Rapport installation et déploiement :

- NetBox
- NetDisco
- Prometheus
- VioctoriaMetrics

1.Installation et configuration de NetBox

1.1.Installation de Docker et Docker Compose

1.1.1.MAJ des paquets

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

1.1.2.Ajouter la clé GPG Docker

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

1.1.3.Ajouter le dépôt Docker officiel

```
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
```

1.1.4.Installer Docker et le plugin Compose (v2)

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

1.1.5.Démarrer et activer Docker

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

1.2.Récupération et configuration des fichiers NetBox

Nous allons récupérer les fichiers de NetBox à partir du dépôt officiel GitHub et entrer dans le répertoire de travail :

1.2.1.Cloner le dépôt

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

1.2.2.(Optionnel) Choisir une version précise

```
echo "VERSION=v4.2.3" > .env
```

1.2.3.Lancer les conteneurs avec Compose v2

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

1.2.4.Vérification et accès aux logs du conteneur NetBox

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

1.2.5.Relancer docker

```
sudo docker compose down
sudo docker compose up -d # dans le répertoire netbox-docker
```

Le port 8000 n'est pas exposé par défaut :

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

```
  GNU nano 7.2                    docker-compose.override.yml *
services:
  netbox:
    ports:
      - "8000:8080"
```

1.2.6.Accès à sur l'IP du serveur : http://192.168.100.160:8000

1.2.7.Création d'un superuser utilisateur : admin

Accès au conteneur NetBox :

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

Nom d'utilisateur : **admin**
Adresse email : (champs ignoré)
Mot de passe : **Progtr00**

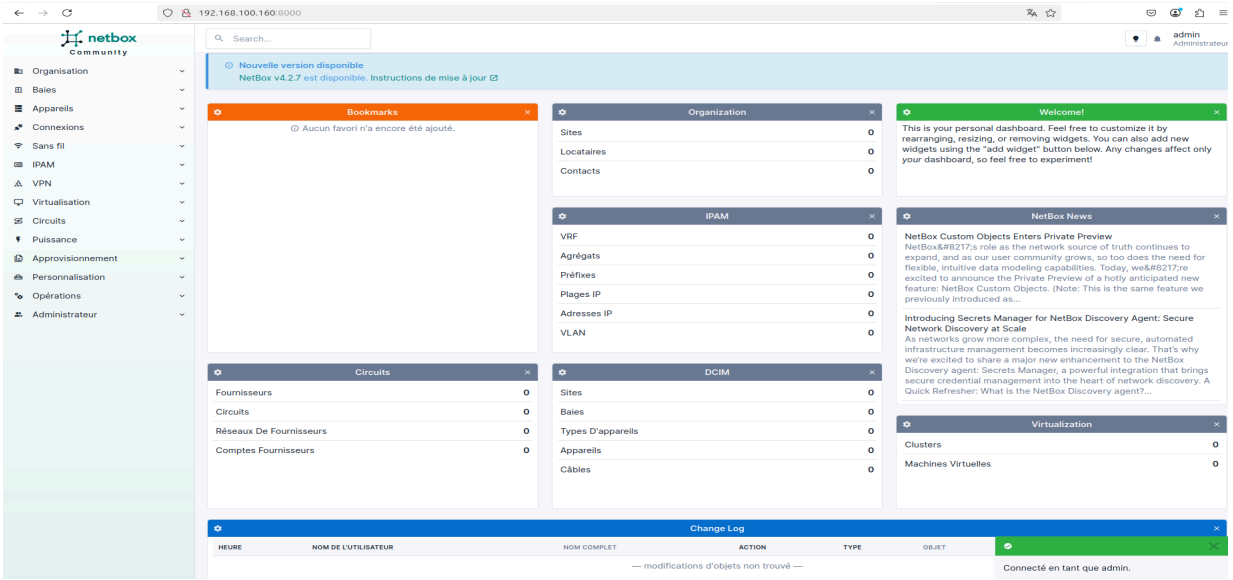
On relance docker :

```
sudo docker compose restart
```

1.2.8.(Sécurisation)Réinitialiser le password de l'admin

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

1.2.9.Accès à NetBox en superuser (admin)



