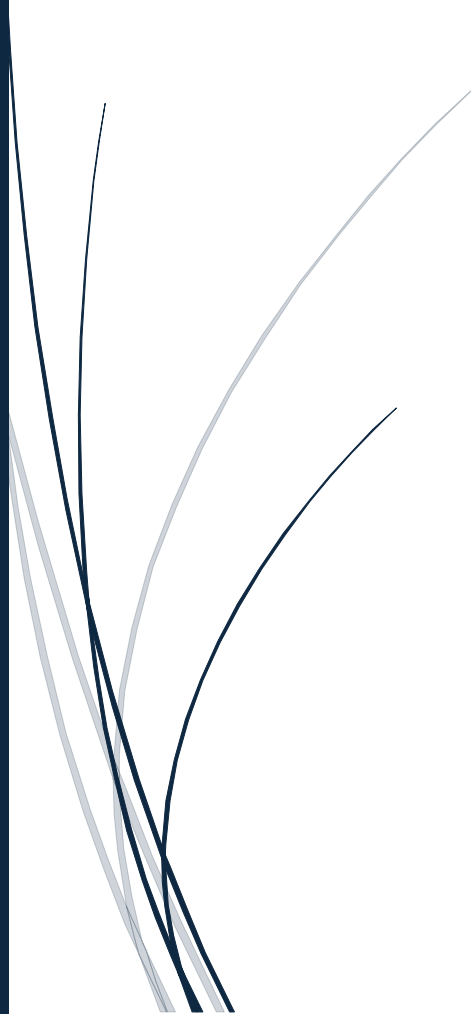
13/11/2025

TP 02 Module Devnet

Automatisation Réseau



Ronan FOURNEUVE
CPE-LYON

Table des matières

Partie 1 Générer et déployer des configurations réseau avec Python

1-Automatiser le déploiement de configuration réseau sur des équipements Cisco

Question 1.7 : Installez les dépendances nécessaires pour utiliser les fonctions du package Jinja2

Nous installons le module Python Jinja2 dans l'environnement virtuel géré par Pipenv.

```
pipenv install Jinja2
```

1.8 -Génération automatique de la configuration à l'aide de Jinja2 (TP-01)

Question 1.8.1 : Développez l'ensemble des templates Jinja2 qui vous permettront de générer la configuration nécessaire pour le bon fonctionnement du site CPE-MARSEILLE. (Pour rappel, le vlan d'admin (99) est déjà configuré sur les équipements.)

Ce template va générer les sous-interfaces VLAN 10 et 20 avec IP et description sur le routeur R2.

R2 :

```
hostname {{ hostname }}
{% for interface in interfaces %}
interface {{ interface.name }}
 description {{ interface.description }}
 ip address {{ interface.ip }} {{ interface.mask }}
 encapsulation dot1Q {{ interface.vlan_id }}
 no shutdown
{% endfor %}
```

ESW2 :

```
hostname {{ hostname }}


! Configuration des interfaces
{% for interface in interfaces %}
interface {{ interface.name }}
 description {{ interface.description }}
{% if interface.mode == "access" %}
 switchport mode access
 switchport access vlan {{ interface.vlan_id }}
```

```
{% elif interface.mode == "trunk" %}
 switchport mode trunk
 switchport trunk allowed vlan add {{ interface.vlan_id }}
{% endif %}
 no shutdown
{% endfor %}
{% if management is defined %}
! Configuration de l'IP de management
interface vlan {{ management.vlan }}
 ip address {{ management.ip_address }} {{ management.subnet }}
 no shutdown
{% endif %}
end
```

Question 1.8.2 : Définissez les structures de données dans des fichiers JSON ou YAML nécessaires pour remplir votre template JInja2 développé à la question 1. (Pour rappel, le vlan d'admin (99) est déjà configuré sur les équipements.)

Voici la configuration de mes fichiers vlan_R02.json et vlan_ESW2.json

vlan_R02.json :

```
{
"hostname": "R2",
"interfaces": [
{
 "name": "g0/0.10",
 "description": "Gateway pour le reseau 172.16.30.0/24",
 "ip": "172.16.30.254",
 "mask": "255.255.255.0",
 "vlan_id": "10"
 },
 {
 "name": "g0/0.20",
 "description": "Gateway pour le reseau 172.16.40.0/24",
 "ip": "172.16.40.254",
 "mask": "255.255.255.0",
 "vlan_id": "20"
 }
 ]
```

```
}
```

vlan_ESW2.json :

```
{
  "hostname": "ESW2",
  "interfaces": [
    {
      "name": "f1/1",
      "mode": "access",
      "vlan_id": "10",
      "description": "Connexion vers le VLAN 10"
    },
    {
      "name": "f1/2",
      "mode": "access",
      "vlan_id": "20",
      "description": "Connexion vers le VLAN 20"
    },
    {
      "name": "f1/0",
      "mode": "trunk",
      "vlan_id": "10,20",
      "description": "Connexion vers le routeur R2"
    }
  ]
}
```

Question 1.8.3 : Développez les fonctions vous permettant de générer automatiquement les configurations à partir du template Jinja2 et de la structure de données que vous avez défini à la question 1 et 2. Écrivez votre code dans le fichier scripts/create_config.py

