TP 02
Générer et déployer des configurations réseau avec Python

5 ETI : Module Automatisation Réseau et conteneurisation Docker

Amar KANTAS

Partie 1
Générer et déployer des configurations réseau avec Python

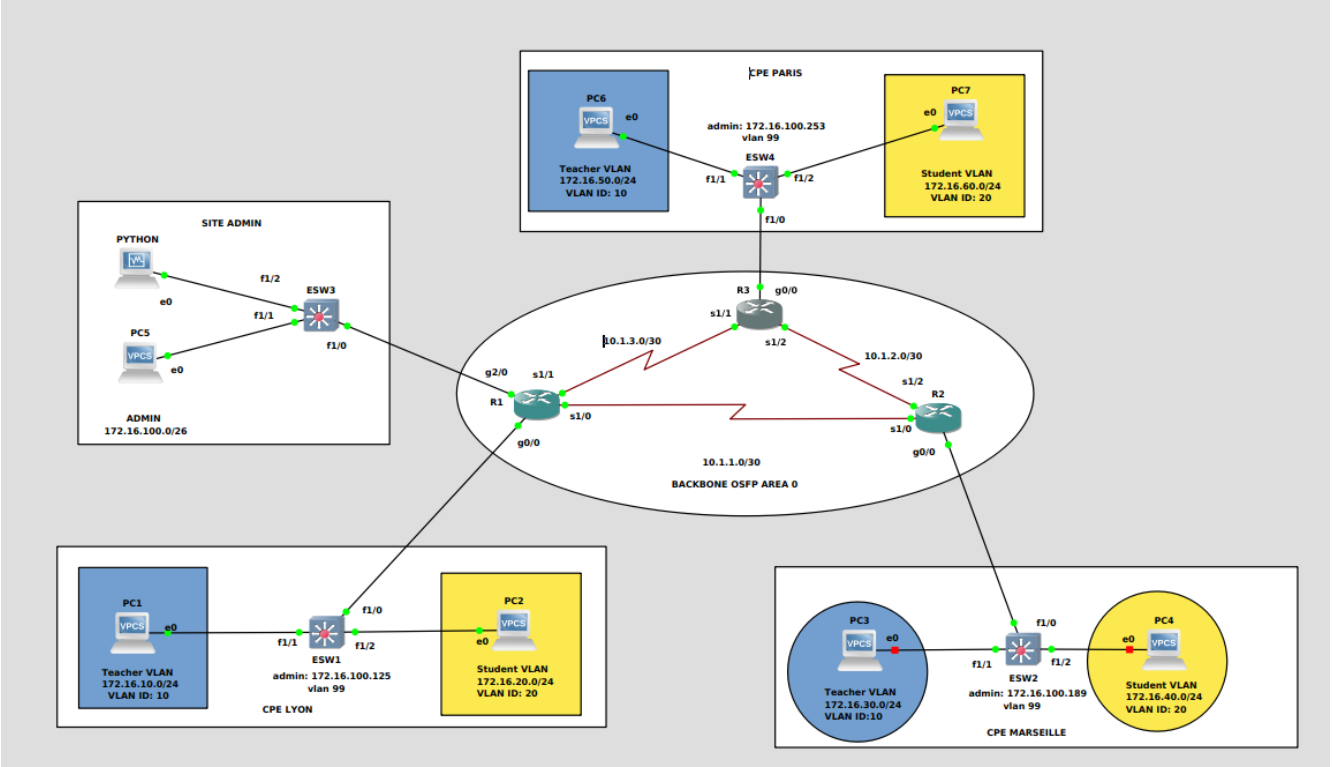
1-Automatiser le déploiement de configuration réseau sur des équipements Cisco

1.1- Objectif

Ce TP a pour objectif de développer en ensemble de scripts Python dans le but d'automatiser la création de configuration réseau et le déploiement de la configuration sur des routeurs Cisco 7200 et des switchs de niv2/ niv3 3725 Cisco.

Pour le déploiement automatique des configurations réseaux sur les équipements nous utiliserons plusieurs librairies python afin d'en tirer les avantages / inconvénients de chacune.

1.2 - Architecture réseau



1.3- Réseau d'administration

Nous disposons du réseau d'admin suivant : 172.16.100.0 /24. Ce réseau est découpé en 4 sous réseau /26 qui sont les suivants

Réseau SITE admin:
Network: 172.16.100.0/26
Broadcast: 172.16.100.63
HostMin: 172.16.100.1
HostMax: 172.16.100.62
Hosts/Net: 62

Réseau admin CPE-LYON:
Network: 172.16.100.64/26
Broadcast: 172.16.100.127
HostMin: 172.16.100.65
HostMax: 172.16.100.126
Hosts/Net: 62

Réseau admin CPE-MARSEILLE:
Network: 172.16.100.128/26
Broadcast: 172.16.100.191
HostMin: 172.16.100.129
HostMax: 172.16.100.190
Hosts/Net: 62

Réseau admin CPE-PARIS:
Network: 172.16.100.192/26
Broadcast: 172.16.100.255
HostMin: 172.16.100.193
HostMax: 172.16.100.254
Hosts/Net: 62

Seules les adresses ip du réseau d'admin seront utilisées pour les connexions SSH depuis la machine virtuelle PYTHON.