1.2.10.Générer un token d'API NetBox

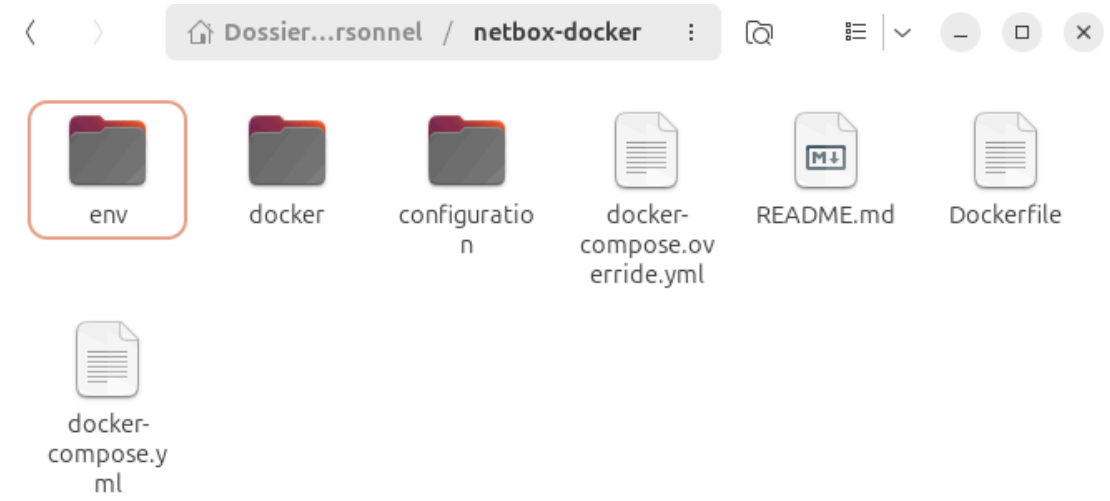
clé_token : 04946ef59ffeb57bc7a0e7c6ac73f787eb272c57

Arborescence : fichiers provenant du dépôt officiel

```
pfamchon@dsiport62:~/netbox-docker$ tree -a
.
├── configuration
│   ├── configuration.py
│   ├── extra.py
│   ├── ldap
│   │   ├── extra.py
│   │   └── ldap_config.py
│   ├── logging.py
│   └── plugins.py
├── docker
│   ├── configuration.docker.py
│   ├── docker-entrypoint.sh
│   ├── housekeeping.sh
│   ├── launch-netbox.sh
│   ├── ldap_config.docker.py
│   └── nginx-unit.json
├── docker-compose.override.yml
├── docker-compose.yml
├── Dockerfile
├── .dockerignore
├── .ecrc
├── .editorconfig
├── env
│   ├── netbox.env
│   ├── postgres.env
│   ├── redis-cache.env
│   └── redis.env
├── .flake8
├── .git
│   ├── branches
│   ├── config
│   ├── description
│   ├── HEAD
│   ├── hooks
│   │   ├── applypatch-msg.sample
│   │   ├── commit-msg.sample
```

```
|   |   ├── fsmonitor-watchman.sample
|   |   ├── post-update.sample
|   |   ├── pre-applypatch.sample
|   |   ├── pre-commit.sample
|   |   ├── pre-merge-commit.sample
|   |   ├── prepare-commit-msg.sample
|   |   ├── pre-push.sample
|   |   ├── pre-rebase.sample
|   |   ├── pre-receive.sample
|   |   ├── push-to-checkout.sample
|   |   ├── sendemail-validate.sample
|   |   └── update.sample
|   ├── index
|   ├── info
|   |   └── exclude
|   ├── logs
|   |   ├── HEAD
|   |   └── refs
|   |     ├── heads
|   |     |   └── release
|   |     └── remotes
|   |          └── origin
|   |               └── HEAD
|   ├── objects
|   |   ├── info
|   |   └── pack
|   |     ├──
pack-5f7651b82a78193a315bec64d27c041d530f5478.idx
|   |     ├──
pack-5f7651b82a78193a315bec64d27c041d530f5478.pack
|   |     └──
pack-5f7651b82a78193a315bec64d27c041d530f5478.rev
|   ├── packed-refs
|   └── refs
|    ├── heads
|    |   └── release
|    ├── remotes
|    |   └── origin
```

```
|     |     └── HEAD
|     └── tags
├── .github
|    ├── FUNDING.yml
|    ├── ISSUE_TEMPLATE
|    |    ├── bug_report.yml
|    |    ├── config.yml
|    |    └── feature_request.yml
|    ├── no-response.yml
|    ├── pull_request_template.md
|    └── workflows
|     ├── push.yml
|     └── release.yml
├── .gitignore
├── .hadolint.yaml
├── .markdown-lint.yml
├── README.md
└── .yamllint.yaml
```