La génération automatique des configurations est assurée en chargeant les données JSON, en appliquant les templates Jinja2 correspondants, puis en retournant les configurations finales du routeur et du switch.

```
import json
from jinja2 import Template, Environment,  FileSystemLoader
env = Environment(loader=FileSystemLoader("templates"))


def load_json_data_from_file(file_path):
    with open(file_path) as json_file:
```

```
        data = json.load(json_file)
    return data


def load_yaml_data_from_file(file_path):
    with open(file_path) as yaml_file:
        data = yaml.safe_load(yaml_file)
    return data


def render_network_config(template_name, data):
    template = env.get_template(template_name)
    return template.render(data)



def create_vlan_config_cpe_marseille():
    """
        Must return two values : router config and the switch config
    """
    esw2_data = load_json_data_from_file(file_path='data/vlan_ESW2.json')
    esw2_config = render_network_config(template_name='vlan_switch.j2', data=esw2_data)

    R2_data = load_json_data_from_file(file_path='data/vlan_R02.json')
    R2_config = render_network_config(template_name='vlan_router.j2', data=R2_data)
    return R2_config,esw2_config
```

Question 1.8.4 : Développez les fonctions vous permettant de sauvegarder automatiquement les configurations générées à la question 3 dans le dossier config de votre workspace

La fonction save_built_config enregistre automatiquement la configuration générée dans un fichier du dossier config en écrivant le contenu fourni dans le fichier spécifié.

```
def save_built_config(file_name, data):
    with open(file_name, "w") as f:
        f.write(data)
    return file_name
```

Question 1.8.5 : Répétez les questions 1 à 4 pour le site CPE-PARIS

Création des deux fichiers JSON pour ESW3 et R3

```
{
"hostname": "R3",
```

```
"interfaces": [
{
 "name": "g0/0.10",
 "description": "Gateway pour le reseau 172.16.50.0/24",
 "ip": "172.16.50.254",
 "mask": "255.255.255.0",
 "vlan_id": "10"
 },
 {
 "name": "g0/0.20",
 "description": "Gateway pour le reseau 172.16.60.0/24",
 "ip": "172.16.60.254",
 "mask": "255.255.255.0",
 "vlan_id": "20"
 }
 ]
}
```

```
{
  "hostname": "ESW3",
  "interfaces": [
    {
      "name": "f1/1",
      "mode": "access",
      "vlan_id": "10",
      "description": "Connexion vers le VLAN 10"
    },
    {
      "name": "f1/2",
      "mode": "access",
      "vlan_id": "20",
      "description": "Connexion vers le VLAN 20"
    },
    {
      "name": "f1/0",
      "mode": "trunk",
      "vlan_id": "10,20",
```

```
      "description": "Connexion vers le routeur R2"
    }
  ]
}
```

Modification aussi dans le script create_config.py

```
def create_vlan_config_cpe_paris():
    """
        Must return two values : router config and the switch config
    """
    esw3_data = load_json_data_from_file(file_path='data/vlan_ESW3.json')
    esw3_config = render_network_config(template_name='vlan_switch.j2', data=esw3_data)

    R3_data = load_json_data_from_file(file_path='data/vlan_R03.json')
    R3_config = render_network_config(template_name='vlan_router.j2', data=R3_data)
    return R3_config,esw3_config
```

et l'appel de la fonction

```
    r03_config, esw3_config = create_vlan_config_cpe_paris()
    save_built_config('config/vlan_R03.conf', r03_config)
    save_built_config('config/vlan_ESW3.conf', esw3_config)
```

1.9 - Automatisation réseau avec netmiko

Question 1.9.6 : Installez la librairie Netmiko dans votre workspace(TP02) et créez un fichier run_netmiko dans le dossier scripts.

La librairie Netmiko est installée dans le workspace via pipenv install netmiko, puis un fichier run_netmiko.py est créé dans le dossier scripts pour préparer l'exécution des connexions réseau automatisées.

pipenv install netmiko

Question 1.9.7 : Créez une variable de type dict représentant les informations nécessaires pour se connecter au routeur R1. Complétez avec les informations du routeur R1.

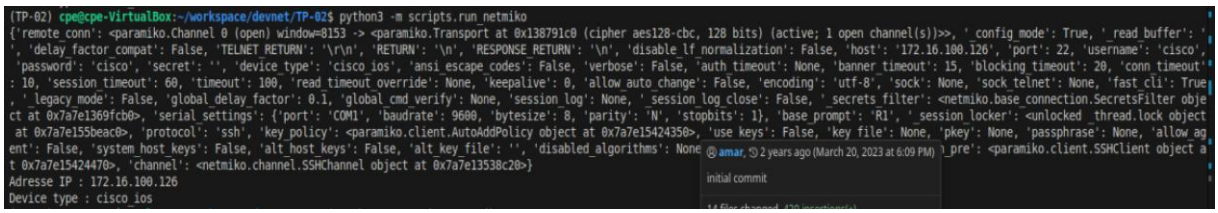
```
    r01 = {
    'device_type': 'cisco_ios',
    'host': '172.16.100.126',
    'username': 'cisco',
    'password': 'cisco'
    }
```

Question 1.9.8 : Initiez la connexion SSH au routeur R1 en instanciant la classe ConnectHandler

```
    net_connect = ConnectHandler(**r01)
```

Question 1.9.9 : Affichez la variable net_connect. Que contient la variable ? Quels sont les attributs de la variable ?

```
def question_9(net_connect):

    print(net_connect.__dict__)

    print("Adresse IP :", net_connect.host)

    print("Device type :", net_connect.device_type)
```



Question 1.9.10 : Quelle méthode de l'objet net_connect permet d'exécuter des commandes "show" ? Affichez l'état des interfaces du routeur R1 depuis le script run_netmiko.

```
def question_10(net_connect):

    output = net_connect.send_command("show ip interface brief")

    print(output)
```

```
(TP-02) cpe@cpe-VirtualBox:~/workspace/devnet/TP-02$ python3 -m scripts.run_netmiko
Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                unassigned      YES NVRAM  administratively down down
GigabitEthernet0/0         unassigned      YES NVRAM  up                    up
GigabitEthernet0/0.10      172.16.10.254   YES NVRAM  up                    up
GigabitEthernet0/0.20      172.16.20.254   YES NVRAM  up                    up
GigabitEthernet0/0.99      172.16.100.126  YES NVRAM  up                    up
Serial1/0                  10.1.1.1        YES NVRAM  up                    up
Serial1/1                  10.1.3.1        YES NVRAM  up                    up
Serial1/2                  unassigned      YES NVRAM  administratively down down
Serial1/3                  unassigned      YES NVRAM  administratively down down
GigabitEthernet2/0         172.16.100.62   YES NVRAM  up                    up
```

