RAPPORT DE STAGE : TECHNICIEN RÉSEAU

Auteurs : Pierre FAMCHON
Tuteur en entreprise : M. Wilfried Quet
Lieu : Université de Technologie de Compiègne
Sujet : Déploiement d'outils de discovery et d'inventory de réseaux (SoT)
Formation : R&T - 2ème Année
Durée : 8 semaines | 14/04 - 06/06
Année : 2024-2025

Une photo de la façade du Centre Pierre Guillaumat 1, UTC

Sommaire

Remerciements

Je souhaite remercier l'ensemble des personnes qui m'ont permis d'obtenir ce stage ainsi que ceux qui m'ont assisté durant ces 2 mois au sein de la Direction des Systèmes d'Information (DSI).

Dans un premier temps, je remercie l'ensemble des intervenants présents à l'Université Technologique de Compiègne, d'avoir pris le temps de me conseiller et de moduler mon dossier de stage afin de commencer ce stage le plus rapidement que possible.

J'adresse mes plus grands remerciements à Monsieur Harry CLAISSE, Directeur de la DSI, de m'avoir accueilli au sein de ses locaux..

Je souhaite exprimer mes remerciements à mon tuteur au sein de la DSI, Monsieur Wilfried QUET, de m'avoir proposé ce stage, ses connaissances, son temps et sa patience dans cette longue aventure.

Je remercie également Madame Patricia HUGOT et Monsieur Rémy HUET pour leur patience, leur confiance et d'avoir consacré de leur temps à mes différentes questions.

Je remercie également tous les membres du Service Système Réseaux pour leurs explications, leur soutien et l'assistance qu'ils ont pu m'apporter durant ce stage.

Enfin, je salue avec gratitude l'ensemble du personnel de la Direction des Systèmes d'Information pour leur accueil chaleureux, leur aide ainsi que leur générosité dans cette aventure.

Introduction

Dans un contexte de profonde transformation numérique, la Direction des Systèmes d'Information (DSI) a récemment initié une phase cruciale de renouveau de son infrastructure réseau. Cette démarche stratégique vise à migrer vers un modèle plus robuste, vaste et sécurisé, afin de répondre aux exigences croissantes en matière de performance, de fiabilité et de protection des données.

Cependant, ce renouvellement a mis en lumière un défi majeur : l'absence d'outils dédiés à la découverte et à l'inventaire précis du réseau. Avant cette migration, la DSI ne disposait pas d'une "source de vérité" unique et centralisée permettant d'inventorier et d'administrer de manière exhaustive l'ensemble de ses composants, qu'ils soient matériels (baies, switchs, bornes Wi-Fi, postes de travail, etc.) ou virtuels (adresses IP, VLAN, VRF, tunnels, etc.).

Cette lacune a engendré un manque critique d'un inventaire complet, fiable et vérifiable en temps réel de l'infrastructure réseau. Une telle situation complexifie non seulement la gestion quotidienne et la maintenance préventive, mais elle représente également un frein majeur à l'optimisation des ressources et à la réactivité face aux incidents.

C'est dans ce contexte que s'inscrit ce rapport de stage. Il détaillera les missions et réalisations effectuées pour pallier ce manque, notamment par l'étude et la mise en place d'outils de supervision et d'inventaire tels que Netbox et NetDisco, essentiels pour rétablir une visibilité complète et une administration efficace de l'infrastructure réseau rénovée.

L’histoire de l’entreprise

1. Le fondateur, Guy Deniélou

Il existe que très peu d'informations sur l’enfance de Guy Deniélou. Pour résumer, il est né le 14 juin 1923 à Toulon d’un lieutenant de vaisseau, Albert Jean René DENIÉLOU. Il fit des études secondaires jusqu’à l’arrivée de la guerre où il les interrompit pour s’engager dans la Marine.

Une photo en uniforme de marin est disponible ci-dessous, Figure 1. À la suite de plusieurs péripéties, il effectua une formation complémentaire au sein de l’École Polytechnique pour ensuite continuer à Science Po.

Durant ces années dans l’armée, il effectua des études sur les sous-marins à propulsion nucléaire. Ce n’est qu’en 1959 où Guy Deniélou fut recruté par le Commissariat à l'énergie atomique et aux énergies alternatives, afin de mener des études et de la conception sur les réacteurs nucléaires

Figure 1 : Guy Deniélou en uniforme de marin,
source : Ecole Navale / Espace tradition / Officiers célèbres

2. La genèse de l'Université de Technologie de Compiègne

Les années 1960 marquèrent les débuts d'une idée. En effet, le retard lié à la technique en France fut important, dû à un certain mépris pour la technologie, très peu reconnu à l'époque et empêchant également l'accès à certains hauts postes. L'idée de la création d'une école focalisée sur les sciences appliquées et de créer un nouveau type d'ingénieur naquit.

En 1972, il quitta le CEA pour fonder l'Université de Technologie de Compiègne ainsi que le réseau de ces universités technologiques (« la première pierre » du Centre Benjamin Franklin, Figure 2). Cette université se veut être en étroite collaboration avec les industriels et participer aux conseils scientifiques afin de définir les futures politiques scientifiques.

Sa localisation n'est également pas anodine. Dans un objectif de réaménagement du territoire, l'Oise, en France, comme lieu pour ce prototype universitaire fut choisie : énormément d'espace, proche mais hors de la région parisienne, raccordée au réseau autoroutier et en lien avec le futur aéroport Roissy Charles de Gaulle.

Figure 2 : Pose de la première pierre du Centre Benjamin Franklin, source : Histoire de l'UTC - UTC

Aujourd'hui, l'UTC propose diverses formations dans les domaines de l'ingénierie : que ce soit dans l'informatique, la mécanique, la biologie ou encore le civil par exemple. Elle offre également la possibilité d'obtenir un double diplôme, qui est une valeur ajoutée vis-à-vis des autres universités.

D'un point de vue pédagogique, les étudiants sont libres de choisir les cours qu'ils souhaitent suivre à hauteur de 300 cours divisés en plusieurs catégories : Connaissances Scientifiques, Techniques et Méthodes et Technologie et Sciences de l'Homme. Son approche pédagogique met également l'accent sur les projets collaboratifs, les stages en entreprise et l'internationalisation. L'école encourage aussi l'esprit entrepreneurial et la prise d'initiative à travers les événements et les associations.

L'UTC fait partie du réseau des écoles du groupe des INSA (Instituts Nationaux des Sciences Appliquées) ce qui lui permet d'avoir une très forte influence et reconnaissance en France et à l'international (la carte des affiliées, Figure 3).

Figure 3 : La liste et localisation des universités partenaires
source : Mobilité sortante - UTC

3. La Direction des Systèmes d'Information

a. Historique de la création

La Direction des Systèmes d'Information (ou communément appelée DSI) a été créée le 1er janvier 2008 suivant les conseils du Ministère. L'ensemble du personnel du support informatique a été regroupé au sein d'une seule et unique entité nommée la DSI. Le Service Informatique et le Service Informatique de Gestion, ainsi que tous les informaticiens des différents départements, ont fusionné et ont été regroupés géographiquement afin de créer la DSI.

b. Organisation structurelle et les activités

La DSI, telle qu'on la connaît aujourd'hui, est composée de trois services :

- Service Assistance et Gestion de Parcs
- Service Système et Réseau
- Service Ingénierie des Applications

Évidemment, chaque service possède ses propres activités qui sont les suivantes :

- Le Service Assistance et Gestion de Parcs est essentiellement constitué de techniciens qui assurent :

    - La gestion du parc informatique pédagogique
    - La formation des utilisateurs (Word, Excel, Outlook, ...)
    - L'assistance aux utilisateurs (personnels et étudiants)
    - L'installation / la configuration / le dépannage des postes du personnel
    - Un ensemble de projets annexes (CIL, carte multiservice, photocopieurs, ...)

- Le Service Système et Réseau, constitué d'ingénieurs et d'un technicien qui assurent :

    - La sécurité informatique
    - La téléphonie
    - La gestion des systèmes et du réseau (local, métropolitain et régional)

- Le Service Ingénierie des Applications est constitué de 8 développeurs et 1 concepteur web design. Ce service assure tous les développements autour du système d'information tel que :

  - o L'interface avec les utilisateurs
  - o Le développement d'applications spécifiques
  - o Les développements des nouvelles fonctionnalités du Système d'Information
  - o La mise en place et la maintenance applicative des logiciels de l'AMUE

J'ai effectué mon stage au sein du Service Système et Réseau, ou SR, en compagnie de mon maître de stage, Monsieur Wilfried QUET. Un schéma plus complet, concernant les activités de la DSI, est disponible en Annexe. Enfin, voici l'organigramme ainsi que les personnes affiliées à la DSI :

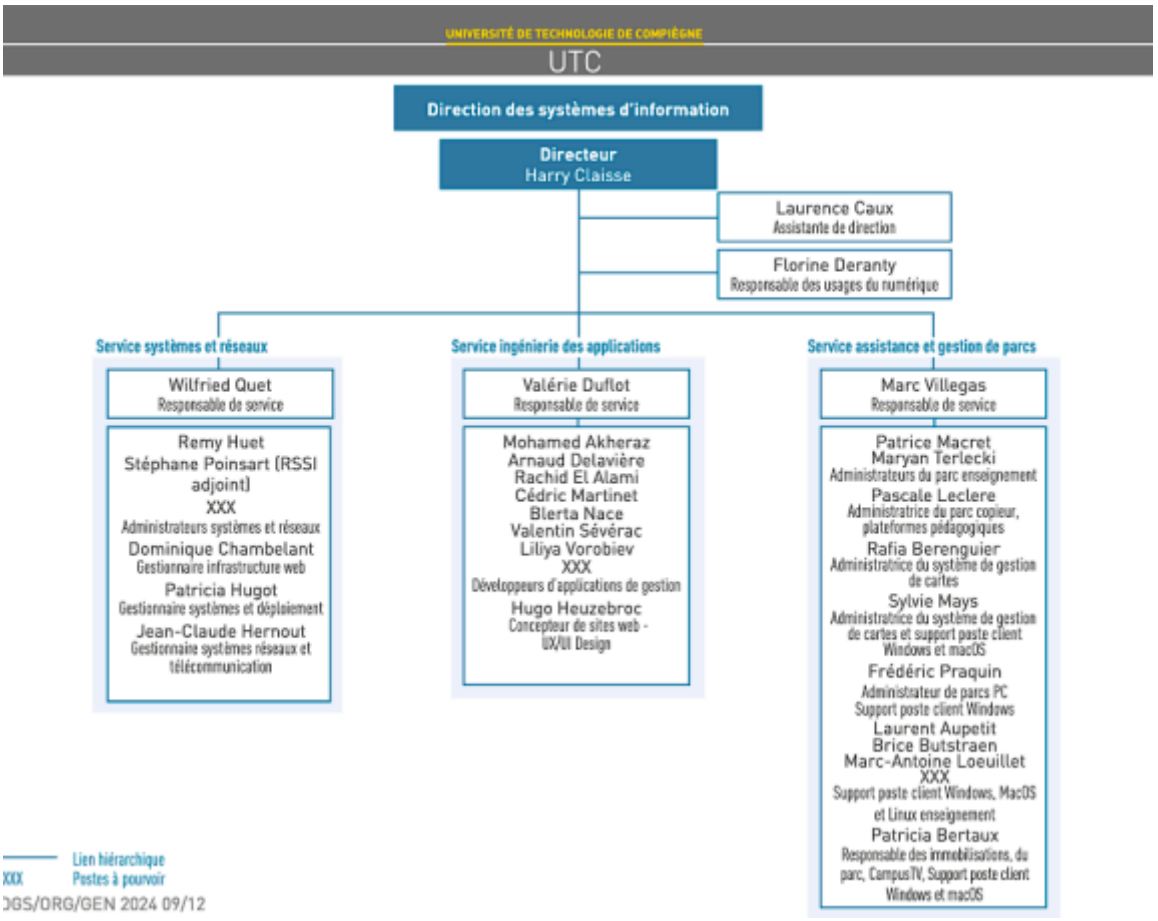


Figure 4 : Organigramme de la DSI

Missions

Au cours de ce stage, mes missions principales se sont articulées autour de l'étude, de la mise en place et du déploiement d'outils de supervision et de monitoring des éléments d'infrastructure réseau. L'objectif était d'améliorer la visibilité sur l'état du réseau, d'optimiser sa gestion et de garantir la disponibilité des services.