1.3.Importation du fichier yaml : network_devices.yaml

1.3.1.Création des répertoire contenant les différents scripts et fichiers yaml

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

(Répertoire au même niveau que le répertoire netbox-docker)

1.3.2.Découpage du fichier yaml en sous fichiers

```
mkdir network_devices
```

Dans ce répertoire j'ai séparé tous les types d'équipement pour simplifier l'exécution du script sans erreur

```
├── network_devices
│   ├── access-points
│   │   ├── ap-bf.yaml
│   │   ├── ap-cima.yaml
│   │   ├── ap-ci.yaml
│   │   ├── ap-cr.yaml
│   │   ├── ap-ct.yaml
│   │   ├── ap-escom.yaml
│   │   ├── ap-gi.yaml
│   │   ├── ap-imi.yaml
│   │   ├── ap-manif.yaml
│   │   ├── ap-mgaudry.yaml
│   │   ├── ap-pg.yaml
│   │   ├── ap-pom.yaml
│   │   ├── ap-ree.yaml
│   │   ├── ap-rob.yaml
│   │   ├── ap-ruche.yaml
│   │   └── ap-si.yaml
│   ├── captive-portals
│   │   └── captive_portals.yaml
```

```
|    ├── network_devices.yaml
|    ├── radius
|    |    └── radius.yaml
|    ├── switches
|    |    ├── sw-5000.yaml
|    |    ├── sw-bf.yaml
|    |    ├── sw-cima.yaml
|    |    ├── sw-ci.yaml
|    |    ├── sw-cr.yaml
|    |    ├── sw-ct.yaml
|    |    ├── sw-escom.yaml
|    |    ├── sw-gi.yaml
|    |    ├── sw-hds.yaml
|    |    ├── sw-imi.yaml
|    |    ├── sw-manif.yaml
|    |    ├── sw-mgaudry.yaml
|    |    ├── sw-parc.yaml
|    |    ├── sw-peupliers.yaml
|    |    ├── sw-pg.yaml
|    |    ├── sw-polea.yaml
|    |    ├── sw-pom.yaml
|    |    ├── sw-ree.yaml
|    |    ├── sw-rob.yaml
|    |    ├── sw-ruche.yaml
|    |    └── sw-si.yaml
|    ├── vpn-servers
|    |    └── vpn_servers.yaml
|    └── wlc
|     └── wlc.yaml
```

1.3.4.Scripts python interagissant avec l'API netbox

Import_yaml/ regroupe tous les scripts liés à l'importation de fichier yaml d'un format spécifique dans Netox en communiquant avec l'API netbox.

Ce répertoire contient :

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

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
```

```python
# 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)
```

2.Installation et configuration de NetDisco

2.1.Récupération des fichiers NetBox

2.1.1.Installation des paquets

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

2.1.2.Création des répertoires

```
mkdir -p netdisco/{logs,config,nd-site-local}
```

2.1.3.Autorisation de l'utilisateur Netdisco à y écrire

```
sudo chown -R 901:901 netdisco
```

Les conteneurs ont besoin que certains répertoires soient présents dans le volume monté.
Dans un répertoire de votre choix, créez cette structure et autorisez l'identifiant utilisateur Netdisco dans le conteneur (901) à y écrire.

2.2.Installation et de Docker Compose

2.2.1.Image par défaut fournir par Netdisco