Question 1.9.11 : Utilisez le paramètre use_textfsm=True à la commande de la question précédente et affichez le résultat, qu'observez-vous ?

```
def question_11(net_connect):

    output = net_connect.send_command("show ip interface brief", use_textfsm=True)

    print(output)
```

```
python3 -m scripts.run_netmiko

[{'interface': 'Ethernet0/0', 'ip_address': 'unassigned', 'status': 'administratively down',
'proto': 'down'}, {'interface': 'GigabitEthernet0/0', 'ip_address': 'unassigned', 'status':
'up', 'proto': 'up'}, {'interface': 'GigabitEthernet0/0.10', 'ip_address': '172.16.10.254',
'status': 'up', 'proto': 'up'}, {'interface': 'GigabitEthernet0/0.20', 'ip_address':
'172.16.20.254', 'status': 'up', 'proto': 'up'}, {'interface': 'GigabitEthernet0/0.99',
'ip_address': '172.16.100.126', 'status': 'up', 'proto': 'up'}, {'interface': 'Serial1/0',
'ip_address': '10.1.1.1', 'status': 'up', 'proto': 'up'}, {'interface': 'Serial1/1',
'ip_address': '10.1.3.1', 'status': 'up', 'proto': 'up'}, {'interface': 'Serial1/2',
'ip_address': 'unassigned', 'status': 'administratively down', 'proto': 'down'}, {'interface':
'Serial1/3', 'ip_address': 'unassigned', 'status': 'administratively down', 'proto': 'down'},
{'interface': 'GigabitEthernet2/0', 'ip_address': '172.16.100.62', 'status': 'up', 'proto':
'up'}]
```

Cela nous le retourne au format « list »

```
(TP-02) cpe@cpe-VirtualBox:~/workspace/devnet/TP-02$ python3 -m scripts.run_netmiko

<class 'list'>
```

<u>Question 1.9.13 :</u> Affichez la table de routage du routeur R1 en utilisant le paramètre textfsm. Quel est le format de données retourné ?

```
def question_13(net_connect):

    output = net_connect.send_command("show ip route", use_textfsm=True)

    print(output)

    print(type(output))
```

```
python3 -m scripts.run_netmiko

[{'vrf': '', 'protocol': 'S', 'type': '', 'network': '172.16.100.192', 'prefix_length': '26',
'distance': '', 'metric': '', 'nexthop_ip': '', 'nexthop_vrf': '', 'nexthop_if': 'Serial1/1',
'uptime': '', 'flag': ''}, {'vrf': '', 'protocol': 'S', 'type': '', 'network': '172.16.100.128',
'prefix_length': '26', 'distance': '', 'metric': '', 'nexthop_ip': '', 'nexthop_vrf': '',
'nexthop_if': 'Serial1/0', 'uptime': '', 'flag': ''}, {'vrf': '', 'protocol': 'C', 'type': '',
'network': '172.16.100.64', 'prefix_length': '26', 'distance': '', 'metric': '', 'nexthop_ip':
'', 'nexthop_vrf': '', 'nexthop_if': 'GigabitEthernet0/0.99', 'uptime': '', 'flag': ''}, {'vrf':
'', 'protocol': 'C', 'type': '', 'network': '172.16.20.0', 'prefix_length': '24', 'distance':
'', 'metric': '', 'nexthop_ip': '', 'nexthop_vrf': '', 'nexthop_if': 'GigabitEthernet0/0.20',
'uptime': '', 'flag': ''}, {'vrf': '', 'protocol': 'C', 'type': '', 'network': '172.16.10.0',
'prefix_length': '24', 'distance': '', 'metric': '', 'nexthop_ip': '', 'nexthop_vrf': '',
'nexthop_if': 'GigabitEthernet0/0.10', 'uptime': '', 'flag': ''}, {'vrf': '', 'protocol': 'C',
'type': '', 'network': '172.16.100.0', 'prefix_length': '26', 'distance': '', 'metric': '',
'nexthop_ip': '', 'nexthop_vrf': '', 'nexthop_if': 'GigabitEthernet2/0', 'uptime': '', 'flag':
''}, {'vrf': '', 'protocol': 'C', 'type': '', 'network': '10.1.3.0', 'prefix_length': '30',
'distance': '', 'metric': '', 'nexthop_ip': '', 'nexthop_vrf': '', 'nexthop_if': 'Serial1/1',
'uptime': '', 'flag': ''}, {'vrf': '', 'protocol': 'C', 'type': '', 'network': '10.1.1.0',
'prefix_length': '30', 'distance': '', 'metric': '', 'nexthop_ip': '', 'nexthop_vrf': '',
'nexthop_if': 'Serial1/0', 'uptime': '', 'flag': ''}]

<class 'list'>
```

Cela nous le retourne au format « list »

Question 1.9.14 : Affichez la table de routage du routeur R1 en utilisant le paramètre textfsm. Quel est le format de données retourné ?

```
def question_14(net_connect):
    interfaces_status = net_connect.send_command("show ip interface brief", use_textfsm=True)
    print("État des interfaces du routeur R1 :")
    #print(interfaces_status)

    for iface in interfaces_status:
        intf_name = iface['interface']
        print(f"\nConfiguration de l'interface {intf_name} :")
        config = net_connect.send_command(f"show run interface {intf_name}")
        print(config)
```

