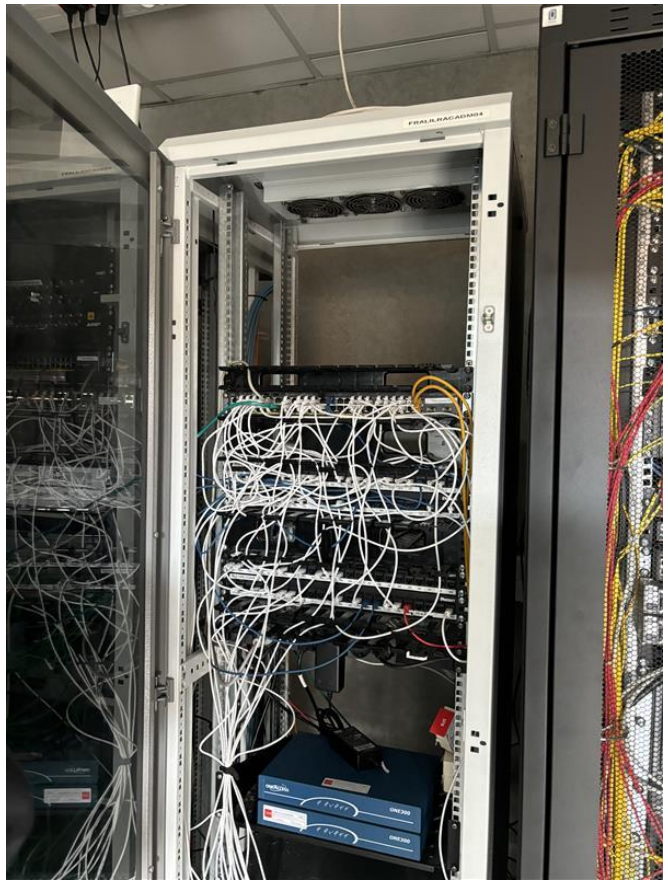
MISSION 2 : RÉORGANISATION DES BAIES DE BRASSAGE

Ma mission consistait à restructurer l'agencement des équipements pour optimiser l'installation, améliorer la gestion des câbles et faciliter la maintenance.

Cette réorganisation visait à réduire les risques d'erreurs, améliorer la performance du réseau et préparer l'infrastructure pour des extensions futures.