```
curl -Ls -o docker-compose.yml
https://tinyurl.com/nd2-dockercompose
```

2.2.2.Lancer la construction des conteneurs

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

2.3.Configuration des fichiers Docker Compose et Netdisco

2.3.1.Remplacer le contenue de l'image docker-compose.yml

```
sudo nano docker-compose.yml
```

Copier coller la configuration suivante :

```
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
```

2.3.2.Remplacer le contenue du fichier deployment.yml

Dans le répertoire netdisco/netdisco/config (répertoire créé plus haut suite à l'installation des paquets officiels) :

```
cd config

sudo nano deployment.yml

# 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:
'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'
```

2.3.3.Relancer les conteneurs

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

2.3.4.Relancer le conteneur netdisco-web

Attention : obligation de relancer ce conteneur séparément pour éviter des erreurs de communication entre le conteneur `netdisco-web` et le conteneur de la base de donnée : `netdisco-postgresql`

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

2.3.5.Mettre à jour le schéma de la BDD

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

Ensuite redémarrer le conteneur netdisco-web et netdisco-backend

2.4.Test et vérification du bon fonctionnement de Netdisco

2.4.1.Vérification de la résolution DNS interne entre conteneurs

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

2.4.2.Test de connectivité applicative à la BDD PostgreSQL

```
sudo docker exec -it netdisco-backend psql -h
netdisco-postgresql -U netdisco -d netdisco
```

2.4.3.Test d'accessibilité IP depuis le backend

```
sudo docker exec -it netdisco-backend ping 172.20.0.42
```

2.4.4.Test SNMPv3 dans le conteneur du backend

```
sudo docker exec -u root -it netdisco-backend sh

apk update
apk add net-tools net-snmp

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

2.4.5.Test de découverte réseau détaillée (mode debug avancé)

Essaie de découverte réseau vérifiant le bon fonctionnement avec des log détaillé :

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

2.5.Logs/Erreur (non nuisible)

2.5.1.Accès au logs global

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

2.5.2.Erreur DB unversioned

Demande d'un upgrade de schéma déjà effectué :

```
DBIx::Class::Schema::Versioned::_on_connect(): Your DB is
currently unversioned. Please call upgrade on your schema to
sync the DB. at /home/netdisco/perl5/lib/perl5/DBICx/Sugar.pm
line 121
```

2.5.3.Erreur session_cookie_key

Demande d'une clé de session, non utile au lancement et au bon fonctionnement de l'outil.

```
Error while loading /home/netdisco/perl5/bin/netdisco-web-fg:
The setting session_cookie_key must be defined at
/home/netdisco/perl5/lib/perl5/Dancer/Session/Cookie.pm line
38.
```

2.5.4.Erreur de daemon:failed

Erreur lié à la clé de session :

Error response from daemon: failed to set up container networking: network ec8f5d358be3d873a026f0a221e70dd940e384e4fc91e7ebb46420e7f9e79b11 not found

tcpdump port 161

ss -tulpen | grep 5000

ip route

Script de découverte pour Netbox

1.Découverte des machines

1.1.Récupération et configuration des fichiers

1.1.1.Récupération des fichiers pour Netdisco

```
git clone
https://github.com/drygdryg/netbox-device-autodiscovery.git
```

1.1.2.Installer les outils nécessaires

```
pip install -r requirements.txt
```

Dans ce fichier requierements.txt :

```
sudo apt install python3-venv -y # version 3.11
sudo apt install nmap
pip install pysnmp
sudo apt install snmp
pip install pynetbox
pip install toml
pip install python-nmap
pip install lxml
pip install pysnmp==4.4.12
pip install pyasn1==0.4.8
pip install xmltodict
pip install pynetbox pyyaml
```

1.1.3.Créer un dossier pour NetDisco

```
cd netbox-device-autodiscovery
```

1.1.4.Installer une version antérieur à Python 3.12

On ajoute le PPA de Python 3.11 :

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
```

Installation de Python 3.11 :

```
sudo apt install python3.11 python3.11-venv python3.11-dev
```

1.1.5.Créer un environnement virtuel Python et Activer l'environnement virtuel

```
python -m venv netbox-discovery-env
source netbox-discovery-env/bin/activate
```

Pour désactiver un environnement virtuel :