Résultat :

```
Configuration de l'interface GigabitEthernet0/0 :
Building configuration...

Current configuration : 131 bytes
!
interface GigabitEthernet0/0
 description "test"
 no ip address
 duplex full
 speed 1000
 media-type gbic
 negotiation auto
end
```

Question 1.9.15 : Quelle méthode de l'objet net_connect permet d'exécuter des commandes en mode config ? Créez une interface loopback sur le routeur R1. Utilisez la méthode save_config de netmiko pour sauvegarder automatiquement la config

```
def question_15(net_connect):
    # Liste de commandes pour créer l'interface Loopback
    loopback_cmds = [
        "interface loopback1",
        "ip address 192.168.1.1 255.255.255.255",
```

```
        "description loopback interface from netmiko",
        "no shutdown"
    ]

    # Exécution des commandes en mode configuration
    output = net_connect.send_config_set(loopback_cmds)
    print("Résultat de la configuration :")
    print(output)

    # Sauvegarder la configuration
    save_output = net_connect.save_config()
    print("\nConfiguration sauvegardée :")
    print(save_output)
```

Résultat sur le shell

```
(TP-02) cpe@cpe-VirtualBox:~/workspace/devnet/TP-02$ python3 -m scripts.run_netmiko
Résultat de la configuration :
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#interface loopback1
R1(config-if)#ip address 192.168.1.1 255.255.255.255
R1(config-if)#description loopback interface from netmiko
R1(config-if)#no shutdown
R1(config-if)#end
R1#

Configuration sauvegardée :
write mem
Building configuration...
[OK]
```

Et sur le routeur

```
interface Loopback1
 description loopback interface from netmiko
 ip address 192.168.1.1 255.255.255.255
!
```

Question 1.9.16 : Supprimez l'interface loopback 1 depuis le script netmiko

Les fonctions Netmiko permettent de supprimer l'interface loopback1 avec no interface loopback1 et de déployer la configuration complète du fichier loopback_R01.conf, en appliquant toutes les commandes et en sauvegardant la configuration sur le routeur.

```
def question_16(net_connect):
    # Exécution en mode configuration
    output = net_connect.send_config_set("no interface loopback1")
    print("Résultat de la suppression de loopback1 :")
    print(output)

    # Sauvegarder la configuration
    save_output = net_connect.save_config()
    print("\nConfiguration sauvegardée après suppression :")
    print(save_output)
```

Résultat de la suppression de loopback1 :

```
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#no interface loopback1
R1(config)#end
R1#

Configuration sauvegardée après suppression :
write mem
Building configuration...
[OK]
R1#
```

```
 --More--
*Nov 13 09:59:46.287: %SYS-5-CONFIG_I: Configured from console by cisco on vty0 (172.16.100.2)
 --More--
*Nov 13 09:59:48.175: %LINK-5-CHANGED: Interface Loopback1, changed state to administratively down
*Nov 13 09:59:49.175: %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback1, changed state to down
```

Question 1.9.17.a : Déployez la configuration du fichier loopback_r01.conf depuis netmiko

```
def question_17(net_connect):
    # Lire les commandes depuis le fichier
    with open("config/loopback_R01.conf", "r") as f:
```

```
        commands = [line.strip() for line in f if line.strip()]  # ignore les lignes vides

    # Exécuter les commandes en mode configuration
    output = net_connect.send_config_set(commands)
    print("Résultat de l'exécution des commandes :")
    print(output)

    # Sauvegarder la configuration
    save_output = net_connect.save_config()
    print("\nConfiguration sauvegardée :")
    print(save_output)
```

```
(TP-02) cpe@cpe-VirtualBox:~/workspace/devnet/TP-02$ python3 -m scripts.run_netmiko
Résultat de l'exécution des commandes :
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#interface loopback 1
R1(config-if)#ip address 192.168.1.1 255.255.255.255
R1(config-if)#description "interface loopback 1"
R1(config-if)#no shut
R1(config-if)#interface loopback 2
R1(config-if)#ip address 192.168.2.1 255.255.255.255
R1(config-if)#description "interface loopback 2"
R1(config-if)#no shut
R1(config-if)#interface loopback 3
R1(config-if)#ip address 192.168.3.1 255.255.255.255
R1(config-if)#description "interface loopback 3"
R1(config-if)#no shut
R1(config-if)#interface loopback 4
R1(config-if)#ip address 192.168.4.1 255.255.255.255
R1(config-if)#description "interface loopback 4"
R1(config-if)#no shut
```

```
R1(config-if)#end
```

```
interface Loopback1
 description "interface loopback 1"
 ip address 192.168.1.1 255.255.255.255
!
interface Loopback2
 description "interface loopback 2"
 ip address 192.168.2.1 255.255.255.255
!
interface Loopback3
 description "interface loopback 3"
 ip address 192.168.3.1 255.255.255.255
!
interface Loopback4
 description "interface loopback 4"
 ip address 192.168.4.1 255.255.255.255
!
```

Question 1.9.17.b : Quelle est l'autre option possible pour déployer ce fichier de configuration depuis netmiko?

Une autre option pour déployer la configuration depuis Netmiko est d'utiliser send_config_from_file(), qui lit directement le fichier et exécute toutes les commandes en mode configuration sans avoir à ouvrir ou transformer le fichier en liste Python.