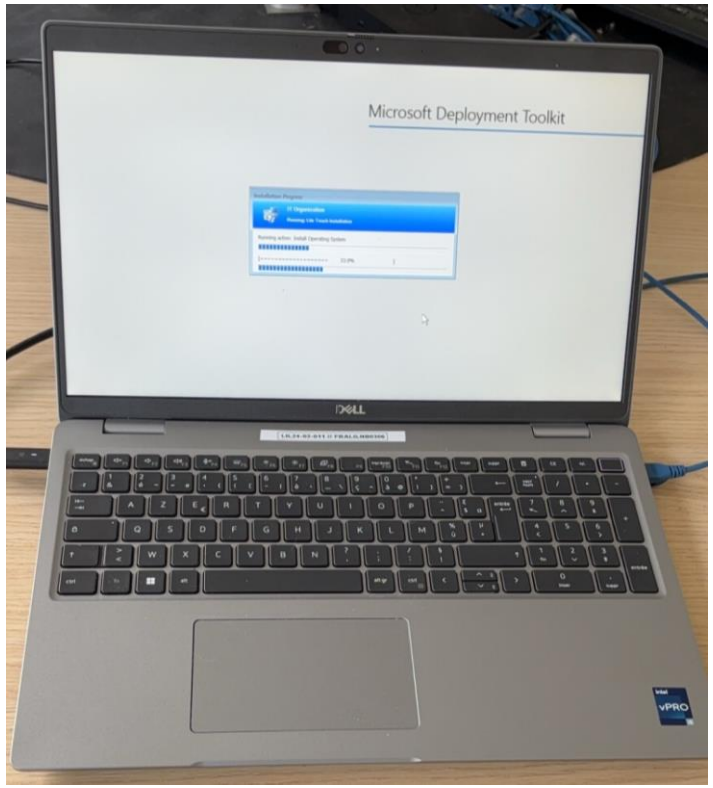


AVANT

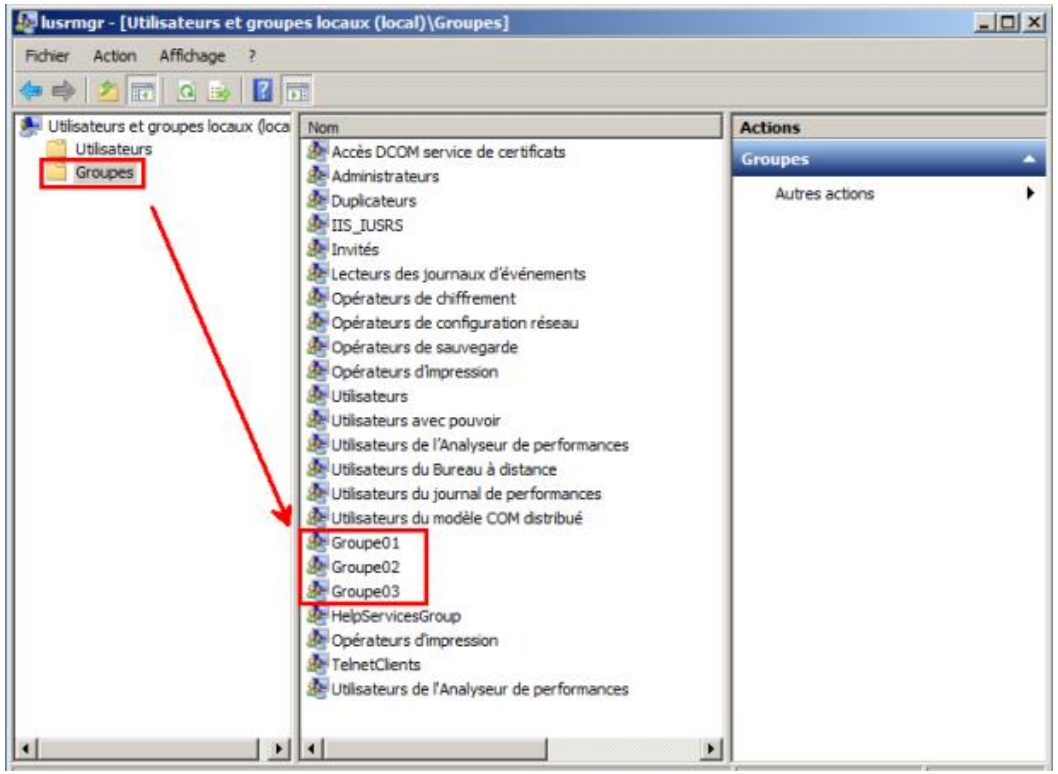
APRES

MISSION 3 : FORMATAGE DE PC / CRÉATION DE SESSION ET AFFILIATION À DES GROUPES

Ma mission consistait à formater (réinitialiser) des PC pour effacer toutes les données et y réinstaller une version propre de Windows. Cela permet de repartir à zéro en supprimant les fichiers, logiciels et paramètres, puis de créer une nouvelle session pour un nouvel utilisateur et de l'affilier aux groupes le concernant.



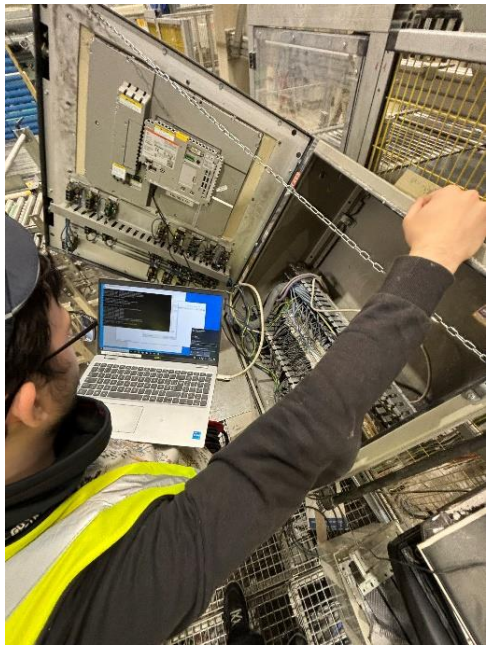
Formatage d’un PC en appuyant sur la touche F2 plusieurs fois jusqu'à ce que le message Entering System Setup s'affiche à son tour (DELL)



Exemple d’affiliation à des groupes

MISSION 4 : INTERVENTION SUR MACHINE DANS L'USINE

- Les interventions sur machines étaient souvent dues à des problèmes de pannes logiciels mais aussi à des problèmes de **console de contrôle** pour une machine automatisée.