Mes responsabilités ont notamment inclus :

- L'étude et la mise en place d'outils de supervision d'éléments d'infrastructure : Cela a impliqué une recherche approfondie des solutions existantes sur le marché, avec un focus particulier sur Netbox et NetDisco. Cette phase a permis de comprendre leurs fonctionnalités, leurs avantages et leurs limites, afin de déterminer leur pertinence pour l'environnement réseau de l'entreprise.

- Le déploiement et la configuration de Netbox : J'ai été chargé de l'installation de Netbox, un outil essentiel pour la gestion de l'infrastructure réseau (IPAM et DCIM). Cette étape a nécessité la configuration précise des différents modules, l'intégration des équipements existants et la structuration des données pour une représentation fidèle de l'infrastructure.

- Le déploiement et la configuration de NetDisco : J'ai également mis en œuvre NetDisco, un outil complémentaire permettant la découverte automatique des équipements réseau. Cette tâche a impliqué la configuration des paramètres de découverte, la gestion des identifiants et l'intégration des données collectées pour une cartographie dynamique du réseau.

- La mise en place d'outils de monitoring sur les liens critiques de l'infrastructure : Au-delà de la supervision globale, une attention particulière a été portée à la surveillance des liens réseau vitaux. J'ai configuré des systèmes de monitoring spécifiques pour ces liens, permettant une détection proactive des problèmes et une réactivité accrue en cas d'incident, contribuant ainsi à la stabilité et à la performance de l'ensemble de l'infrastructure.

Réalisations

Au cours de mon stage, les principales réalisations ont porté sur l'étude, le déploiement, la configuration et l'intégration de deux outils fondamentaux pour la supervision et l'inventaire de l'infrastructure réseau : Netbox et NetDisco. Ces implémentations ont été cruciales pour répondre au besoin identifié d'une “source de vérité complète” et fiable pour les équipements réseau de la DSI, notamment suite à une période de renouveau et de migration vers une infrastructure plus robuste, vaste et sécurisée.

1. NETBOX

Netbox a été sélectionné comme l'outil central de gestion de l'infrastructure réseau (IPAM et DCIM), essentiel pour inventorier et administrer les équipements physiques et virtuels. Sa capacité à offrir une vue d'ensemble détaillée et structurée de l'infrastructure en fait un pilier de la gestion moderne des réseaux.

a. Découverte de l'outil Netbox

Netbox est une application web open-source de gestion d'adresses IP (IPAM - *IP Address Management*) et de gestion de l'infrastructure de centre de données (DCIM - *Data Center Infrastructure Management*). Conçu spécifiquement pour répondre aux besoins des professionnels de l'IT souhaitant documenter et modéliser leurs réseaux, il se distingue par sa capacité à maintenir une **"Source of Truth"** fiable et centralisée pour l'ensemble de l'infrastructure réseau.

Son architecture est basée sur le framework web Django (Python), ce qui lui confère robustesse, extensibilité et une grande flexibilité. Les données sont stockées dans une base de données PostgreSQL, réputée pour sa fiabilité et sa performance dans la gestion de grands volumes de données relationnelles. Netbox ne se limite pas à un simple inventaire statique ; il permet de modéliser des relations complexes entre les différents composants réseau, incluant :

- **Adresses IP et sous-réseaux (IPAM)** : Gestion des préfixes, adresses IP individuelles, VLANs, VRF (Virtual Routing and Forwarding) et tunnels VPN. Cette fonctionnalité est cruciale pour éviter les conflits d'adresses et pour une planification rigoureuse de l'adressage réseau.

- **Équipements physiques (DCIM)** : Inventaire des baies, des dispositifs (commutateurs, routeurs, serveurs, bornes Wi-Fi, pare-feu), de leurs emplacements physiques, de leurs types, de leurs rôles et de leurs connexions physiques et logiques (câblage, interfaces).

- **Composants modulaires** : Prise en charge des modules d'interfaces, des alimentations électriques et d'autres composants internes des équipements.

- **Opérateurs et circuits** : Documentation des circuits de données fournis par des opérateurs externes.

- **Virtualisation** : Gestion des clusters, des machines virtuelles et de leurs connexions réseau.

L'interface utilisateur de Netbox est intuitive et riche en fonctionnalités, permettant des recherches rapides, des filtrages complexes et une visualisation claire de la topologie. De plus, son API RESTful exhaustive permet une automatisation poussée des tâches d'inventaire et d'administration, ce qui est un avantage considérable dans les environnements dynamiques comme celui de la DSI.

Contexte de la DSI : Pour la DSI, l'adoption de Netbox est une réponse directe à l'absence d'un inventaire complet et fiable après la migration des équipements. Sans une vision claire de l'infrastructure matérielle (baies, switchs, bornes Wi-Fi, PC) et virtuelle (IP, VLAN, VRF, tunnels), la gestion, le dépannage et la planification des évolutions étaient devenus particulièrement complexes. Netbox a été identifié comme l'outil capable de centraliser ces informations et de servir de référence unique pour tous les intervenants, garantissant ainsi une meilleure cohérence des données et une efficacité accrue des opérations.

b. Déploiement de Netbox

Le déploiement de Netbox a été réalisé en utilisant la technologie des conteneurs Docker et l'outil d'orchestration Docker Compose. Ce choix technique a été motivé par plusieurs avantages significatifs :

- **Isolation de l'environnement** : Docker permet d'encapsuler l'application Netbox et toutes ses dépendances (base de données PostgreSQL, Redis pour le cache) dans des conteneurs isolés, évitant ainsi les conflits avec d'autres applications ou librairies présentes sur le système hôte.

- **Portabilité et reproductibilité** : Les configurations Docker Compose garantissent que l'environnement de Netbox peut être déployé de manière identique sur n'importe quel serveur compatible Docker, facilitant la reproductibilité de l'installation et les futurs transferts ou mises à jour.

- **Facilité de gestion** : Docker Compose simplifie la gestion de l'ensemble des services nécessaires à Netbox (base de données, application web, worker, etc.) via un seul fichier de configuration (`docker-compose.yml`) et des commandes unifiées.

- **Rapidité de déploiement** : L'utilisation de conteneurs pré-configurés réduit considérablement le temps et la complexité de l'installation par rapport à un déploiement manuel de chaque composant.

Voici les étapes détaillées du processus de déploiement :

Mise à jour des paquets système et installation des prérequis Docker :
Avant toute installation, il est impératif de s'assurer que le système d'exploitation est à jour pour bénéficier des dernières corrections de sécurité et des dépendances logicielles.

```
sudo apt update
sudo apt install -y ca-certificates curl gnupg lsb-release
```

Ces commandes mettent à jour la liste des paquets disponibles et installent les outils nécessaires pour gérer les certificats SSL, télécharger des fichiers (curl), gérer les clés GPG et identifier la version de la distribution Linux.

Ensuite, la clé GPG officielle de Docker a été ajoutée au trousseau de clés du système. Cette étape est cruciale pour que le gestionnaire de paquets APT puisse vérifier l'authenticité des paquets Docker et s'assurer qu'ils proviennent d'une source fiable.

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

Le dépôt Docker officiel a été ajouté à la liste des sources APT. Cela permet au système de savoir où trouver les paquets Docker à installer.

La commande `lsb_release -cs` insère dynamiquement le nom de code de la distribution Ubuntu (`stable` dans ce cas), assurant que le bon dépôt est utilisé.

```
echo "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
```

1. **Installation de Docker Engine et Docker Compose Plugin :**

Une fois le dépôt ajouté, les paquets Docker Engine (`docker-ce`, `docker-ce-cli`, `containerd.io`) et le plugin Docker Compose (v2) ont été installés.

```
sudo apt update
```

```
sudo apt install -y docker-ce docker-ce-cli containerd.io
docker-compose-plugin
```

- `docker-ce` est le moteur Docker de la communauté.
- `docker-ce-cli` est l'outil en ligne de commande pour interagir avec Docker.
- `containerd.io` est un *runtime* de conteneurs de haut niveau.
- `docker-compose-plugin` est la version v2 de Docker Compose, intégrée en tant que plugin Docker, simplifiant son utilisation (`docker compose` au lieu de `docker-compose`).

2. **Démarrage et activation du service Docker :**

Après l'installation, le service Docker a été démarré et configuré pour se lancer automatiquement au démarrage du système.

```
sudo systemctl start docker
sudo systemctl enable docker
```

Ceci assure que Docker est opérationnel et persistant après un redémarrage du serveur.

3. **Récupération des fichiers de Netbox Docker :**

Les configurations Docker Compose pour Netbox sont fournies par la communauté via un dépôt GitHub. Le dépôt a été cloné et le répertoire de travail a été défini sur ce nouveau dossier.

```
git clone https://github.com/netbox-community/netbox-docker.git
cd netbox-docker
```

Il est à noter qu'il est possible de spécifier une version précise de Netbox à utiliser en modifiant le fichier `.env` ou en le créant s'il n'existe pas, par

exemple `echo "VERSION=v4.2.3" > .env`. Cela garantit la stabilité et la reproductibilité des déploiements.

4. **Lancement des conteneurs Netbox avec Docker Compose :**

Les images Docker nécessaires pour Netbox (application, base de données, Redis, etc.) ont été téléchargées, puis les conteneurs ont été démarrés en mode détaché (`-d`).

```
sudo docker compose pull
sudo docker compose up -d
```

`docker compose pull` télécharge les images les plus récentes spécifiées dans le `docker-compose.yml`. `docker compose up -d` démarre tous les services définis dans le fichier `docker-compose.yml` en arrière-plan.

Pour vérifier le bon fonctionnement des conteneurs et accéder à leurs logs en cas de problème, les commandes suivantes ont été utilisées :

```
sudo docker compose ps
sudo docker compose logs netbox
```

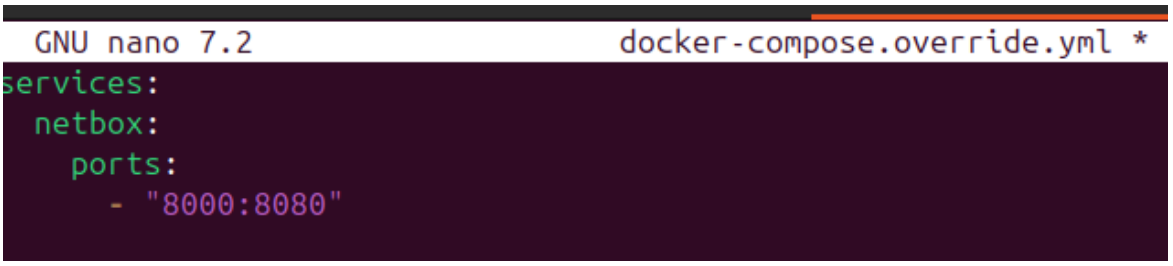
`docker compose ps` liste les conteneurs et leur statut. `docker compose logs netbox` affiche les logs du conteneur spécifique de l'application Netbox.