Question 1.9.18 : Supprimez les interfaces loopback du routeur r01 depuis netmiko.

```
def question_18(net_connect):
    for i in range(1, 5):
        loopback_name = f"loopback{i}"
        commands = [f"no interface {loopback_name}"]
        print(f"\nSuppression de l'interface {loopback_name} :")
        output = net_connect.send_config_set(commands)
        print(output)
    # Sauvegarder la configuration
    save_output = net_connect.save_config()
    print("\nConfiguration sauvegardée :")
    print(save_output)
```

```
Suppression de l'interface loopback1 :
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#no interface loopback1
```

```
R1(config)#end
R1#

Suppression de l'interface loopback2 :
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#no interface loopback2
R1(config)#end
R1#

Suppression de l'interface loopback3 :
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#no interface loopback3
R1(config)#end
R1#

Suppression de l'interface loopback4 :
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#no interface loopback4
R1(config)#end
```

Question 1.9.19 : Créez un fichier hosts.json dans le dossier inventory et inventoriez les équipements (routeur et switch) de l'architecture . Utilisez le modèle suivant :

```
[
    {
    "hostname":"R01",
    "ip":"172.16.100.126",
    "device_type": "cisco_ios",
    "username": "cisco",
    "password": "cisco"
    },
    {
    "hostname":"R02",
    "ip":"172.16.100.190",
    "device_type": "cisco_ios",
    "username": "cisco",
    "password": "cisco"
```

```
    },
    {
    "hostname":"R03",
    "ip":"172.16.100.254",
    "device_type": "cisco_ios",
    "username": "cisco",
    "password": "cisco"
    },
    {
    "hostname":"ESW1",
    "ip":"172.16.100.125",
    "device_type": "cisco_ios",
    "username": "cisco",
    "password": "cisco"
    },
    {
    "hostname":"ESW2",
    "ip":"172.16.100.189",
    "device_type": "cisco_ios",
    "username": "cisco",
    "password": "cisco"
    },
    {
    "hostname":"ESW3",
    "ip":"172.16.100.253",
    "device_type": "cisco_ios",
    "username": "cisco",
    "password": "cisco"
    }
]
```

Question 1.9.20 : Développez une fonction get_inventory qui retourne le contenu du fichier inventory/hosts.json

```
def get_inventory():
    """
    Lit le fichier inventory/hosts.json et retourne son contenu.
```

```
    """
    inventory_file = "inventory/hosts.json"
    try:
        with open(inventory_file, "r") as f:
            data = json.load(f)
        return data
    except FileNotFoundError:
        print(f"Erreur : le fichier {inventory_file} n'existe pas.")
        return []
hosts = get_inventory()
print(hosts)
```

Question 1.9.21 : Affichez la config de la sous-interface g0/0.99 de chaque routeur en parcourant la liste des hosts de votre inventory.

```
=== Connexion au routeur R01 (172.16.100.126) ===

Configuration de GigabitEthernet0/0.99 sur R01 :

Building configuration...

Current configuration : 175 bytes
!
interface GigabitEthernet0/0.99
 description 'sub-interface for admin vlan access - set by paramiko'
 encapsulation dot1Q 99
 ip address 172.16.100.126 255.255.255.192
end



=== Connexion au routeur R02 (172.16.100.190) ===

Configuration de GigabitEthernet0/0.99 sur R02 :

Building configuration...

Current configuration : 175 bytes
!
interface GigabitEthernet0/0.99
```

```
 description 'sub-interface for admin vlan access - set by paramiko'
 encapsulation dot1Q 99
 ip address 172.16.100.190 255.255.255.192
end


=== Connexion au routeur R03 (172.16.100.254) ===

Configuration de GigabitEthernet0/0.99 sur R03 :

Building configuration...

Current configuration : 175 bytes
!
interface GigabitEthernet0/0.99
 description 'sub-interface for admin vlan access - set by paramiko'
 encapsulation dot1Q 99
 ip address 172.16.100.254 255.255.255.192
end

def question_21():
    inventory = get_inventory()

    for device in inventory:
        print(f"\n=== Connexion au routeur {device['hostname']} ({device['ip']}) ===")

        try:
            # Connexion au routeur
            net_connect = ConnectHandler(
                device_type=device["device_type"],
                host=device["ip"],
                username=device["username"],
                password=device["password"]
            )

            # Commande à exécuter
            command = "show run interface GigabitEthernet0/0.99"
            output = net_connect.send_command(command)
```

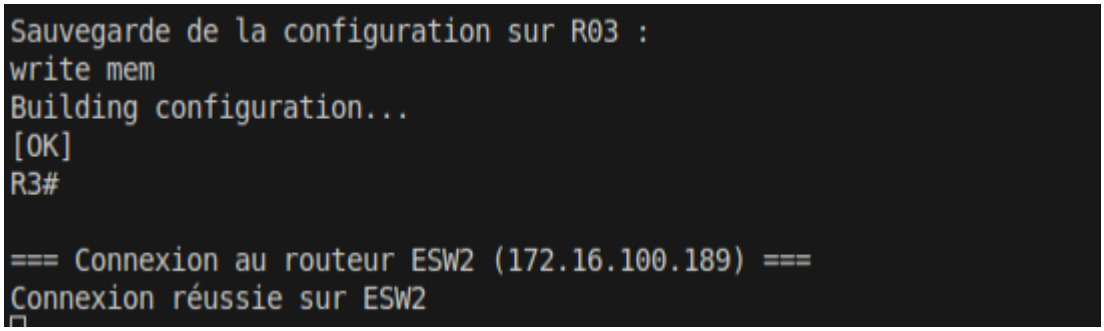
```
            print(f"\nConfiguration de GigabitEthernet0/0.99 sur {device['hostname']} :\n")
            print(output)

            net_connect.disconnect()

        except Exception as e:
            print(f"Erreur sur {device['hostname']} : {e}")
```

Question 1.9.22 : Déployez les configurations générées à la question 5 pour les routeurs et switchs des sites CPE-MARSEILLE et CPE-Paris. Pensez à sauvegarder vos implémentations avec la méthode save_config().