PRESENTATION DE L'ENTREPRISE DAMARTEX

•**Création** : 1953

•**Siège social** : Roubaix, France

•**Secteur** : Distribution textile, spécialisée dans l'habillement et les produits de confort pour les seniors

•**Effectif** : Environ **2 800 collaborateurs**

•**Chiffre d'affaires** : Environ **540 millions d'euros** (2022/2023)

•**Présence géographique** : Actif principalement en France, Belgique, Royaume-Uni, Suisse et Allemagne

•**Nombre de marques** : 8 marques, dont **Damart**, **Xandres**, **Sedagyl**, **Delaby**,

•**Canaux de distribution** : Vente à distance (catalogue, e-commerce) et magasins physiques (environ **180 points de vente**)

REALISATION

Mission 1 : Industrialiser la configuration de 300 firewalls

Mission 2 : Script bruno en js avec call API

Mission 3 : Migration de listener vers AWS

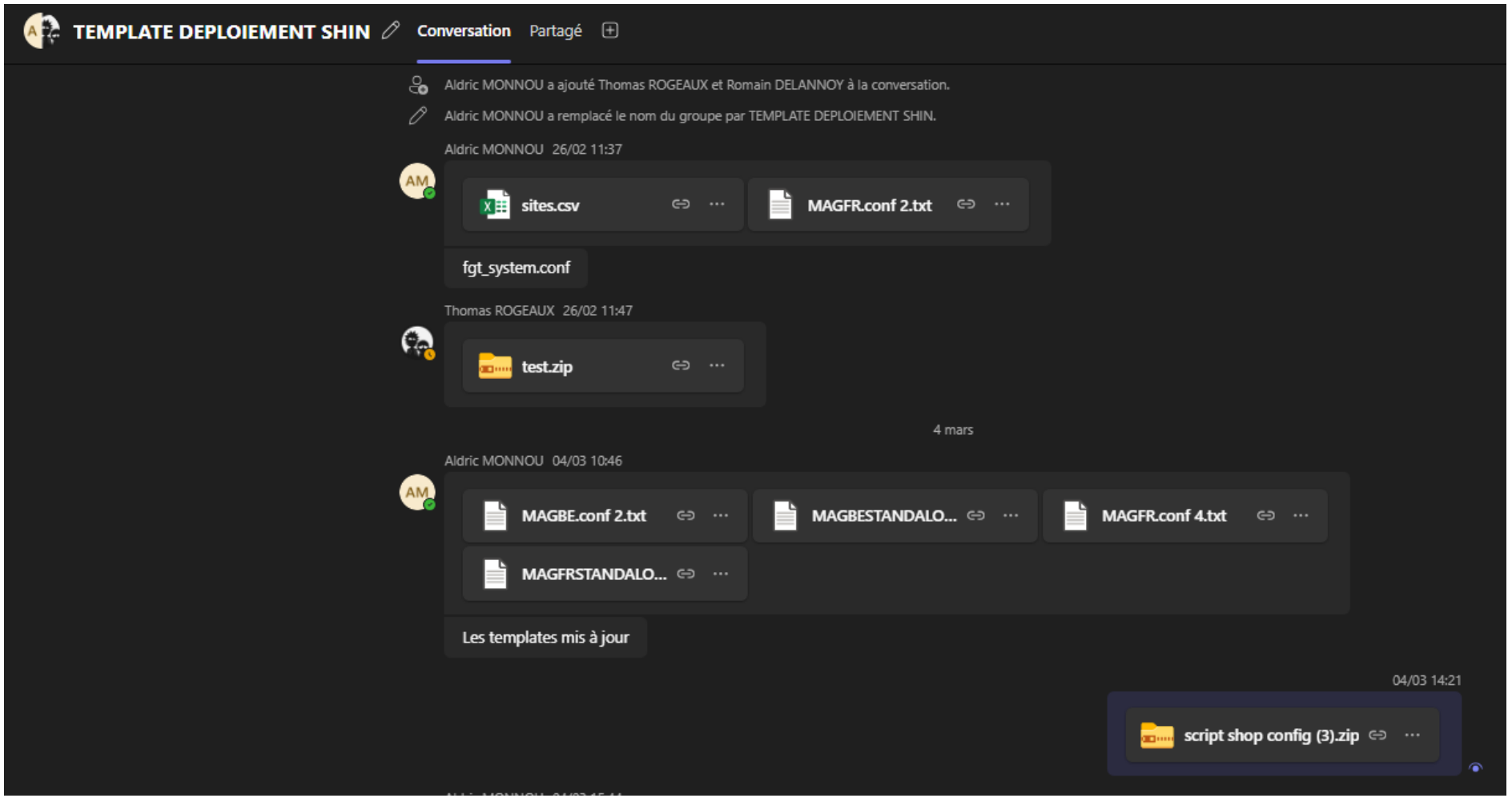
Mission 4 : Intégrer et gérer des tâches/services ECS (Elastic Container Service) d'AWS