5. **Exposition du port web de Netbox :**

Par défaut, l'image Docker de Netbox n'expose pas directement son port web (8080 interne au conteneur) à l'hôte. Pour rendre l'interface web accessible depuis l'extérieur du serveur, une modification a été apportée au fichier `docker-compose.override.yml`. Ce fichier permet de surcharger les configurations du `docker-compose.yml` principal sans le modifier directement.

```
nano docker-compose.override.yml
```

Le contenu suivant a été ajouté ou modifié pour mapper le port 8000 de l'hôte au port 8080 du conteneur `netbox`:



Après cette modification, les conteneurs ont été relancés pour appliquer les changements :

```
sudo docker compose down
sudo docker compose up -d
```

`docker compose down` arrête et supprime les conteneurs existants, et `docker compose up -d` les recrée avec la nouvelle configuration de port.

6. Accès à l'interface web et création d'un superutilisateur :

Une fois le port exposé, l'interface web de Netbox est devenue accessible via un navigateur web en utilisant l'adresse IP du serveur et le port 8000 (ex: `http://192.168.100.160:8000`).

Pour la première connexion, il est nécessaire de créer un superutilisateur (administrateur). Ceci se fait en accédant au shell du conteneur Netbox et en exécutant une commande Django :

```
sudo docker exec -it netbox-docker-netbox-1 bash
python3 /opt/netbox/netbox/manage.py createsuperuser
```

Il a été demandé de fournir un nom d'utilisateur (ex: `admin`) et un mot de passe (`Progtr00`). L'adresse email est un champ facultatif.

Pour des raisons de sécurité, le mot de passe de l'utilisateur `admin` a été réinitialisé après la création initiale pour s'assurer qu'il soit fort et ne soit pas un mot de passe par défaut connu :

```
python3 /opt/netbox/netbox/manage.py changepassword admin
```

Enfin, un redémarrage des conteneurs Netbox a été effectué pour s'assurer que toutes les configurations soient bien prises en compte:

```
sudo docker compose restart
```

Accès à la page de connexion

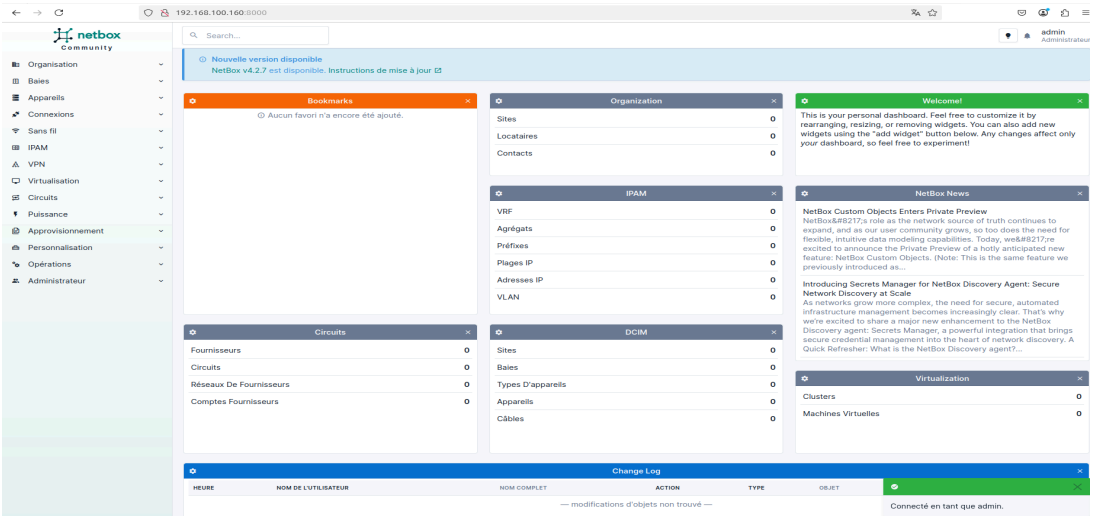
7. Génération du token :

Une fois connecté avec le superutilisateur, il a été possible de générer un token d'API Netbox.

Ce token est essentiel pour permettre à des scripts externes (comme ceux développés pour l'importation de données) d'interagir de manière sécurisée avec l'API RESTful de Netbox.

Ce token, par exemple `04946ef59ffeb57bc7a0e7c6ac73f787eb272c57`, sera utilisé dans la configuration des scripts d'automatisation.

Page d'accueil de Netbox :



c. Configuration de Netbox

La phase de configuration de Netbox a été axée sur la préparation de l'importation massive et automatisée des données de l'infrastructure réseau. L'objectif était de structurer un processus permettant de peupler Netbox avec des informations complètes et précises, en minimisant les interventions manuelles et en garantissant la cohérence des données.

1. Arborescence des fichiers du dépôt Netbox Docker :

Avant de commencer la configuration, une analyse de l'arborescence des fichiers clonés du dépôt `netbox-docker` a été effectuée. Cela a permis de comprendre l'organisation du projet, les fichiers de configuration importants (`configuration.py`, `docker-compose.yml`, `docker-compose.override.yml`), les scripts (`docker-entrypoint.sh`), et les répertoires d'environnement (`env`). Cette compréhension est fondamentale pour toute modification ou personnalisation future. L'outil `tree -a` a été utilisé pour visualiser cette arborescence.

2. Création des répertoires pour l'importation des fichiers :

Afin d'organiser les scripts d'automatisation et les fichiers de données, une nouvelle structure de répertoires a été mise en place, distincte du répertoire `netbox-docker`. Le répertoire `netbox-device-autodiscovery` a été créé, avec un sous-répertoire `import_yaml` pour les scripts et un sous-répertoire `network_devices` pour les fichiers de données YAML.

```
mkdir netbox-device-autodiscovery/import_yaml
cd netbox-device-autodiscovery
mkdir network_devices
```

Cette organisation permet de maintenir une séparation claire entre les configurations officielles de Netbox et les scripts développés spécifiquement pour l'entreprise, facilitant la gestion des versions et les mises à jour.

3. Découpage des données réseau en fichiers YAML structurés :

Le fichier `network_devices.yaml` initial, qui contenait l'inventaire des équipements (obsolète mais constituant la base de mon backend), a été découpé en plusieurs sous-fichiers YAML, organisés par type d'équipement. Cette granularité simplifie grandement l'exécution des scripts d'importation, car elle permet de traiter des catégories spécifiques d'équipements et de minimiser les erreurs. L'arborescence résultante ressemble à ceci:

```
- network_devices/
  ├── access-points/
  │   ├── ap-bf.yaml
  │   ├── ap-cima.yaml
  │   └── ...
  ├── captive-portals/
  │   └── captive_portals.yaml
  ├── radius/
  │   └── radius.yaml
  ├── switches/
  │   ├── sw-5000.yaml
  │   ├── sw-bf.yaml
  │   └── ...
  ├── vpn-servers/
  │   └── vpn_servers.yaml
  └── wlc/
      └── wlc.yaml
```

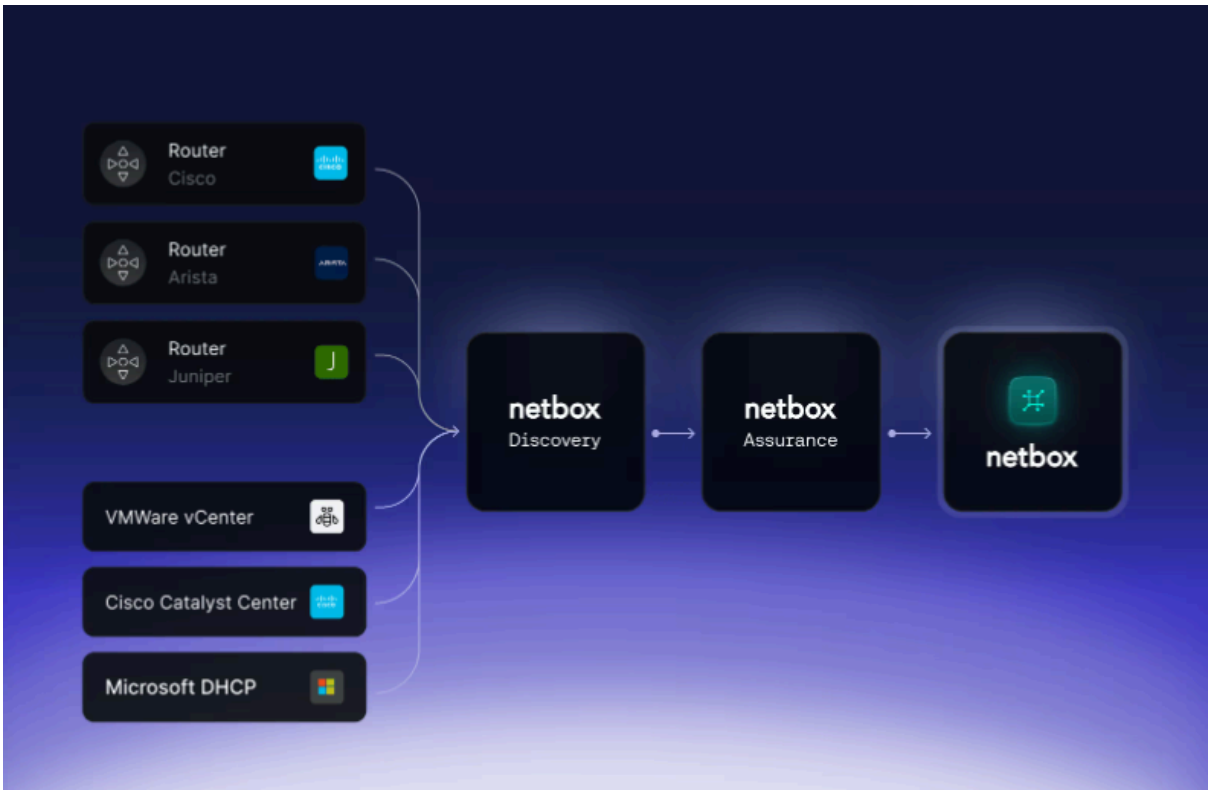
Chaque fichier YAML contient la description d'un ou plusieurs équipements de même type, avec leurs propriétés (nom, type, adresse IP, adresse MAC, rôle, ports, description, etc.). Ce format structuré est essentiel pour que les scripts Python puissent interpréter correctement les données et les mapper aux objets correspondants dans Netbox.

4. Développement des scripts Python pour l'interaction avec l'API Netbox :

Un ensemble de scripts Python a été développé pour automatiser le processus d'importation des données YAML dans Netbox via son API RESTful. Ces scripts sont situés dans le répertoire `import_yaml`.

Ce répertoire contient :

```
├── network_devices        # répertoire contenant les .yaml
├── __pycache__
│
├── main.py
├── device_manager.py
├── netbox_api.py
├── netbox_config.py
└── yaml_processor.py
```



Automatisation possible de Netbox
source : https://netboxlabs.com

main.py : C'est le point d'entrée principal pour l'exécution du processus d'importation. Il est responsable de définir le chemin d'accès au fichier YAML que `yaml_processor.py` doit utiliser.

```
import os
from yaml_processor import load_and_process_yaml  # Importation
correcte de ton module

def main():
    # Chemin relatif du fichier YAML dans le sous-dossier
network_devices
    yaml_path = "network_devices/access-points/ap-rob.yaml"
    print("🚀 Démarrage de l'importation des équipements vers
NetBox...")
    load_and_process_yaml(yaml_path)  # Appel de la fonction de
traitement
    print("✅ Importation terminée.")

if __name__ == "__main__":
    main()
```

Ce script permet de lancer le processus pour un fichier YAML spécifique, facilitant le débogage et l'importation sélective.

device_manager.py : Ce module est central pour la gestion des équipements et de leurs interfaces dans Netbox. Il contient des fonctions pour :

- `get_interface_id_by_name(device_id, interface_name)` :

  Récupère l'ID d'une interface spécifique pour un périphérique donné, en interrogeant l'API Netbox.

- `device_exists(device_name)` :

  Vérifie si un équipement avec un nom donné existe déjà dans Netbox. Cette fonction est cruciale pour déterminer si une opération de création ou de mise à jour est nécessaire.