Les configurations VLAN des sites CPE-MARSEILLE et CPE-PARIS ont été déployées et sauvegardées avec save_config() avec la fonction de « question_22 ».

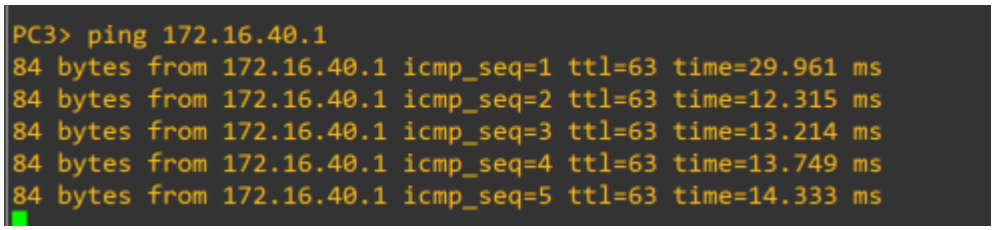
```
def question_22():
    inventory = get_inventory()

    for device in inventory:
        # On ne traite pas les routeurs dont le hostname != 'R1' de ESW1
        if 'R1' == device["hostname"] or 'ESW1' == device["hostname"]:
            continue

        print(f"\n=== Connexion au routeur {device['hostname']} ({device['ip']}) ===")

        try:
            # Conn
            device_params = {
                'device_type': device['device_type'],
                'host': device['ip'],
                'username': device['username'],
                'password': device['password']
            }
            net_connect = ConnectHandler(**device_params)
            net_connect.enable()

            # affiche le hostname
            print(f"Connexion réussie sur {device['hostname']}")
```

```
            config_file = f'config/vlan_{device["hostname"]}.conf'

            output = net_connect.send_config_from_file(config_file)

            print(f"\nRésultat de l'application de la configuration sur {device['hostname']}
:\n{output}")


            # Sauvegarde la configuration

            save_output = net_connect.save_config()

            print(f"\nSauvegarde de la configuration sur {device['hostname']}
:\n{save_output}")


            net_connect.disconnect()


        except Exception as e:

            print(f"Erreur sur {device['hostname']} : {e}")
```



La fonction parcourt l'inventaire des appareils, ignore R1 et ESW1, se connecte en SSH pour appliquer et sauvegarder une configuration VLAN spécifique, tout en gérant les erreurs pour continuer l'exécution.

Question 1.9.23 : Testez que les communications entre les vlans 10 et 20 du site CPE-Marseille sont fonctionnelles.



Les communications entre les VLAN 10 et 20 du site CPE-Marseille ont été testées et sont fonctionnelles.

Question 1.9.24 : Testez que les communications entre les vlans 10 et 20 du site CPE-Paris sont fonctionnelles.

Le test est fonctionnel à condition que les liens entre les routeurs R3 et R2 soient correctement configurés et que les routes IP soient correctement définies.

```
PC4> ping 172.16.60.1
84 bytes from 172.16.60.1 icmp_seq=1 ttl=62 time=60.682 ms
84 bytes from 172.16.60.1 icmp_seq=2 ttl=62 time=34.389 ms
84 bytes from 172.16.60.1 icmp_seq=3 ttl=62 time=32.660 ms
84 bytes from 172.16.60.1 icmp_seq=4 ttl=62 time=35.316 ms
84 bytes from 172.16.60.1 icmp_seq=5 ttl=62 time=36.012 ms
```

1.10- Automatisation réseau avec Napalm

Question 1.10.25 : Installez la librairie NAPALM à votre workspace (TP02) et créez un fichier nommé run_napalm.py dans le dossier scripts

Nous allons installer la librairie avec la commande : pipenv install napalm.

Question 1.10.26 : Créez une variable r01 de type dict représentant les informations nécessaires pour se connecter au routeur R1 et initialiser une connexion depuis le script napalm

```
if __name__ == "__main__":
    r01 = {
        'hostname':'172.16.100.62',
        'username': "cisco",
        'password': "cisco"
    }

    driver = get_network_driver('ios')
    device  = driver(**r01)
    device.open()
```

Question 1.10.27: Quel est la méthode de l'objet device permettant d'envoyer une commande show ? Exécutez la commande permettant de récupérer l'état des interfaces du routeur R1.

La méthode utilisée est device.cli().

C'est elle qui permet d'envoyer une ou plusieurs commandes show à un équipement réseau.

```
def question_27(device):

    command = ['show ip interface brief']

    output = device.cli(command)

    print(output)
```

```
{'show interfaces status': "^\n% Invalid input detected at '^' marker."}
(TP-02) cpe@cpe-VirtualBox:~/workspace/devnet/TP-02$ python3 -m scripts.run_napalm
{'show ip interface brief': 'Interface                IP-Address      OK? Method Status                Protocol\nEthernet0/0                unassigned      YES NVRAM  administratively down d
own    \nGigabitEthernet0/0         unassigned      YES NVRAM  up                    up      \nGigabitEthernet0/0.10      172.16.10.254   YES NVRAM  up                    up      \nGigabitEthe
rnet0/0.20      172.16.20.254   YES NVRAM  up                    up      \nGigabitEthernet0/0.99      172.16.100.126  YES NVRAM  up                    up      \nSerial1/0                 10.1
.1.1        YES NVRAM  up                    up      \nSerial1/1                 10.1.3.1        YES NVRAM  up                    up      \nSerial1/2                 unassigned      YES NVRA
M  administratively down down    \nSerial1/3                 unassigned      YES NVRAM  administratively down down    \nGigabitEthernet2/0         172.16.100.62   YES NVRAM  up
    up'}
```

Question 1.10.28: Quel est le format de sortie de la commande à la question précédente ? Quelle est la clé utilisée ?

La sortie retournée par device.cli() est au format dictionnaire Python (dict).