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

```
#config-version=FGT40F-7.2.10-FW-build1706-240918:opmode=0:vdom=0:user=admin
#conf_file_ver=330768316443747
#buildno=1706
#global_vdom=1
config system global
    set admintimeout 15
    set alias "FortiGate-40F"
    set gui-auto-upgrade-setup-warning disable
    set hostname "FR[ID_MAG]FW[N_FW]"
    set switch-controller enable
    set timezone 28
end
config system accprofile
    edit "prof_admin"
        set secfabgrp read-write
        set ftviewgrp read-write
        set authgrp read-write
        set sysgrp read-write
        set netgrp read-write
        set loggrp read-write
        set fwgrp read-write
        set vpngrp read-write
```

	A	B	C	D	E	F	G	H	I	J	K	L
1	[Code Site]	[ID_MAG]	[N_FW]	[MAG_Y]	[LO_ID]	[HA_PRIORIT	[PASS]	[HIST_IP]	[HIST_MASK]	[HIST_GW]	[HIST_IP_STA	[HIST_IP_END]
2	900	900	1	4	102	250	MAGFR900!	10.33.214.32	255.255.255.2	10.33.214.62	10.33.214.33	10.33.214.61
3	900	900	2	4	102	200	MAGFR900!	10.33.214.32	255.255.255.2	10.33.214.62	10.33.214.33	10.33.214.61


TEMPLATE DEPLOIEMENT SHIN
Conversation
Partagé
Aldric MONNOU a ajouté Thomas ROGEAUX et Romain DELANNOY à la conversation.
Aldric MONNOU a remplacé le nom du groupe par TEMPLATE DEPLOIEMENT SHIN.
Aldric MONNOU 26/02 11:37
sites.csv
MAGFR.conf 2.txt
fgt_system.conf
Thomas ROGEAUX 26/02 11:47
test.zip
4 mars
Aldric MONNOU 04/03 10:46
MAGBE.conf 2.txt
MAGBESTANDALO...
MAGFR.conf 4.txt
MAGFRSTANDALO...
Les templates mis à jour
04/03 14:21
script shop config (3).zip

Untitled-1

```python
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'exixte 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")
```

MISSION 2 : SCRIPT BRUNO EN JAVASCRIPT AVEC CALL API

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

Le fait de stocker en variable le token permet le renouvellement automatique du token pour faciliter l'usage et garder l'accès sécurisé aux applications.

*API : **Application Programming Interface**, en français **Interface de Programmation d'Application**
Exemple : L'API de **Google Maps** permet d'afficher une carte sur un site web.