- `create_or_update_device(payload, existing_device=None)` :

  Créer un nouvel équipement ou met à jour un équipement existant dans Netbox. Cette fonction prend en charge le "payload" (les

données de l'équipement) et gère la logique de création (`POST`) ou de mise à jour (`PATCH`) en fonction de l'existence de l'équipement. Les codes de statut HTTP (200, 201, 204) sont vérifiés pour confirmer le succès de l'opération.

- `get_device_type_id(device_type_name)` :

  Récupère l'ID du type de périphérique correspondant au nom fourni (ex: "cisco-switch").

- `get_site_id(site_name)` :

  Récupère l'ID du site (emplacement physique) dans Netbox.

- `create_interfaces(device_id, port_list, mgmt_only)` :

  Créer les interfaces pour un équipement donné. Il vérifie d'abord si l'interface existe déjà ; si ce n'est pas le cas, il la crée avec un type par défaut (par exemple, "1000base-t") et la propriété `mgmt_only` (interface de gestion uniquement). Les IDs des interfaces créées ou récupérées sont stockées.

- `assign_mac_to_interface(interface_id, mac_address)` :

  Assigne une adresse MAC à une interface spécifique. La fonction gère la création de l'adresse MAC si elle n'existe pas et son association à l'interface, ou la mise à jour si l'adresse MAC existe mais n'est pas encore assignée. Une fonction utilitaire `format_mac_address` assure que l'adresse MAC est au bon format (`XX:XX:XX:XX:XX:XX`).

- `assign_ip_to_device(device_id, ip_address)` :

  Assigne une adresse IP à la première interface trouvée pour un équipement donné (ou crée l'adresse IP si elle n'existe pas).

`netbox_api.py` : Ce module regroupe des fonctions utilitaires pour simplifier les interactions récurrentes avec l'API Netbox, fournissant des méthodes pour récupérer les IDs de différents objets Netbox. Il utilise les configurations de `netbox_config.py`.

- `get_device_type_id(device_type_name)` :

Récupère l'ID d'un type de périphérique par son modèle.

- `device_exists(device_name)` :

  Vérifie si un périphérique existe par son nom, retournant l'objet périphérique si trouvé.

- `get_device_role()` :

  Récupère l'ID d'un rôle de périphérique (ex: "Wi-Fi AP").

- `get_site_id(site_name)` :

  Récupère l'ID d'un site par son nom.

- `create_or_update_device(device_payload, existing_device=None)` :

  Une version plus générique pour créer ou mettre à jour un périphérique, incluant l'assignation du rôle.

- `get_device_role_id(role_name)` :

  Récupère l'ID d'un rôle de périphérique générique.

`netbox_config.py` : Ce fichier centralise les paramètres de connexion à l'API Netbox, rendant les autres scripts indépendants de ces valeurs et facilitant leur mise à jour.

```
NETBOX_URL = "http://192.168.100.160:8000/api/"
NETBOX_TOKEN = "04946ef59ffeb57bc7a0e7c6ac73f787eb272c57"

HEADERS = {
     "Authorization": f"Token {NETBOX_TOKEN}",
     "Content-Type": "application/json",
}
DEBUG_MODE = True
```

`yaml_processor.py` : C'est le module principal qui orchestre le processus d'importation. Il charge le fichier YAML, itère sur chaque équipement défini et appelle les fonctions appropriées de `device_manager.py` et

`netbox_api.py` pour créer ou mettre à jour les équipements, leurs interfaces, adresses IP et MAC dans Netbox.

La fonction `load_and_process_yaml(file_path)` est le cœur de ce module. Elle lit le fichier YAML, extrait les informations pour chaque périphérique (nom, type, IP, MAC, rôle, ports), formate les adresses MAC si nécessaire, récupère les IDs des types de périphériques, rôles et sites, puis appelle `create_or_update_device` pour gérer l'équipement lui-même. Ensuite, elle traite la liste des ports pour créer les interfaces (y compris la gestion des plages de ports `G0/1-24`) et assigne les adresses IP et MAC aux interfaces correspondantes.

élévation de Baie de la DSI, réalisé suite à mon référencement de ces dernières

2. NETDISCO

NetDisco a été intégré en complément de Netbox pour sa capacité à découvrir automatiquement les équipements réseau et à collecter des informations détaillées via des protocoles comme SNMP. Il joue un rôle crucial en alimentant Netbox avec des données dynamiques et à jour, garantissant que l'inventaire reste pertinent au fil des évolutions de l'infrastructure.

a. Découverte de l'outil Netdisco

NetDisco est une application open-source basée sur Perl, conçue pour la découverte, l'inventaire et la cartographie des réseaux informatiques. Il se positionne comme un outil d'audit et de monitoring passif, capable d'interroger les équipements réseau (commutateurs, routeurs, pare-feu, points d'accès) pour récupérer des informations précieuses sur leur configuration, leurs interfaces, leurs tables de routage, leurs MAC-adresses, et leurs connexions physiques (via CDP, LLDP, FDP).

Les fonctionnalités clés de NetDisco incluent :

- **Découverte automatique (Discovery)** : NetDisco peut scanner des plages d'adresses IP ou des sous-réseaux pour identifier les équipements actifs. Il utilise principalement le protocole SNMP (Simple Network Management Protocol) pour interroger les dispositifs, mais peut également s'appuyer sur d'autres méthodes pour identifier les connexions et les propriétés des équipements.

- **Collecte de données (Collect)** : Une fois un équipement découvert, NetDisco collecte une quantité significative de données :
    - Informations système (nom d'hôte, modèle, numéro de série, version du firmware).
    - Interfaces (nom, type, statut, vitesse, adresses MAC).
    - Tables de pontage (MAC-adresse / port).
    - Tables ARP (adresse IP / MAC-adresse).
    - Informations VLAN.
    - Détails sur les ports PoE.

- **Cartographie de la topologie (Topology Mapping)** : En utilisant les informations de protocole de découverte de voisins (CDP pour Cisco, LLDP pour des équipements multi-constructeurs), NetDisco peut établir une carte des interconnexions entre les équipements réseau, offrant une visualisation graphique de la topologie.

- **Inventaire et recherche** : Toutes les données collectées sont stockées dans une base de données (généralement PostgreSQL), permettant des recherches rapides et la génération de rapports détaillés sur l'état du réseau.

- **Alertes** : Bien que moins orienté monitoring temps réel que d'autres outils, NetDisco peut générer des alertes sur des changements d'état ou des découvertes d'équipements non autorisés.

Contexte de la DSI : Dans le cadre du renouvellement de l'infrastructure, l'intégration de NetDisco répond au besoin urgent d'une découverte automatisée et continue des équipements réseau. La DSI se trouvait sans un inventaire complet et à jour en temps réel. NetDisco permet de combler cette lacune en scannant le réseau pour identifier tous les nouveaux équipements ou les changements sur les équipements existants. Ses données seront ensuite exploitées pour maintenir Netbox à jour, assurant ainsi la fiabilité de la "source de vérité" de l'infrastructure. L'automatisation de la découverte réduit considérablement la charge de travail manuelle et garantit que l'inventaire dans Netbox reflète toujours la réalité du terrain.

b. Déploiement de Netdisco

Le déploiement de NetDisco, à l'instar de Netbox, a été réalisé en utilisant Docker et Docker Compose. Cette approche garantit la même isolation, portabilité et facilité de gestion pour l'environnement NetDisco, qui se compose de plusieurs services (application web, backend de découverte, base de données PostgreSQL).

Les étapes détaillées du déploiement sont les suivantes :

1. Récupération des fichiers NetDisco Docker :

La première étape consiste à récupérer les configurations Docker Compose de NetDisco depuis son dépôt GitHub officiel.

```
git clone https://github.com/netdisco/netdisco.git
cd netdisco
```

Cette commande télécharge l'intégralité du projet NetDisco, y compris les fichiers Docker Compose et les configurations par défaut.

2. Création des répertoires et ajustement des permissions :

NetDisco nécessite des répertoires spécifiques pour stocker ses logs et sa configuration locale, et ces répertoires doivent avoir des permissions d'écriture pour l'utilisateur sous lequel NetDisco s'exécute dans le conteneur (généralement l'UID 901).

```
mkdir logs config nd-site-local
chmod 777 logs config nd-site-local
```

- `logs` : Pour les fichiers de log de l'application NetDisco.
- `config` : Pour les fichiers de configuration de NetDisco (comme `deployment.yml`).
- `nd-site-local` : Pour les personnalisations spécifiques au site ou les plugins.
- `chmod 777` : est une permission temporaire très permissive. Pour un environnement de production, il faudrait affiner ces permissions pour qu'elles soient moins ouvertes, par exemple en attribuant la propriété au groupe de l'utilisateur NetDisco ou en utilisant des ACLs plus granulaires une fois l'UID de l'utilisateur Docker connu.

Cependant, pour une installation rapide et un environnement de stage, cela permet de s'assurer qu'il n'y a pas de problème de permission bloquant.

3. Installation et Lancement initial de Docker Compose :

Une fois les répertoires préparés, les images Docker de NetDisco sont téléchargées et les conteneurs sont lancés.

```
docker compose pull
docker compose up -d
```

- `docker compose pull` : Télécharge les images Docker nécessaires pour tous les services définis dans le `docker-compose.yml` de NetDisco (par exemple, `netdisco/netdisco`, `postgres`).
- `docker compose up -d` : Démarre tous les services en arrière-plan. Cela inclut le conteneur de la base de données PostgreSQL, le backend de NetDisco (pour la découverte), le service web et d'autres services auxiliaires.

4. Configuration des fichiers Docker Compose et NetDisco :

La configuration est une étape cruciale pour adapter NetDisco à l'environnement réseau spécifique de la DSI.

Modification de `docker-compose.yml` : Bien que le `docker-compose.yml` par défaut soit fonctionnel, il a pu nécessiter des ajustements pour des mappings de ports spécifiques, des volumes persistants ou des configurations réseau avancées si l'intégration avec d'autres outils (comme Netbox sur un autre conteneur) l'exigeait.

Pour NetDisco, il est crucial que les services (web, backend, db) puissent communiquer entre eux. Le fichier par défaut est généralement bien configuré pour cela.

Configuration de `deployment.yml` : Ce fichier est le cœur de la configuration de NetDisco. Il se trouve dans le répertoire `netdisco/netdisco/config` (ou mappé dans `./config` sur l'hôte).

deployment.yml - Configuration Netdisco

```
database:
  name: 'netdisco'
  user: 'netdisco'
  pass: 'netdisco'
  #host: 'netdisco-postgresql'
  port: 5432

domain_suffix: 'localdomain'
site_name: 'Réseau SNMP'

# Clé de session requise pour Netdisco Web
#session_cookie_key:
'd43b52e4f1d44f39b681db9494a7d2cf0fc2e2a9a79be30fc2d4039e037f47b2
'
device_auth:
  - tag: 'snmpv3'
     user: stagiaire
     auth:
     pass: nJS9cq5TFwWnQs
     proto: SHA
     priv:
     pass: DHGC5uxBGBpn4d
     proto: AES

# Pas d'authentification (utilisateur guest en admin)
no_auth: true

# Détection automatique
discover_no:
  - '127.0.0.1'
```

Les paramètres clés à configurer incluent :

- `db` : Les informations de connexion à la base de données PostgreSQL. Dans un environnement Docker Compose, `host=db` signifie que le service `backend` de NetDisco se connecte au service `db` via le nom de service Docker Compose.

- `site` : Des informations générales sur le déploiement de NetDisco (nom, logo, etc.).
- `discovery` : C'est la section la plus importante. Elle contient les informations nécessaires à NetDisco pour interroger les équipements réseau :
- `default_community` : Les communautés SNMPv2c à essayer.
- `snmp_versions` : Les versions de SNMP à utiliser (ici 2c et 3).
- `snmpv3` : La configuration détaillée pour les identifiants SNMPv3 (nom d'utilisateur, protocoles d'authentification et de confidentialité, et leurs phrases secrètes).