```
deactivate
```

Pour supprimer un environnement virtuel si erreur :

```
rm -rf netbox-discovery-env
```

1.2.Découverte automatique avec `netdisco`

- Un token d'API NetBox (Admin → API Tokens)
- L'URL de ton NetBox (ex: http://192.168.100.160:8000)
- Python avec requests installé (tu l'as déjà)
- Un compte avec les droits d'écriture dans NetBox

Ajout automatique des équipements dans NetBox via son **API REST**
Réutilisation du Token généré dans NetBox

1.2.1.Configuration du fichier .toml

```
cp configuration.example.toml configuration.toml
```

```
nano configuration.toml
```

Dans ce fichier, je remplace le token, l'url :

```
[netbox]
url = "http://192.168.100.160:8000"
api_token = "04946ef59ffeb57bc7a0e7c6ac73f787eb272c57"
ssl_verify = false
# ssl_verify = "/path/to/certfile"
default_devices_site = "Autodiscovered devices"
```

Ainsi que la plage d'adresse IP scanné et le nombre de retries possible :

```
[nmap_network_scanner]
nmap_guess_os = true
nmap_additional_args = "-T4 -n"
snmp_recognition_enabled = true
snmp_recognition_retries = 3
shared_snmp_communities = ["public", "abcdef12345"]
snmp_retry_count = 2
networks = ["192.168.100.0/30"] # pour commencer de manière plus simple
```

1.2.2.Désactivation du module Avaya pour les téléphones

Ce module crée énormément d'erreur je reviendrai dessus dans la suite du projet pour l'instant :

```
# Désactivation du module "avaya_ip_office_phones_enumerator"
     if module_name == 'avaya_ip_office_phones_enumerator':
          log.info('Module "avaya_ip_office_phones_enumerator"
désactivé temporairement.>
          continue  # Ignore ce module et passe au suivant
```

1.2.3.Autorisation d'effectuer un nmap sans entrer de password

Dans /etc/sudoers.tmp avec :

```
sudo visudo

# User privilege specification
root ALL=(ALL:ALL) ALL
pfamchon ALL=(ALL) NOPASSWD: /usr/local/bin/nmap
```

1.2.4.Json (à détailler j'ai pas tout capté)

Une histoire de texte en brut...de sortie json....

1.2.5.Création d'une target_list

La liste target_list est mal construite (peut-être vide ?), créant des erreur, m'obligeant à utiliser cette commande dans nmap_network_scanner.py :

```
scan_results = nmap.scan(hosts='192.168.1.0/24',
arguments=nmap_arguments)
```

1.2.6.Ajout d'une vérification dans create_or_update_nb_obj > run.py

1.2.7.Ajout d'un rôle par défaut car manquant

1.2.8.Retirer la suppression automatique

Pour retirer la suppression automatique des IP dans Netbox, ip précédemment découverte, lors de nouveau scan sur des plages ip différentes, ce qui par défaut dans ce code supprime les IP qu'il n'a pas trouvé lors du dernier scan (soit le scan le plus récent) :

1.2.9.Lancer le script run.py