Je vérifie avec la fonction type de python

```
def question_26(device):
    command = ['show ip interface brief']
    output = device.cli(command)
    print(type(output))
```

Question 1.10.29: Quelle est la seconde méthode permettant de lire les données de configuration du routeur ? Affichez la table arp du routeur R1.

La seconde méthode « device.get_config() »

```
def question_29(device):
    output = device.get_config()
    print(output)
```

Liens vers la doc https://napalm.readthedocs.io/en/latest/base.html

get_arp_table(*vrf: str = ''*) → List[ARPTableDict]

Returns a list of dictionaries having the following set of keys:

- interface (string)
- mac (string)
- ip (string)
- age (float)

Si dans notre cas on veut lire la table arp on va utiliser la méthode get_arp_table

```
def question_29(device):
    #output = device.get_config()
    output = device.get_arp_table()
    print(output)
```

```
(TP-02) cpe@cpe-VirtualBox:~/workspace/devnet/TP-02$ python3 -m scripts.run_napalm
[{'interface': 'GigabitEthernet0/0.10', 'mac': 'CA:01:0E:18:00:08', 'ip': '172.16.10.254', 'age': -1.0}, {'interface': 'GigabitEthernet0/0.20', 'mac': 'CA:01:0E:18:00:08', 'ip': '172.16.20.254
', 'age': -1.0}, {'interface': 'GigabitEthernet2/0', 'mac': '08:00:27:87:33:F5', 'ip': '172.16.100.2', 'age': 11.0}, {'interface': 'GigabitEthernet2/0', 'mac': 'CA:01:0E:18:00:38', 'ip': '172.
16.100.62', 'age': -1.0}, {'interface': 'GigabitEthernet0/0.99', 'mac': 'CA:01:0E:18:00:08', 'ip': '172.16.100.126', 'age': -1.0}]
```

Question 1.10.30: Quel est le format de sortie de la commande à la question précédente ?

La sortie est de type *dict* (dictionnaire), ce que l'on peut confirmer en utilisant la fonction type() de Python.

```python
def question_30(device):
    output = device.get_arp_table()
    print(type(output))
```

```
(TP-02) cpe@cpe-VirtualBox:~/workspace/devnet/TP-02$ python3 -m scripts.run_napalm
<class 'list'>
```

Question 1.10.31.a: Déployez la configuration depuis le script napalm en utilisant la methode load_merge_candidate et en prenant soin d'afficher les changements apportés avant de commit votre config sur le routeur R1.

```python
def question_31():
    device.load_merge_candidate(filename='config/loopback_R01_1.conf')
    print(device.compare_config())
    device.commit_config()
```

- **load_merge_candidate() : charge la configuration à appliquer sans la déployer.**
- **compare_config() : affiche les différences entre la config actuelle et la config candidate.**
- **commit_config() : applique définitivement la configuration sur le routeur.**

```
+interface loopback 1
+    ip address 192.168.1.1 255.255.255.255
+    description "interface loopback 1"
-    no shut
+interface loopback 2
+    ip address 192.168.2.1 255.255.255.255
+    description "interface loopback 2"
-    no shutend
```

Question 1.10.31.b: Exécutez la commande show ip int brief sur le routeur R1, que

Remarquez-vous sur la ligne des interfaces loopback 1 et loopback 2 ? Comment expliquer cette différence ?

Sur la commande show ip int brief, les interfaces Loopback1 et Loopback2 ont toutes deux étés configurés via TFTP.

```
R1#show ip int brief
Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                unassigned      YES NVRAM  administratively down down
GigabitEthernet0/0         unassigned      YES NVRAM  up                    up
GigabitEthernet0/0.10      172.16.10.254   YES NVRAM  up                    up
GigabitEthernet0/0.20      172.16.20.254   YES NVRAM  up                    up
GigabitEthernet0/0.99      172.16.100.126  YES NVRAM  up                    up
Serial1/0                  10.1.1.1        YES NVRAM  up                    up
Serial1/1                  10.1.3.1        YES NVRAM  up                    up
Serial1/2                  unassigned      YES NVRAM  administratively down down
Serial1/3                  unassigned      YES NVRAM  administratively down down
GigabitEthernet2/0         172.16.100.62   YES NVRAM  up                    up
Loopback1                  192.168.1.1     YES TFTP   up                    up
Loopback2                  192.168.2.1     YES TFTP   up                    up
```

Question 1.10.32.a: Créez le template jinja2 pour une configuration OSPF et la structure de données pour chaque routeur (R1, R2 et R3) permettant de générer la configuration du backbone OSPF en utilisant les données du tableau ci-dessous.

Voici le Template Jinja pour la configuration d'OSPF

```
router ospf 1
 router-id {{ id_router }}
{% for network in networks %}
 network {{ network }} area 0
{% endfor %}
end
```

Question 1.10.32.b:

J'opte pour l'option A, consistant à créer un fichier distinct pour chaque routeur.