5. Redémarrage et mise à jour de la Base de Données NetDisco :

Après toute modification de `deployment.yml` ou `docker-compose.yml`, les conteneurs doivent être redémarrés pour que les changements soient pris en compte.

```
sudo docker compose restart
```

Il est parfois nécessaire de redémarrer spécifiquement le conteneur `netdisco-web` si des problèmes de connexion à la base de données persistent, pour s'assurer qu'il recharge bien sa configuration.

Enfin, le schéma de la base de données de NetDisco doit être créé ou mis à jour.

```
sudo docker exec -it netdisco-backend bin/netdisco-db-deploy
```

Cette commande exécute le script `netdisco-db-deploy` à l'intérieur du conteneur `netdisco-backend`. Ce script crée toutes les tables nécessaires, les vues et les fonctions dans la base de données PostgreSQL pour que NetDisco puisse stocker ses données. Après l'exécution réussie, il est recommandé de redémarrer à nouveau les services `netdisco-web` et `netdisco-backend` pour s'assurer qu'ils utilisent le schéma de base de données à jour.

```
sudo docker compose restart netdisco-web netdisco-backend
```

c. Configuration

La configuration de NetDisco a été validée par une série de tests rigoureux, s'assurant de sa capacité à interagir correctement avec les équipements réseau de la DSI et à collecter des informations fiables.

1. Tests de connectivité inter-conteneurs et applicative :

Il est primordial de s'assurer que les différents composants de l'application NetDisco peuvent communiquer entre eux, ainsi qu'avec les ressources externes (les équipements réseau).

Résolution DNS interne des conteneurs : Un test a été effectué pour vérifier que le conteneur `netdisco-backend` pouvait résoudre le nom d'hôte de la base de données.

```
sudo docker exec -it netdisco-backend ping netdisco-postgresql
```

- Un ping réussi vers `db` (le service PostgreSQL) confirme que la résolution DNS interne fonctionne correctement.

- Si des erreurs de connexion (utilisateur/mot de passe incorrects, base de données non trouvée) apparaissent, elles sont généralement visibles dans les logs du conteneur `netdisco-backend`.

Accessibilité IP depuis le conteneur backend : Pour que NetDisco puisse découvrir les équipements réseau, le conteneur `netdisco-backend` doit avoir une connectivité réseau vers le réseau de la DSI.

```
sudo docker exec -it netdisco-backend ping 172.20.0.42
 # Exemple d'IP d'un équipement réseau de la DSI
```

- Un ping réussi indique que le routage et les règles de pare-feu entre le conteneur Docker et le réseau interne sont correctement configurés. C'est une étape fondamentale avant toute tentative de découverte SNMP.

2. **Test SNMPv3 dans le conteneur du backend :**

La découverte de NetDisco repose fortement sur SNMP. Un test direct depuis le conteneur `netdisco-backend` a été effectué pour valider que les identifiants SNMPv3 configurés (`stagiaire`, phrases secrètes SHA et AES) permettaient bien d'interroger un équipement cible.

Installation des outils SNMP dans le conteneur : Par défaut, les outils de ligne de commande SNMP (comme `snmpwalk`) peuvent ne pas être installés dans l'image Docker de NetDisco. Pour les besoins de test, ils ont été installés temporairement.

```
sudo docker exec -u root -it netdisco-backend sh
apk update
apk add net-tools net-snmp
```

- `apk update` met à jour la liste des paquets Alpine Linux (la base de l'image Docker NetDisco). `apk add net-tools net-snmp` installe les outils réseau classiques et la suite d'outils SNMP.

Exécution de `snmpwalk` : Une fois les outils installés, un `snmpwalk` direct a été exécuté pour vérifier l'authentification et le chiffrement SNMPv3.

```
sudo docker exec -it netdisco-backend   snmpwalk -v3 -u stagiaire
-l authPriv -a SHA -A nJS9cq5TFwWnQs -x AES -X DHGC5uxBGBpn4d
172.20.0.42
```

`-v3` : Spécifie la version SNMP 3.

`-u stagiaire` : Le nom d'utilisateur SNMPv3.

`-l authPriv` : Indique que l'authentification et la confidentialité (chiffrement) sont utilisées.

`-a SHA -A nJS9cq5TFwWnQs` : Protocole d'authentification SHA et sa phrase secrète.

`-x AES -X DHGC5uxBGBpn4d` : Protocole de confidentialité AES et sa phrase secrète.

`172.20.0.42` : L'adresse IP de l'équipement réseau cible (un switch, routeur, etc.).

Un résultat affichant des OID SNMP et leurs valeurs (par exemple, des informations sur le système ou les interfaces) a confirmé que les paramètres SNMPv3 étaient corrects et que NetDisco serait en mesure d'interroger cet équipement. Une erreur ici aurait indiqué un problème d'identifiants, de protocole ou de configuration SNMP sur l'équipement lui-même.

3. **Test de découverte réseau détaillée (mode debug avancé) :**

Le test ultime de la configuration de NetDisco est l'exécution d'une découverte complète sur un équipement cible en mode débogage avancé. Cela permet de vérifier le bon fonctionnement de l'outil et d'observer les logs détaillés de chaque étape du processus de découverte.

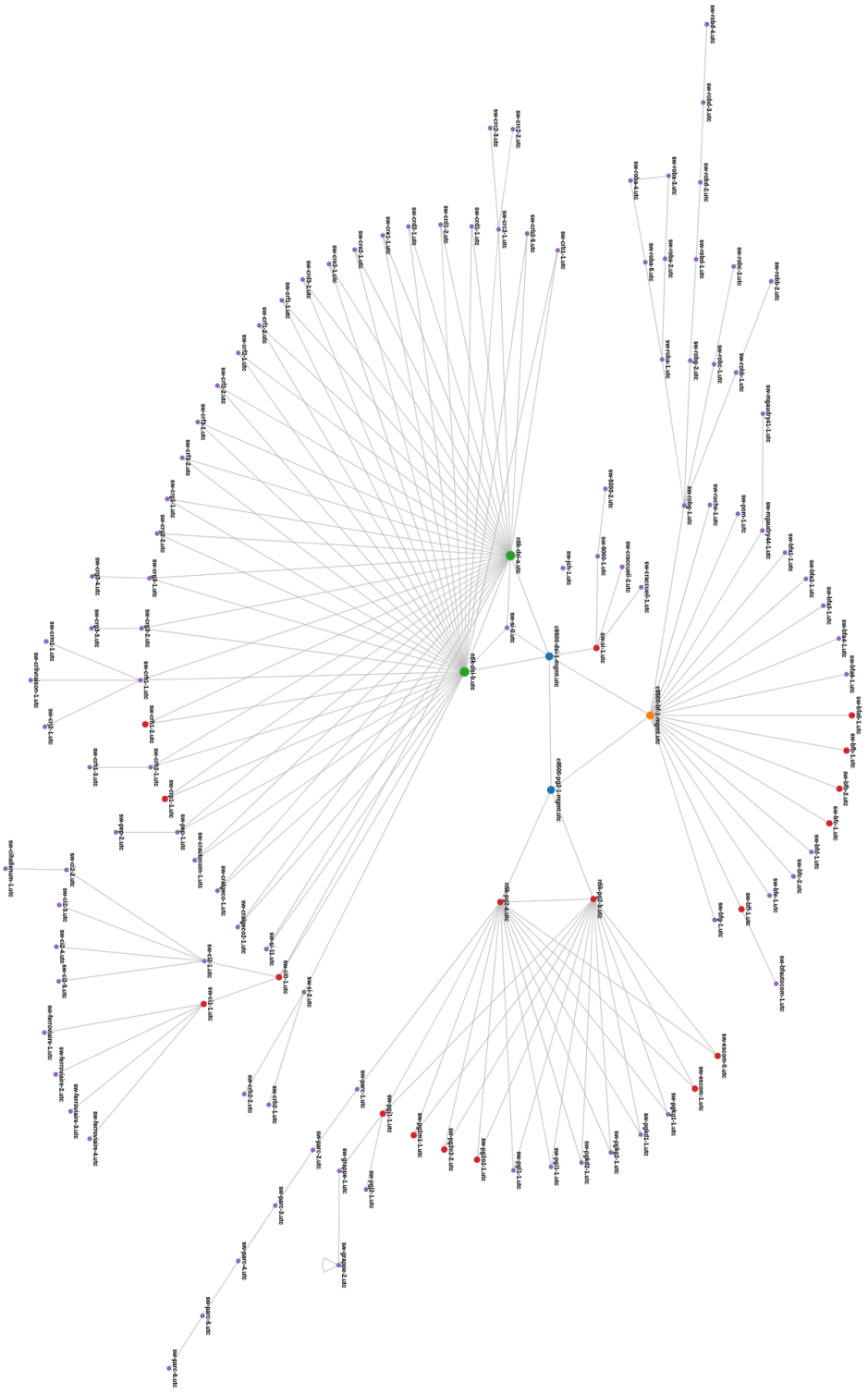
```
sudo docker exec -it netdisco-backend bin/netdisco-do discover
-DDDD -d 172.20.0.42
```

- `bin/netdisco-do discover` : La commande NetDisco pour lancer une découverte.
- `-DDDD` : Active le niveau de débogage le plus élevé, fournissant des informations très granulaires sur les étapes de la découverte (tentatives SNMP, requêtes MIB, traitement des réponses).
- `-d 172.20.0.42` : Spécifie l'équipement cible par son adresse IP.

L'analyse des logs générés par cette commande a permis de :

- Confirmer que NetDisco a pu établir une session SNMPv3 avec l'équipement.
- Vérifier que les MIBs nécessaires ont été interrogées et que des données significatives ont été retournées (informations système, interfaces, tables MAC, etc.).
- Identifier d'éventuels problèmes de parsing de données ou de compatibilité avec des équipements spécifiques.
- Observer la progression de la découverte, y compris les tentatives de détection de voisins via CDP/LLDP.

d. Cartographie de la topologie Réseau



Carte du réseau de la DSI obtenue grâce à netdisco

4. **Gestion des Logs et Erreurs (non nuisibles) :**

Au cours du déploiement et des tests, l'observation des logs des conteneurs est essentielle.

Accès aux logs globaux :

```
sudo docker compose logs -f -t
```

Cette commande permet de suivre en temps réel (`-f`) les logs de tous les services Docker Compose, avec un horodatage (`-t`). Cela a été utile pour détecter les problèmes au démarrage des conteneurs ou lors des premières tentatives de découverte.

Erreurs "DB unversioned" et "session_cookie_key" : certaines erreurs ou avertissements peuvent apparaître dans les logs mais ne sont pas nécessairement nuisibles.

- `DBIx::Class::Schema::Versioned::_on_connect(): Your DB is currently unversioned. Please call upgrade on your schema to sync the DB.` Cette erreur indique que la base de données n'a pas été "versionnée" ou que son schéma n'est pas à jour. Cependant, si le script `netdisco-db-deploy` a été exécuté avec succès, cette erreur peut parfois apparaître de manière transitoire ou être un avertissement résiduel, indiquant qu'une mise à jour de schéma a déjà été effectuée ou qu'une vérification de version est requise à chaque connexion sans impacter le fonctionnement. Dans ce cas, elle a été identifiée comme non bloquante.

- `Demande d'une clé... session_cookie_key` Cet avertissement concerne une clé secrète utilisée pour chiffrer les cookies de session de l'application web. Si cette clé n'est pas définie explicitement dans la configuration, NetDisco en génère une de manière aléatoire au démarrage, mais un avertissement peut apparaître. Pour un environnement de production, il est recommandé de générer une clé unique et de la spécifier dans le fichier de configuration (`deployment.yml`) pour assurer la persistance des sessions et renforcer la sécurité. Dans le cadre du stage, cet avertissement a été considéré comme non critique pour la fonctionnalité de découverte.