```
sudo env "PATH=$PATH" python run.py
```

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

devices : OK

\copy (SELECT d.name AS name, d.ip AS primary_ip4, 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);

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

devices types : OK

\copy (SELECT DISTINCT d.vendor AS manufacturer, d.model AS model, LOWER(REPLACE(d.model, ' ', '-')) AS slug, 1 AS u_height FROM device d WHERE d.vendor IS NOT NULL AND d.vendor != '' AND d.model IS NOT NULL AND d.model != '') TO '/tmp/netdisco_device_types.csv' WITH (FORMAT CSV, HEADER);

manufacturers : OK

\copy (SELECT DISTINCT d.vendor AS name, LOWER(REPLACE(d.vendor, ' ', '-')) AS slug FROM device d WHERE d.vendor IS NOT NULL AND d.vendor != '') TO '/tmp/netdisco_manufacturers.csv' WITH (FORMAT CSV, HEADER);

macs : NON

\copy (SELECT d.mac AS mac_address, d.name AS device, (SELECT dp_sub.port FROM device_port dp_sub WHERE dp_sub.ip = d.ip ORDER BY dp_sub.port ASC LIMIT 1) AS interface, 'false' AS is_primary FROM device d WHERE d.mac IS NOT NULL AND d.mac != '00:00:00:00:00:00') TO '/tmp/netdisco_device_macs_to_interfaces.csv' WITH (FORMAT CSV, HEADER);

ip : OK (prendre que vlan9)

\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);

vlan : oublie

prefix : oublie

vrf : oublie

pour le script python associant les vlan au interfaces de tout les equipement :

\copy (WITH DuplicatedPortLabels AS (SELECT dp.ip, dp.descr AS port_label FROM public.device_port dp GROUP BY
dp.ip, dp.descr HAVING COUNT(*) > 1) SELECT d.name AS device_name, CASE WHEN EXISTS (SELECT 1 FROM DuplicatedPortLabels dpl WHERE dpl.ip = dp.ip AND dpl.port_label = dp.descr) THEN dp.descr || '-' || dp.name ELSE dp.descr END AS interface_name, COALESCE((SELECT dpv_native.vlan::text FROM public.device_port_vlan dpv_native WHERE dpv_native.ip = dp.ip AND dpv_native.port = dp.name AND dpv_native.native IS TRUE LIMIT 1), dp.vlan) AS untagged_vlan_vid, STRING_AGG(DISTINCT dpv.vlan::text, ',' ORDER BY dpv.vlan::text) FILTER (WHERE dpv.native IS FALSE) AS tagged_vlan_vids FROM public.device d JOIN public.device_port dp ON d.ip = dp.ip LEFT JOIN public.device_port_vlan dpv ON dp.ip = dpv.ip AND dp.name = dpv.port GROUP BY d.name, dp.name, dp.ip, dp.vlan, dp.descr ORDER BY d.name, interface_name) TO '/tmp/test2.csv' WITH (FORMAT CSV, HEADER);
COPY 13146

interface : OK

\copy (WITH DuplicatedLabels AS (SELECT d.name AS device_name, dp.descr AS port_label FROM public.device d JOIN public.device_port dp ON d.ip = dp.ip GROUP BY d.name, dp.descr HAVING COUNT(dp.descr) > 1) SELECT d.name AS device, CASE WHEN EXISTS (SELECT 1 FROM DuplicatedLabels dl WHERE dl.device_name = d.name AND dl.port_label = dp.descr) THEN dp.descr || '-' || dp.name ELSE dp.descr END AS name, dp.name AS label, CASE WHEN dp.descr ILIKE 'HundredGigE%' OR dp.descr ILIKE 'Hu%' THEN '100gbase-x-qsfp28' WHEN dp.descr ILIKE 'FortyGigE%' OR dp.descr ILIKE 'Fo%' THEN '40gbase-x-qsfpp' WHEN dp.descr ILIKE 'TwentyFiveGigE%' OR dp.descr ILIKE 'Twe%' THEN '25gbase-x-sfp28' WHEN dp.descr ILIKE 'TenGigabitEthernet%' OR dp.descr ILIKE 'Te%' THEN '10gbase-x-sfpp' WHEN dp.descr ILIKE 'GigabitEthernet%' OR