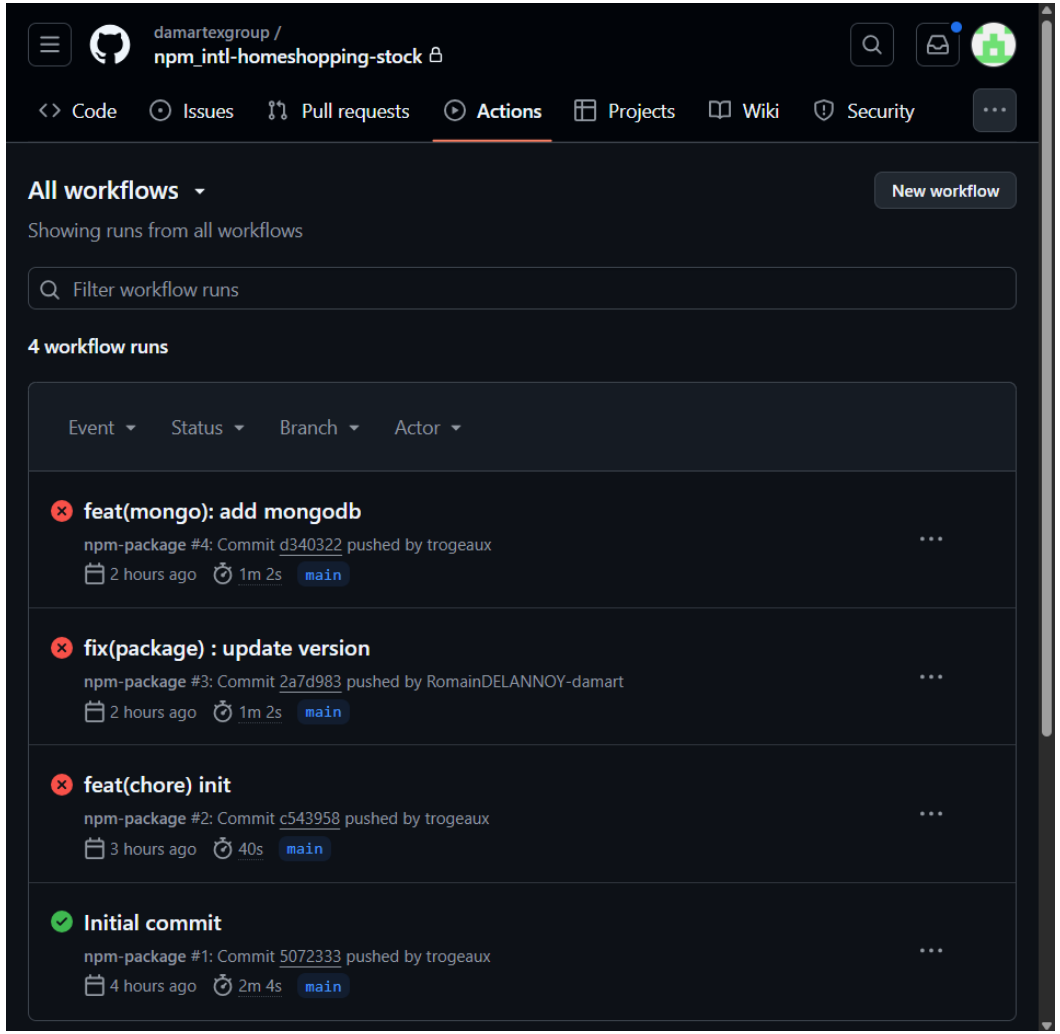
Untitled-1

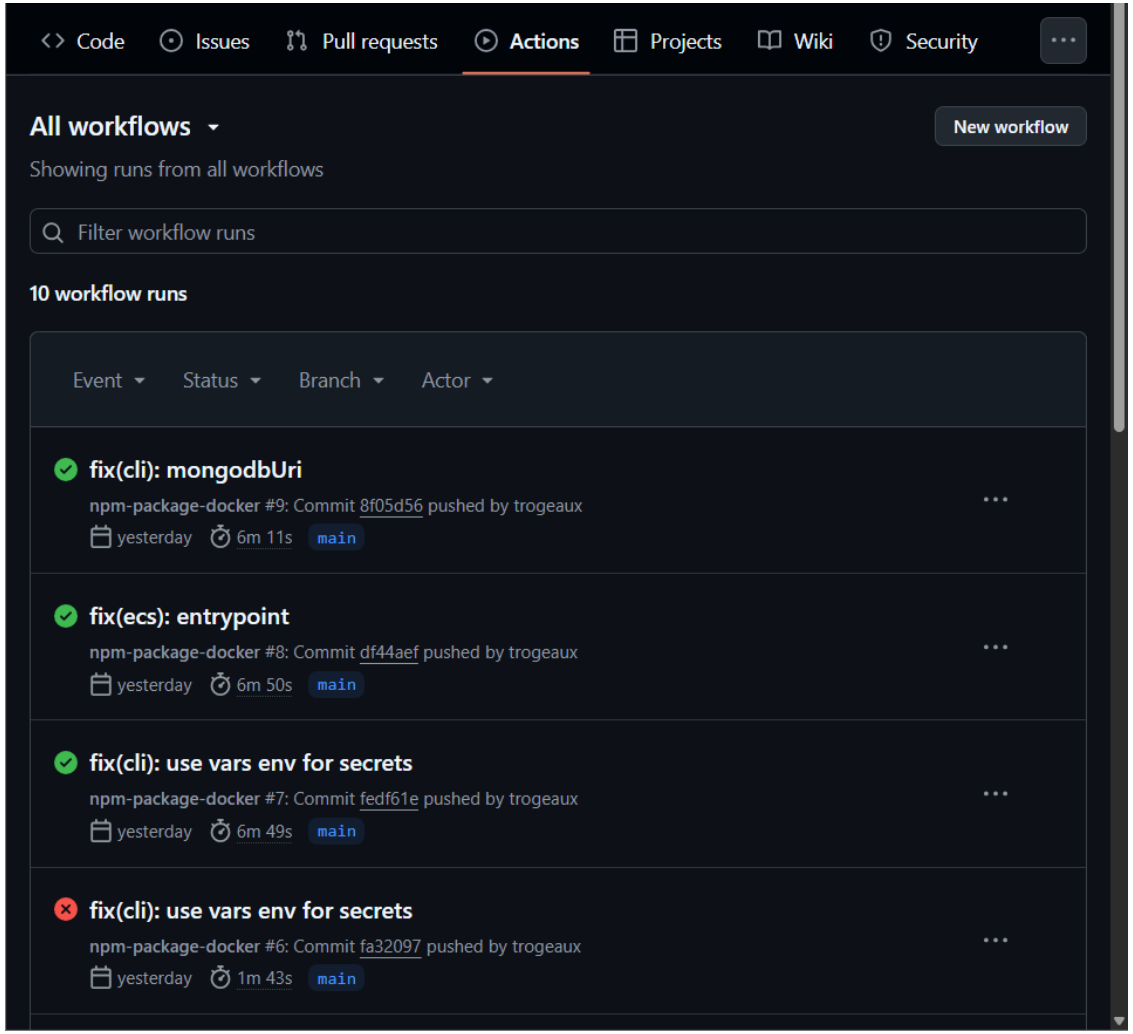
```
const axios = require('axios');

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);
  });
```

MISSION 3 : MIGRATION DE LISTENER VERS AWS

Migrer un listener* de Google vers AWS consiste à déplacer un service recevant des données ou des événements depuis Google (comme Google Cloud) vers Amazon Web Services (AWS). Cela implique de créer des services équivalents sur AWS (comme SNS ou ECS) pour recevoir ces événements et de modifier les configurations de l'application pour qu'elle envoie les données vers AWS au lieu de Google.

*Listener est un composant qui "écoute" et **réagit automatiquement** lorsqu'un certain événement se produit (connexion, requête, clic, etc.).


damartexgroup /
npm_intl-homeshopping-stock
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
All workflows
New workflow
Showing runs from all workflows
Filter workflow runs
4 workflow runs
Event
Status
Branch
Actor
feat(mongo): add mongodb
npm-package #4: Commit d340322 pushed by trogeaux
2 hours ago
1m 2s
main
fix(package) : update version
npm-package #3: Commit 2a7d983 pushed by RomainDELANNOY-damart
2 hours ago
1m 2s
main
feat(chore) init
npm-package #2: Commit c543958 pushed by trogeaux
3 hours ago
40s
main
Initial commit
npm-package #1: Commit 5072333 pushed by trogeaux
4 hours ago
2m 4s
main


Code
Issues
Pull requests
Actions
Projects
Wiki
Security
All workflows
New workflow
Showing runs from all workflows
Filter workflow runs
10 workflow runs
Event
Status
Branch
Actor
fix(cli): mongodbUri
npm-package-docker #9: Commit 8f05d56 pushed by trogeaux
yesterday
6m 11s
main
fix(ecs): entrypoint
npm-package-docker #8: Commit df44aef pushed by trogeaux
yesterday
6m 50s
main
fix(cli): use vars env for secrets
npm-package-docker #7: Commit fedf61e pushed by trogeaux
yesterday
6m 49s
main
fix(cli): use vars env for secrets
npm-package-docker #6: Commit fa32097 pushed by trogeaux
yesterday
1m 43s
main

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 sauvegardées.
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.

---

*ECS : **Elastic Container Service**. C'est un **service d'AWS (Amazon Web Services)** qui permet de **déployer, gérer et faire tourner des conteneurs Docker** dans le cloud

Untitled-1

```python
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")
```

Merci de votre attention