3. INTEGRATION A NETBOX

a. Importation fichiers CSV

L'objectif de cette étape est de transférer les informations brutes et vérifiées de l'infrastructure, collectées par NetDisco et stockées dans sa base de données PostgreSQL, vers Netbox. L'utilisation de fichiers CSV est privilégiée pour les importations massives ou lorsque l'on souhaite un contrôle granulaire sur les données avant leur injection.

Le processus se déroule en plusieurs étapes clés :

Accès à la base de données PostgreSQL de NetDisco :

La première action consiste à se connecter au shell de la base de données PostgreSQL qui est conteneurisée avec NetDisco.

```
sudo docker exec -it netdisco-postgresql psql -U netdisco
netdisco
```

> `sudo docker exec -it netdisco-postgresql` : Cette commande permet d'exécuter une commande à l'intérieur du conteneur Docker nommé `netdisco-postgresql` (qui héberge la base de données de NetDisco)
>
> `psql` : C'est le client en ligne de commande de PostgreSQL.
>
> `-U netdisco` : Spécifie l'utilisateur de la base de données à utiliser, ici `netdisco`.
>
> `netdisco` : Le nom de la base de données à laquelle se connecter, également `netdisco` par convention.

Une fois cette commande exécutée, l'utilisateur est connecté à la console `psql` et peut interagir directement avec la base de données NetDisco.

Extraction des données des équipements vers un fichier CSV :

Dans la console `psql`, la commande `\copy` est utilisée pour exporter les résultats d'une requête SQL directement vers un fichier sur le système de fichiers du conteneur. Cette commande est plus puissante et plus rapide que `COPY` pour les transferts vers des fichiers locaux.

Exemple 1 : Extraction des périphériques avec leur :

- nom
- fabricant
- serial
- model
- rôle Netbox
- status
- site

```
\copy (SELECT d.name AS name, d.serial AS serial, d.vendor AS manufacturer, d.model AS model, CASE WHEN d.layers = '00000100' THEN 'Router' WHEN d.layers = '00000010' THEN 'Switch' WHEN d.layers = '00001000' THEN 'AP' ELSE 'Other' END AS device_role, 'active' AS status, 'SiteInconnu' AS site FROM device d) TO '/tmp/netdisco_devices.csv' WITH (FORMAT CSV, HEADER);
```

`SELECT ... FROM device d` : Sélectionne des colonnes de la table `device` de NetDisco, qui contient les informations sur les équipements découverts.

`d.name AS name` : Le nom de l'équipement.

`d.serial AS serial` : Le numéro de série de l'équipement.

`d.vendor AS manufacturer, d.model AS model` : Le fabricant et le modèle de l'équipement.

`CASE WHEN d.layers = '00000100' THEN 'Router' WHEN d.layers = '00000010' THEN 'Switch' WHEN d.layers = '00001000' THEN 'AP' ELSE 'Other' END AS device_role` : Cette clause `CASE` est cruciale. Elle mappe la valeur binaire `d.layers` (qui représente les couches OSI de l'équipement dans NetDisco) à un `device_role` lisible pour Netbox. Par exemple, '00000010' (couche 2) est traduit en 'Switch', '00000100' (couche 3) en 'Router', et '00001000' (couche 4)

en 'AP'. Cette transformation est indispensable car Netbox utilise des rôles sémantiques.

`'active' AS status` : Assigne un statut 'active' par défaut à tous les équipements exportés. Ce statut est compatible avec Netbox.

`'SiteInconnu' AS site` : Assigne un site par défaut 'SiteInconnu'. Il est important de noter que ce site doit exister préalablement dans Netbox, ou les équipements devront être réassignés manuellement après l'importation.

`TO '/tmp/netdisco_devices.csv'` : Spécifie le chemin du fichier CSV où les données seront écrites *à l'intérieur du conteneur NetDisco*. Le répertoire `/tmp/` est souvent utilisé pour des fichiers temporaires.

`WITH (FORMAT CSV, HEADER)` : Ces options indiquent que le format de sortie doit être CSV et qu'une ligne d'en-tête (avec les noms de colonnes) doit être incluse.

Exemple 2 : Extraction des ip périphériques avec

- ip
- masque
- status
- dns
- interface

```
\copy (SELECT HOST(d_ip.ip)::text || '/' || MASKLEN(d_ip.subnet) AS address, 'active' AS status, d_ip.dns AS dns_name, d.dns AS device_name, d_ip.port AS interface_name, 'true' AS is_primary, 'false' AS is_oob FROM device_ip d_ip JOIN device d ON d_ip.ip = d.ip WHERE d_ip.ip IS NOT NULL AND d_ip.port IS NOT NULL) TO '/tmp/ip.csv' WITH (FORMAT CSV, HEADER);
```

Cette deuxième requête SQL a pour objectif d'extraire des informations détaillées sur les adresses IP et les interfaces réseau depuis la base de données de NetDisco, afin de les exporter dans un fichier CSV. Ce fichier peut ensuite être utilisé pour importer ces données dans un autre système, comme Netbox, pour enrichir son inventaire d'adresses IP.

Récupération du fichier CSV depuis le conteneur :

Une fois le fichier CSV généré à l'intérieur du conteneur (`/tmp/netdisco_devices.csv`), il doit être copié sur le système hôte pour être accessible par les scripts Python d'importation dans Netbox. La commande `docker cp` est utilisée à cet effet :

```
sudo docker cp netdisco-postgresql:/tmp/netdisco_devices.csv
```

Cette commande copie le fichier `netdisco_devices.csv` depuis le répertoire `/tmp/` du conteneur `netdisco-postgresql` vers le répertoire courant (`.`) sur le système hôte.

Le fichier est maintenant prêt à être intégré à Netbox grace a l'interface web proposant une option d'importation : soit fichier par fichier dans cet ordre précis :

1. Sites :
- `Regions.csv`
- `Sites.csv`
- `Locations.csv`
2. Fabricants :
- `Manufacturers.csv`
3. Types d'appareils et de modules :
- `DeviceTypes.yaml`
- `ModuleTypes.yaml`
4. Racks :
- `Racks.csv`
5. Appareils :
- `Devices.csv`
6. Composants d'appareils :
- `Interfaces.csv`
- `ConsolePorts.csv`
- `PowerPorts.csv`
- `FrontPorts.csv`
- `RearPorts.csv`
- `DeviceBays.csv`
- `InventoryItems.csv`
7. Modules :
- `Modules.csv`
8. Adresses IP :

- `IPAddresses.csv`
- `Prefixes.csv`
- `VLANs.csv`

9. Câblage :

- `Câbles.csv`

b. Automatisation avec scripts pour l'intégration à Netbox

Le fichier CSV ainsi obtenu devient la source de données pour les scripts Python précédemment développés, qui sont chargés d'interagir avec l'API RESTful de Netbox. Ce processus combine la puissance de découverte de NetDisco avec la capacité de gestion de Netbox et la flexibilité de l'automatisation Python.

Traitement du fichier CSV par les scripts Python : Le script `yaml_processor.py` (ou un script similaire adapté au format CSV) lira le fichier `netdisco_devices.csv`. Pour chaque ligne du CSV (représentant un équipement), le script extraira les informations : `name`, `primary_ip4`, `serial`, `manufacturer`, `model`, `device_role` (ou `role` et `device_type`), `status`, et `site`.

Mapping des données et interaction avec l'API Netbox :

- Récupération des IDs d'objets Netbox :

  Avant de créer ou mettre à jour un équipement, les scripts utiliseront les fonctions de `netbox_api.py` pour récupérer les IDs numériques des objets Netbox correspondants :

  `get_device_type_id()` : Pour le modèle de l'équipement (ex: "WS-C2960-24TT-L").

  `get_device_role_id()` : Pour le rôle de l'équipement (ex: "Switch", "Router", "AP").

  `get_site_id()` : Pour le site géographique (ex: "SiteInconnu"). Ces étapes sont cruciales car l'API Netbox attend des IDs numériques pour relier les objets.

- Création ou mise à jour des équipements (`device_manager.py`) :

  Pour chaque ligne du CSV, le script vérifiera si un équipement avec le même nom (ou numéro de série) existe déjà dans Netbox en utilisant `device_exists()`.

  Si l'équipement n'existe pas, une requête `POST` sera envoyée à l'API `/api/dcim/devices/` pour créer un nouvel équipement avec toutes les propriétés extraites du CSV (nom, rôle, type, site, statut, numéro de série).

  Si l'équipement existe déjà, une requête `PATCH` sera utilisée pour mettre à jour ses propriétés, garantissant ainsi que l'inventaire Netbox est synchronisé avec les dernières découvertes de NetDisco sans créer de doublons.

- Gestion des interfaces et des adresses IP/MAC :

  Bien que l'extraction CSV ci-dessus se concentre sur les informations de base des équipements, une extension des scripts pourrait également traiter les interfaces et les adresses IP/MAC. Les requêtes `\copy` pourraient être adaptées pour exporter les tables d'interfaces et d'adresses de NetDisco, puis les scripts Python seraient enrichis pour créer ou mettre à jour les objets `Interface` et `IPAddress` dans Netbox, en les liant aux `Device` correspondants. Les fonctions comme `create_interfaces()`, `assign_mac_to_interface()`, et `assign_ip_to_device()` de `device_manager.py` seraient appelées pour ces opérations.

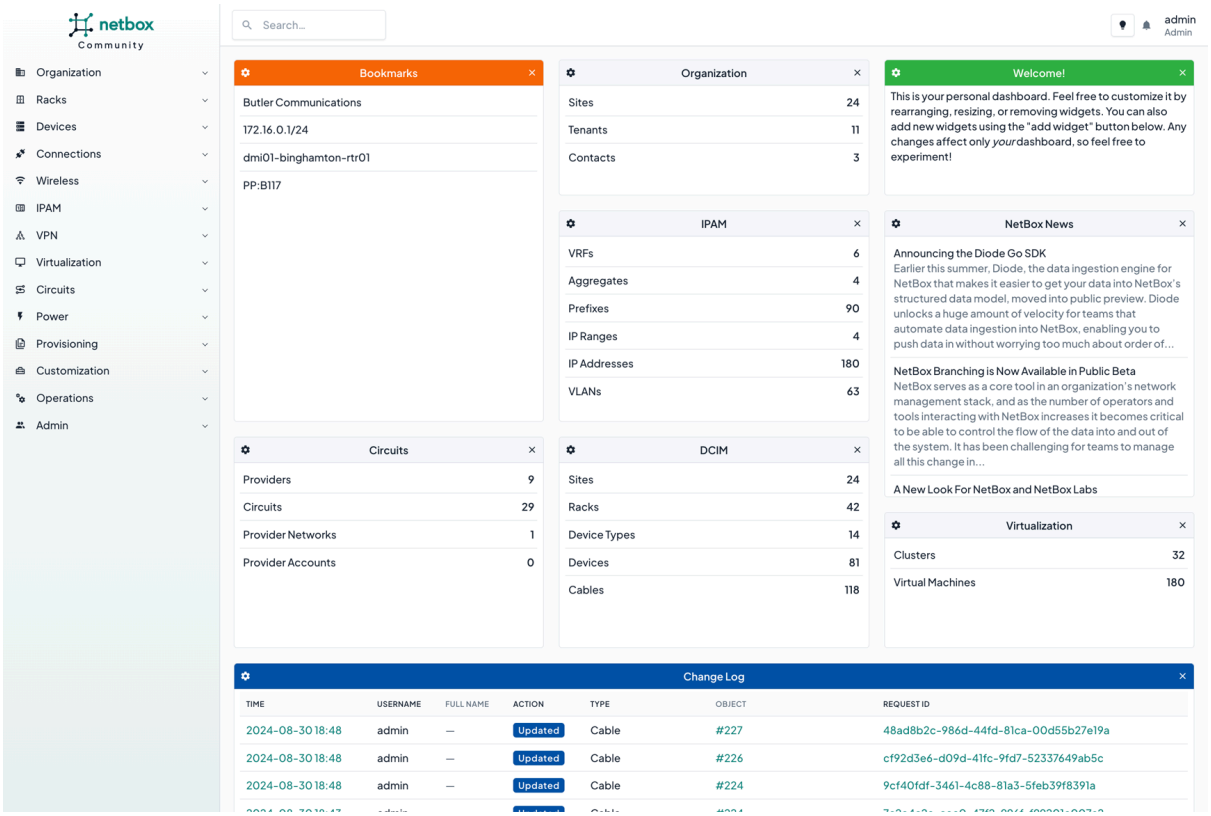
Exemple de flux de données et d'automatisation :

1. Découverte NetDisco : NetDisco scanne le réseau de la DSI et stocke les informations brutes dans sa BDD PostgreSQL.
2. Extraction CSV : Les commandes `\copy` sont exécutées pour extraire les données pertinentes et formatées dans un fichier CSV.
3. Transfert du CSV : Le fichier CSV est copié du conteneur NetDisco vers l'environnement où les scripts Python s'exécutent (potentiellement le même hôte que Netbox, ou un autre serveur de gestion).
4. Importation par script : Le script Python lit le CSV, traite les données ligne par ligne, effectue les mappings nécessaires et utilise l'API Netbox pour créer ou mettre à jour les équipements.