dp.descr ILIKE 'Gi%' THEN '1000base-t' WHEN dp.descr ILIKE 'FastEthernet%' OR dp.descr ILIKE 'Fa%' THEN '100base-tx' WHEN dp.descr ILIKE 'Ethernet%' OR dp.descr ILIKE 'Eth%' THEN '1000base-t' WHEN dp.descr ILIKE 'Port-channel%' OR dp.descr ILIKE 'Po%' THEN 'lag' WHEN dp.descr ILIKE 'VSL Link%' THEN 'lag' WHEN dp.descr ILIKE 'Vlan%' OR dp.descr ILIKE 'Vl%' THEN 'virtual' WHEN dp.descr ILIKE 'Loopback%' OR dp.descr ILIKE 'Lo%' THEN 'virtual' WHEN dp.descr ILIKE 'Tunnel%' OR dp.descr ILIKE 'Tu%' THEN 'virtual' WHEN dp.descr = 'Null0' OR dp.descr = 'Nu0' THEN 'virtual' WHEN dp.descr ILIKE 'Bluetooth%' OR dp.descr ILIKE 'Bl%' THEN 'ieee802.15.1' WHEN dp.descr ILIKE 'Bridge%' THEN 'bridge' ELSE 'virtual' END AS type, dp.mtu AS mtu, CASE WHEN dp.speed ILIKE '%Gbps' THEN (REPLACE(dp.speed, ' Gbps', '')::numeric * 1000)::bigint WHEN dp.speed ILIKE '%Mbps' THEN (REPLACE(dp.speed, ' Mbps', '')::numeric)::bigint WHEN dp.speed

ILIKE '%Kbps' THEN (REPLACE(dp.speed, ' Kbps', '')::numeric / 1000)::bigint WHEN dp.speed ~ '^[0-9]+$' THEN (dp.speed::numeric / 1000000)::bigint ELSE NULL END AS speed, CASE WHEN dp.up = 'up' THEN 'true' ELSE 'false' END AS enabled, dp.duplex AS duplex, CASE WHEN dp.descr ILIKE 'Vlan%' OR dp.descr ILIKE 'Loopback%' OR dp.descr ILIKE 'Tu%' OR dp.descr ILIKE 'VSL Link%' OR dp.descr = 'Null0' THEN NULL WHEN COUNT(DISTINCT dv.vlan) FILTER (WHERE dv.vlan IS NOT NULL) > 1 THEN 'tagged' ELSE 'access' END AS mode FROM public.device d JOIN public.device_port dp ON d.ip = dp.ip LEFT JOIN public.device_vlan dv ON d.ip = dv.ip GROUP BY d.name, dp.name, dp.descr, dp.mtu, dp.speed, dp.up, dp.duplex, dp.vlan ORDER BY d.name, dp.descr) TO '/tmp/test1.csv' WITH (FORMAT CSV, HEADER)

\copy (SELECT dns, 'cisco' AS manufacturer, 'WIFI AP' AS role ‘active' AS status, ‘cisco AIR-CAP2702I-E-K9’ AS device_type, ‘SiteInconnu’ AS site FROM node_ip WHERE dns ILIKE 'ap%') TO '/tmp/ap.csv' WITH CSV HEADER;

\copy (SELECT dns, 'true' AS enabled, 'GigabitEthernet0' AS name, '1000base-tx' AS type, 'pd' AS poe_mode, 'type2-ieee802.3at' AS poe_type FROM node_ip WHERE dns ILIKE 'ap%') TO '/tmp/ap_ip.csv' WITH CSV HEADER;

\copy (SELECT dns, dns,ip, 'active' AS status, 'GigabitEthernet0' AS interface, 'true' AS is_primary FROM node_ip WHERE dns ILIKE 'ap%') TO '/tmp/ap_ip.csv' WITH CSV HEADER;

\copy (SELECT dns,dns, ip::text || '/32' AS ip_with_slash32, 'active' AS status, 'GigabitEthernet0' AS interface, 'true' AS is_primary FROM node_ip WHERE dns ILIKE 'ap%') TO '/tmp/ap_ip.csv' WITH CSV HEADER;
\copy (SELECT remote_id, remote_port, mac, 'true' AS is_primary FROM device_port WHERE remote_id ILIKE 'ap%') TO '/tmp/ap_mac.csv' WITH CSV HEADER;

3.

remplacer les équipement de la baie par ceux de netdiso
rajouter les mac qui reste et regarder sur quels équipements elles sont

token netbox : 992d9c996c7ae48c8f4be16411d8ffbf6669cbe1

netdisco déployment : ssh root@netdisco.int.utc.fr

laison de donnée
SELECT dns,os,os_ver,snmp_ver,snmp_comm,snmp_class,vtp_domain FROM device;

mot de passe netbox : 0dP0xRkfmnSaGDGw1RVHwT0vr

netdisco :

patricia : petittrain123

wilf : Boss123 meme password pour netbox

rémy : fromage123

danail : test123

jc :