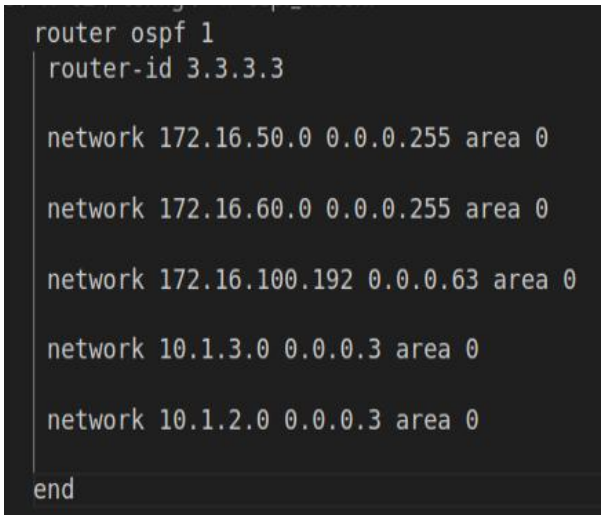
```
{
  "hostname": "R1",
  "id_router": "1.1.1.1",
  "networks": [
    "172.16.10.0 0.0.0.255",
    "172.16.20.0 0.0.0.255",
    "172.16.100.0 0.0.0.63",
    "10.1.3.0 0.0.0.3",
    "10.1.1.0 0.0.0.3"
  ]
}
```

Question 1.10.32.e:

La fonction create_ospf_config_cpe_lyon charge les données OSPF spécifiques au routeur R3 à partir d'un fichier JSON, puis génère et renvoie la configuration réseau correspondante en utilisant un template Jinja2.

```
def create_ospf_config_cpe_lyon():
    R3_data = load_json_data_from_file(file_path='data/ospf_R03.json')
    R3_config = render_network_config(template_name='config_ospf.j2', data=R3_data)

    return R3_config
```

La fonction retourne la configuration OSPF complète du routeur R3 pour le CPE de Lyon, prête à être appliquée sur le routeur.



Question 1.10.33: Déployez les configurations OSFP sur les routeurs R1, R2 et R3 (en une seule fois) depuis votre script python en utilisant les méthodes fournies par NAPALM. Pensez à sauvegarder vos déploiements (méthode commit)

Le script déploie en une seule exécution les configurations OSPF sur les routeurs R1, R2 et R3 en utilisant NAPALM, compare les configurations avant validation et sauvegarde les modifications avec la méthode commit_config

```
def question_33():
    r01 = {
        'hostname': '172.16.100.62',
        'username': 'cisco',
        'password': 'cisco'
    }
    r02 = {
        'hostname': '172.16.100.190',
        'username': 'cisco',
        'password': 'cisco'
    }
    r03 = {
        'hostname': '172.16.100.254',
        'username': 'cisco',
        'password': 'cisco'
    }

    routers = {'1': r01, '2': r02, '3': r03}

    for i in range(1, 4):
```

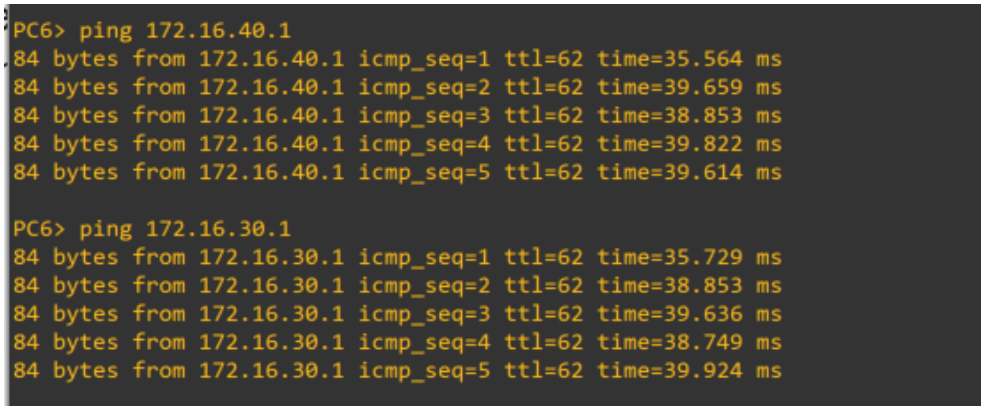
```
        driver = get_network_driver('ios')
        device = driver(**routers[str(i)])
        device.open()
        device.load_merge_candidate(filename=f'config/ospf_R{i}.conf')
        print(device.compare_config())
        device.commit_config()
        device.close()
```

Exemple résultat de la fonction :

```
+endend
+router ospf 1
+ router-id 2.2.2.2
+ network 172.16.30.0 0.0.0.255 area 0
+ network 172.16.40.0 0.0.0.255 area 0
+ network 172.16.100.64 0.0.0.63 area 0
+ network 10.1.2.0 0.0.0.3 area 0
+ network 10.1.1.0 0.0.0.3 area 0
+endend
+router ospf 1
+ router-id 3.3.3.3
+ network 172.16.50.0 0.0.0.255 area 0
+ network 172.16.60.0 0.0.0.255 area 0
+ network 172.16.100.192 0.0.0.63 area 0
+ network 10.1.3.0 0.0.0.3 area 0
+ network 10.1.2.0 0.0.0.3 area 0
```

Question 1.10.33: A ce stade si votre configuration est correcte les machines de chaque site peuvent communiquer les unes avec les autres. Testez également le fonctionnement du routage inter-vlan entre les différents sites

La communication entre les sites s'effectue correctement.



Question 1.10.35 : Développez une fonction permettant de créer un backup de chaque équipement réseau de l'infra à l'aide de la méthode get_config de napalm. Chaque backup devra être stocké dans le dossier config/backup et devra être nommé

```
def question_35():
    liste_hosts = get_inventory()
    for host in liste_hosts:
        nom_host = host['hostname']
        connexion_info = {
            'hostname': host['ip'],
            'username': host['username'],
            'password': host['password']
        }
        try:
            driver_ios = get_network_driver('ios')
            device = driver_ios(**connexion_info)
            device.open()
            config_sauvegarde = device.get_config()

            with open(f'backup/{nom_host}.bak', 'w') as fichier_sauvegarde:
                fichier_sauvegarde.write(str(config_sauvegarde))
            device.close()
        except Exception as erreur:
            print(f"Erreur lors de la connexion à {nom_host} : {erreur}")
            continue
```

Cette fonction parcourt la liste de périphériques réseau, se connecte à chacun d'eux, récupère leur configuration IOS, puis la sauvegarde dans un fichier.