Avantages de cette approche :

- Contrôle précis : Permet de filtrer, transformer et valider les données de NetDisco avant leur importation dans Netbox, assurant une meilleure qualité d'inventaire.
- Importation rapide : Pour de grands volumes d'équipements, l'importation par CSV est beaucoup plus rapide que l'ajout manuel.
- Flexibilité : Les requêtes SQL et les scripts Python peuvent être adaptés pour extraire et formater n'importe quelle donnée présente dans NetDisco, et l'injecter dans n'importe quel champ de Netbox.
- Auditabilité : Le processus est scripté et reproductible, ce qui facilite le débogage et l'audit des données importées.
- Bridge entre outils : Cette méthode crée un lien efficace entre un outil de découverte dynamique (NetDisco) et un système de "Source of Truth" structuré (Netbox).

Cette synergie entre NetDisco et Netbox, orchestrée par des scripts Python et des extractions SQL, a permis de transformer un inventaire manquant en un système d'information réseau centralisé, fiable et à jour pour la DSI.



Intégration des données de NetDisco dans NetBox complété

Conclusion

Ce stage a représenté une opportunité significative de contribuer à la modernisation de l'infrastructure réseau de la DSI, en répondant à un besoin crucial : l'établissement d'une **source de vérité fiable et en temps réel** pour la gestion de ses équipements. L'absence d'outils d'inventaire et de découverte, exacerbée par la migration vers une infrastructure plus robuste, a souligné l'importance capitale d'une cartographie précise et dynamique du réseau.

La mise en œuvre de **Netbox** et **NetDisco** a été au cœur de cette démarche. Le déploiement et la configuration de Netbox ont permis de structurer les données d'infrastructure matérielle et virtuelle, transformant ainsi un inventaire fragmenté en une base de données centralisée et interrogeable. Parallèlement, l'intégration de NetDisco a automatisé la découverte des équipements, offrant une visibilité continue sur les évolutions du réseau et garantissant la pertinence des informations enregistrées dans Netbox. L'automatisation via des scripts a également démontré le potentiel d'optimisation des tâches d'intégration, libérant du temps pour des activités à plus forte valeur ajoutée.

Au-delà des aspects techniques, ce stage m'a permis de développer une compréhension approfondie des enjeux liés à la gestion d'une infrastructure réseau complexe. J'ai acquis une expérience précieuse dans la planification, le déploiement et la configuration d'outils de supervision, ainsi que dans la résolution des défis inhérents à ces projets. L'autonomie et l'adaptabilité ont été des compétences clés renforcées tout au long de cette période.

En définitive, les solutions mises en place constituent une avancée majeure pour la DSI. Elles fournissent désormais les bases d'une **gestion proactive et éclairée de l'infrastructure réseau**, permettant une meilleure compréhension des interdépendances, une détection plus rapide des anomalies et une prise de décision plus pertinente. Ces outils sont un pilier essentiel pour accompagner la DSI dans ses futures évolutions et maintenir la performance d'un réseau en constante mutation.

Bibliographie

Présentation de l'entreprise

Officiers et anciens élèves - Albert Jean René DENIÉLOU. (s.d.).
Récupéré sur Ecole Nav Traditions:
http://ecole.nav.traditions.free.fr/officiers_denielou_albert.htm

Officiers et anciens élèves - Guy DENIÉLOU. (s.d.).
Récupéré sur Ecole Nav Traditions:
http://ecole.nav.traditions.free.fr/officiers_denielou_guy.htm

NetBox

NetBox Community. (n.d.). *Installation Guide*. NetBox Documentation.
https://docs.netbox.dev/en/stable/installation/

Netbox dépôt d'installation officiel
https://github.com/netbox-community

Docker

Docker. (n.d.). *Install Docker Engine on Ubuntu*. Docker Documentation.
https://docs.docker.com/engine/install/ubuntu/

Docker dépôt d'installation officiel
https://download.docker.com

NetDisco

NetDisco. (n.d.). *NetDisco Deployment*. NetDisco Documentation.
https://www.netdisco.org/doc/Deployment.html

NetDisco documentation
https://blog.pascal-mietlicki.fr/administration-reseau-simplifiez-la-gestion-de-votre-infrastructure-avec-netdisco/

NetDisco documentation
https://metacpan.org/pod/App::Netdisco

NetDisco dépôt d'installation officiel
https://github.com/netdisco/netdisco

Glossaire

DSI : Direction des Systèmes d'Information

PDF : Portable Document Format, il s'agit d'une norme ISO décrivant la structure d'un document. Il se veut portable sur différents systèmes sans modifier la structure d'un document.

CEA : Commissariat à l'énergie atomique et aux énergies alternatives, il s'agit d'une institution scientifique majeure travaillant sur les énergies bas carbones, le numérique, la médecine et la défense. UTC : Université de Technologie de Compiègne

ENT : Environnement Numérique de Travail

SGBD : Système de Gestion de Base de Données, il s'agit du logiciel qui permet de gérer une base de données.

SQL : Structured Query Language, il s'agit d'un langage permettant de communiquer avec une base de données.

Annexes

Script Python : intégration Netbox en utilisant l'API

main.py : Donne le chemin d'accès au fichier yaml à utiliser par yaml_procesor.py

```
import os
from yaml_processor import load_and_process_yaml  #
Importation correcte de ton module

def main():
     # Chemin relatif du fichier YAML dans le sous-dossier
network_devices
     yaml_path = "network_devices/access-points/ap-rob.yaml"
     print("🚀 Démarrage de l'importation des équipements vers
NetBox...")
     load_and_process_yaml(yaml_path)  # Appel de la fonction
de traitement
     print("✅ Importation terminée.")

if __name__ == "__main__":
     main()
```

device_manager.py : Gère la création, la mise à jour des équipements, et la création des interfaces sur NetBox.

```
import requests
from netbox_config import DEBUG_MODE
from icecream import ic

if not DEBUG_MODE:
     ic.disable()

from netbox_config import NETBOX_URL, HEADERS
from utils import format_slug
```

```
# Fonction pour récupérer l'ID d'une interface par son nom
def get_interface_id_by_name(device_id, interface_name):
     """
     Récupère l'ID de l'interface par son nom pour un
périphérique donné.
     """
     response =
requests.get(f"{NETBOX_URL}dcim/interfaces/?device_id={device_
id}&name={interface_name}", headers=HEADERS)
     if response.status_code == 200 and
response.json()["count"] > 0:
     return response.json()["results"][0]["id"]
     return None

def device_exists(device_name):
     response =
requests.get(f"{NETBOX_URL}dcim/devices/?name={device_name}",
headers=HEADERS)
     if response.status_code == 200 and
response.json()["count"] > 0:
     return response.json()["results"][0]
     return None

def create_or_update_device(payload, existing_device=None):
     if existing_device:
     device_id = existing_device["id"]
     response =
requests.patch(f"{NETBOX_URL}dcim/devices/{device_id}/",
headers=HEADERS, json=payload)
     if response.status_code in (200, 204):
          print(f"✅ Équipement mis à jour :
{payload['name']}")
          return device_id
     else:
          print(f"❌ Échec mise à jour {payload['name']} :
{response.status_code}")
          return None
     else:
```

```
     response = requests.post(f"{NETBOX_URL}dcim/devices/",
headers=HEADERS, json=payload)
     if response.status_code == 201:
          print(f"✅ Équipement créé : {payload['name']}")
          return response.json()["id"]
     else:
          print(f"❌ Échec création {payload['name']} :
{response.status_code}")
          return None

def get_device_type_id(device_type_name):
     if not device_type_name:
     return None
     slug = format_slug(device_type_name)
     response =
requests.get(f"{NETBOX_URL}dcim/device-types/?slug={slug}",
headers=HEADERS)
     if response.status_code == 200 and
response.json()["count"] > 0:
     return response.json()["results"][0]["id"]

def get_site_id(site_name):
     if not site_name:
     return None
     response =
requests.get(f"{NETBOX_URL}dcim/sites/?name={site_name}",
headers=HEADERS)
     if response.status_code == 200 and
response.json()["count"] > 0:
     return response.json()["results"][0]["id"]

def get_interface_id_by_name(device_id, interface_name):
     response =
requests.get(f"{NETBOX_URL}dcim/interfaces/?device_id={device_
id}&name={interface_name}", headers=HEADERS)
     if response.status_code == 200 and
response.json()["count"] > 0:
     return response.json()["results"][0]["id"]
```

```
def create_interfaces(device_id, port_list, mgmt_only):
     interface_ids = {}

     for port_name in port_list:
     # Vérifie si l'interface existe déjà
     response = requests.get(
          f"{NETBOX_URL}dcim/interfaces/",
          headers=HEADERS,
          params={"device_id": device_id, "name": port_name}
     )
     results = response.json().get("results", [])
     if results:
          interface_id = results[0]["id"]
          ic(f"🔁 Interface existante récupérée : {port_name}
-> {interface_id}")
     else:
          # Crée l'interface
          payload = {
               "device": device_id,
               "name": port_name,
               "type": "1000base-t",  # ou autre si nécessaire
               "mgmt_only": mgmt_only or False,
          }
          response =
requests.post(f"{NETBOX_URL}dcim/interfaces/",
headers=HEADERS, json=payload)
          response.raise_for_status()
          interface_id = response.json()["id"]
          ic(f"🆕 Interface créée : {port_name} ->
{interface_id}")

     interface_ids[port_name] = interface_id

     return interface_ids

def assign_mac_to_interface(interface_id, mac_address):
     import requests
```

```
    url = f"{NETBOX_URL}/dcim/mac-addresses/"

    # Vérifier si la MAC existe déjà
    existing = requests.get(url, headers=HEADERS,
params={"mac_address": mac_address}).json()

    if existing["count"] > 0:
    # Si elle existe, mettre à jour avec l'interface si non
assignée
    mac_entry = existing["results"][0]
    if mac_entry.get("interface") is None:
        update_url = f"{url}{mac_entry['id']}/"
        response = requests.patch(update_url,
headers=HEADERS, json={"interface": interface_id})
        if response.status_code in [200, 204]:
            print(f"✅ MAC {mac_address} assignée à
l'interface {interface_id}")
        else:
            print(f"❌ Erreur lors de l'assignation de la
MAC à l'interface : {response.text}")
    else:
        print(f"⚠️ Adresse MAC {mac_address} déjà associée à
une interface.")
    else:
    # Créer la MAC avec l'interface directement
    payload = {
        "mac_address": mac_address,
        "interface": interface_id,
    }
    response = requests.post(url, headers=HEADERS,
json=payload)
    if response.status_code in [200, 201]:
        print(f"✅ MAC {mac_address} créée et assignée à
l'interface {interface_id}")
    else:
        print(f"❌ Erreur lors de la création de la MAC :
{response.text}")
```

```
def format_mac_address(mac_address):
     mac = mac_address.replace(".", "").replace("-",
"").replace(":", "").strip().upper()
     if len(mac) != 12:
     print(f"❌ Adresse MAC invalide : {mac_address}")
     return None
     return ":".join([mac[i:i+2] for i in range(0, 12, 2)])

def assign_ip_to_device(device_id, ip_address):
     response =
requests.get(f"{NETBOX_URL}ipam/ip-addresses/?address={ip_addr
ess}", headers=HEADERS)
     if response.status_code == 200 and
response.json()["count"] > 0:
     ip_id = response.json()["results"][0]["id"]
     else:
     payload = {"address": ip_address}
     response =
requests.post(f"{NETBOX_URL}ipam/ip-addresses/",
headers=HEADERS, json=payload)
     if response.status_code == 201:
          ip_id = response.json()["id"]
     else:
          print(f"❌ Erreur ajout IP {ip_address}:
{response.status_code}")
          return None

     interface_response =
requests.get(f"{NETBOX_URL}dcim/interfaces/?device_id={device_
id}", headers=HEADERS)
     if interface_response.status_code == 200 and
interface_response.json()["count"] > 0:
     interface_id =
interface_response.json()["results"][0]["id"]
     else:
```

```
        print(f"❌ Pas d'interface trouvée pour device
{device_id}")
        return None

        payload = {"assigned_object_type": "dcim.interface",
"assigned_object_id": interface_id}
        assign_response =
requests.patch(f"{NETBOX_URL}ipam/ip-addresses/{ip_id}/",
headers=HEADERS, json=payload)

        if assign_response.status_code in (200, 204):
        print(f"✅ Adresse IP {ip_address} assignée à l'interface
{interface_id}")
        return interface_id
        else:
        print(f"❌ Erreur assignation IP {ip_address}:
{assign_response.status_code}")
        return None
```

netbox_api.py : Contient des fonctions utilitaires pour interagir avec l'API de NetBox, telles que la récupération de l'ID du type de périphérique, du rôle, du site, etc.

```
import requests
import json
from netbox_config import DEBUG_MODE
from icecream import ic

if not DEBUG_MODE:
     ic.disable()

from netbox_config import NETBOX_URL, HEADERS


# Fonction pour récupérer l'ID d'un type de périphérique par
son nom (ex: "cisco-switch")
def get_device_type_id(device_type_name):
     response = requests.get(

f"{NETBOX_URL}dcim/device-types/?model={device_type_name}",
     headers=HEADERS
     )
     if response.status_code == 200:
     device_types = response.json()['results']
     ic(device_types)
     if device_types:
          return device_types[0]['id']
     print(f"❌ Erreur: type de périphérique introuvable :
{device_type_name}")
     return None

# Vérifie si un périphérique existe déjà par son nom
def device_exists(device_name):
     response = requests.get(
     f"{NETBOX_URL}dcim/devices/?name={device_name}",
     headers=HEADERS
     )
```

```
    if response.status_code == 200:
    devices = response.json()['results']
    if devices:
        return devices[0]
    return None

# Récupère l'ID du rôle "Switch"
def get_device_role():
    role_name = "Wi-Fi AP"
    response = requests.get(
    f"{NETBOX_URL}dcim/device-roles/?name={role_name}",
    headers=HEADERS
    )
    if response.status_code == 200:
    roles = response.json()['results']
    if roles:
        return roles[0]['id']
    print(f"❌ Erreur: rôle '{role_name}' introuvable dans
NetBox.")
    return None

# Récupère l'ID du site à partir de son nom (description dans
YAML)
def get_site_id(site_name):
    if not site_name:
    return None
    response = requests.get(
    f"{NETBOX_URL}dcim/sites/?name={site_name}",
    headers=HEADERS
    )
    if response.status_code == 200 and
response.json()["count"] > 0:
    return response.json()["results"][0]["id"]
    else:
    print(f"🏷️ Site '{site_name}' non trouvé dans NetBox.")
    return None

# Crée ou met à jour un périphérique avec le rôle bon role
```

```
def create_or_update_device(device_payload,
existing_device=None):
     device_role_id = get_device_role()
     if not device_role_id:
     print("❌ Rôle 'VPN' introuvable, arrêt.")
     return None

     # On ajoute bien le rôle ici, AVANT de sortir de la
fonction
     device_payload['role'] = device_role_id

     if existing_device:
     device_id = existing_device["id"]
     response = requests.patch(
          f"{NETBOX_URL}dcim/devices/{device_id}/",
          headers=HEADERS,
          data=json.dumps(device_payload)
     )
     if response.status_code == 200:
          print(f"✅ Équipement mis à jour :
{device_payload['name']}")
          return device_id
     else:
          print(f"❌ Erreur MAJ {device_payload['name']}:
{response.status_code} - {response.text}")
          return None
     else:
     response = requests.post(
          f"{NETBOX_URL}dcim/devices/",
          headers=HEADERS,
          data=json.dumps(device_payload)
     )
     if response.status_code == 201:
          print(f"✅ Équipement créé :
{device_payload['name']}")
          return response.json()['id']
     else:
```

```
            print(f"❌ Erreur création {device_payload['name']}:
{response.status_code} - {response.text}")
            return None


# Récupère l'ID d'un rôle de périphérique par son nom
(générique)
def get_device_role_id(role_name):
     response = requests.get(
     f"{NETBOX_URL}dcim/device-roles/?name={role_name}",
     headers=HEADERS
     )
     if response.status_code == 200:
     roles = response.json().get("results", [])
     if roles:
          return roles[0]["id"]
     print(f"❌ Rôle '{role_name}' introuvable dans NetBox.")
     return None
```

netbox_config.py : Autorise la connexion à l'API netox, grace à son IP et à son Token

```
NETBOX_URL = "http://192.168.100.160:8000/api/"
NETBOX_TOKEN = "04946ef59ffeb57bc7a0e7c6ac73f787eb272c57"

HEADERS = {
     "Authorization": f"Token {NETBOX_TOKEN}",
     "Content-Type": "application/json",
}
DEBUG_MODE = True
```

yaml_processor.py : Le module principal pour charger le fichier YAML, créer ou mettre à jour les équipements dans NetBox, et gérer les interfaces, les IPs, et autres paramètres.

```
import yaml
from netbox_config import DEBUG_MODE
from icecream import ic

if not DEBUG_MODE:
     ic.disable()

from netbox_api import (
     get_device_type_id,
     get_device_role,
     device_exists,
     create_or_update_device,
     get_site_id,
)
from device_manager import create_interfaces,
assign_ip_to_device, assign_mac_to_interface
from utils import format_mac

def load_and_process_yaml(file_path):
     ic(file_path)
     with open(file_path, 'r') as file:
```

```
    data = yaml.safe_load(file)

    for device_name, device_info in data.items():
    ic(device_name, device_info)

    device_type_name = device_info.get("type")
    ip_address = device_info.get("ip_address")
    mac_address = device_info.get("mac_address")
    role_name = device_info.get("role")
    raw_ports = device_info.get("ports")

    if mac_address:
        mac_address = format_mac(mac_address)

    device_type_id = get_device_type_id(device_type_name)

    device_role_id = None
    if role_name:
        device_role_id = get_device_role(role_name)

    existing_device = device_exists(device_name)

    device_payload = {
        "name": device_name,
        "device_type": device_type_id,
        "site": get_site_id("Roberval"),
    }


    if device_role_id:
        device_payload["device_role"] = device_role_id

    if "description" in device_info:
        device_payload["description"] =
device_info["description"]

    device_id = create_or_update_device(device_payload,
existing_device)
```

```
     port_list = []
     if raw_ports:
          for part in raw_ports.split(","):
               part = part.strip()
               if "-" in part:
                    base = part[:part.index("/") + 1]
                    range_part = part.split("/")[1]
                    start, end = map(int,
range_part.split("-"))
                    for i in range(start, end + 1):
                         port_list.append(f"{base}{i}")
               else:
                    port_list.append(part)

     # Crée les interfaces ET récupère les IDs
     interface_map = {}
     if port_list:
          interface_map = create_interfaces(device_id,
port_list, None)

     # Assigne l'IP
     if ip_address:
          interface_id = assign_ip_to_device(device_id,
ip_address)

          # Utilise l'interface IP aussi pour la MAC
(optionnel)
          if interface_id and mac_address:
               assign_mac_to_interface(interface_id,
mac_address)

     # Sinon, assigne la MAC à G0/1 par défaut si dispo
     elif mac_address and "G0/1" in interface_map:
          assign_mac_to_interface(interface_map["G0/1"],
mac_address)
```

Fichier de Configuration Netdisco

docker-compose.yml :

```
services:
  netdisco-postgresql:
     image: netdisco/netdisco:latest-postgresql
     container_name: netdisco-postgresql
     hostname: netdisco-postgresql
     volumes:
     - pgdata:/var/lib/postgresql/data
     networks:
     - netdisco-net

  netdisco-backend:
     image: netdisco/netdisco:latest-backend
     container_name: netdisco-backend
     hostname: netdisco-backend
     init: true
     volumes:
     - ./nd-site-local:/home/netdisco/nd-site-local
     - ./config:/home/netdisco/environments
     - ./logs:/home/netdisco/logs
     environment:
     NETDISCO_DOMAIN: localdomain
     NETDISCO_DB_HOST: netdisco-postgresql
     depends_on:
     - netdisco-postgresql
     networks:
     - netdisco-net

  netdisco-web:
     image: netdisco/netdisco:latest-web
     container_name: netdisco-web
     hostname: netdisco-web
     init: true
     ports:
     - "5000:5000"
```

```
    volumes:
    - ./nd-site-local:/home/netdisco/nd-site-local
    - ./config:/home/netdisco/environments
    environment:
    NETDISCO_DOMAIN: localdomain
    NETDISCO_DB_HOST: netdisco-postgresql
    HYPNOTOAD_LISTEN: http://0.0.0.0:5000

    depends_on:
    - netdisco-postgresql
    networks:
    - netdisco-net

  netdisco-do:
    image: netdisco/netdisco:latest-do
    container_name: netdisco-do
    hostname: netdisco-do
    volumes:
    - ./nd-site-local:/home/netdisco/nd-site-local
    - ./config:/home/netdisco/environments
    environment:
    NETDISCO_DOMAIN: localdomain
    NETDISCO_DB_HOST: netdisco-postgresql
    HYPNOTOAD_LISTEN: http://0.0.0.0:5000

    depends_on:
    - netdisco-postgresql
    profiles:
    - cli-manual
    networks:
    - netdisco-net

volumes:
  pgdata:

networks:
  netdisco-net:
    driver: bridge
```

```
    ipam:
    config:
    - subnet: 10.10.0.0/16
```

deployment.yml :

```
database:
  name: 'netdisco'
  user: 'netdisco'
  pass: 'netdisco'
  #host: 'netdisco-postgresql'
  port: 5432

domain_suffix: 'localdomain'
site_name: 'Réseau SNMP'

# Clé de session requise pour Netdisco Web
#session_cookie_key:
'd43b52e4f1d44f39b681db9494a7d2cf0fc2e2a9a79be30fc2d4039e037f4
7b2'

device_auth:
  - tag: 'snmpv3'
     user: stagiaire
     auth:
     pass: nJS9cq5TFwWnQs
     proto: SHA
     priv:
     pass: DHGC5uxBGBpn4d
     proto: AES


# Pas d'authentification (utilisateur guest en admin)
no_auth: true

# Détection automatique
discover_no:
  - '127.0.0.1'
```