Chaque routeur et switch dispose d'une adresse ip dans le réseau d'admin. De manière générale les routeurs ont la dernière adresse ip de la plage et les switch l'avant-dernière. Les switch disposent d'une interface de management pour l'accès à distance (via l'ip admin) qui est configurée dans le vlan 99 (admin) de chaque site.

1.4 Liste des équipements

1.4.1 - liste des équipements réseau et de leurs adresses ip d'admin

Hostname	Ip address	SITE
PC5	172.16.100.1	ADMIN
PYTHON	172.16.100.2	ADMIN
R1	g2/0 : 172.16.100.62 g0/0.99 : 172.16.100.126	CPE-LYON
ESW1	172.16.100.125 (vlan 99)	CPE-LYON
R2	g0/0.99 : 172.16.100.190	CPE-MARSEILLE
ESW2	172.16.100.189 (vlan 99)	CPE-MARSEILLE
R3	g0/0.99 : 172.16.100.254	CPE-PARIS
ESW3	172.16.100.253 (vlan 99)	CPE-PARIS

1.4.2 - liste des équipements clients et de leurs adresses ip

Hostname	Ip address	SITE
PC1	172.16.10.1 (vlan 10 - teacher)	CPE-LYON
PC2	172.16.20.1 (vlan 20 - student)	CPE-LYON
PC3	172.16.30.1 (vlan 10 - teacher)	CPE-MARSEILLE
PC4	172.16.40.1 (vlan 20 - student)	CPE-MARSEILLE
PC6	172.16.50.1 (vlan 10 - teacher)	CPE-PARIS
PC7	172.16.60.1(vlan 20 - student)	CPE-PARIS

1.5- Prenez connaissance de l'architecture et de la configuration mise en place

Eléments déjà configurés dans l'architecture :

- L'ensemble des adresses ip d'admin et des clients PC est configuré
- Les liaisons point à point entre les routeurs du backbone OSPF sont également configurées (la configuration OSPF n'est pas implémentée)
- Des routes statiques sont configurées sur chaque routeur afin que le pc d'admin et la machine virtuelle (python) puissent joindre l'ensemble des routeurs et switchs sur leur ip d'admin
- Le site CPE-LYON est entièrement configuré
    - Routage inter-vlan (teacher - student - admin)
    - Commutation vlan (teacher - student - admin)

Travail demandé (résumé):

- Développez l'ensemble des scripts nécessaires pour générer la configuration réseau (Jinja2) du routage inter-vlan des sites : CPE-MARSEILLE et CPE-Paris
    - Développez l'ensemble des scripts nécessaires pour déployer automatiquement la configuration sur les équipements cibles
    - Testez le résultat de l'implémentation et faites vérifier par votre responsable pédagogique
- Développez l'ensemble des scripts nécessaires pour générer la configuration réseau (JInja2) du backbone OSPF (area 0)
    - Développez l'ensemble des scripts nécessaires pour déployer automatiquement la configuration sur les équipements cibles
    - Testez le résultat de l'implémentation et faites vérifier par votre responsable pédagogique
- Développez l'ensemble des scripts nécessaires pour générer des backups sur chaque équipement réseau de l'architecture
    - Testez le résultat de l'implémentation et faites vérifier par votre responsable pédagogique

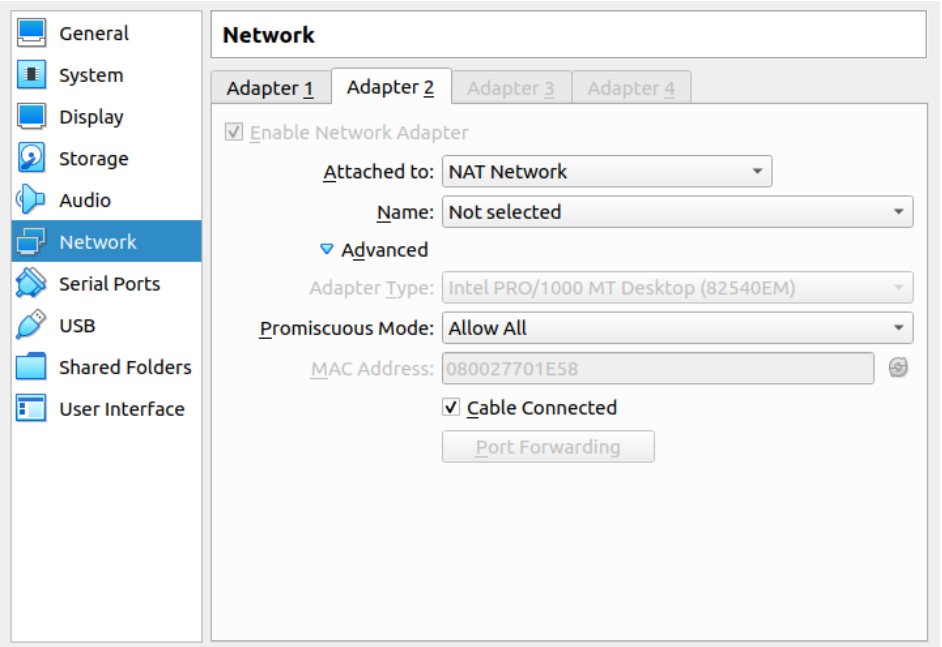
1.6 - Connectez-vous sur la machine virtuelle d'automatisation

Utilisez le login / password suivant : cpe / 123

Vérifiez que la machine virtuelle peut joindre l'ensemble des équipements du réseau d'ADMIN. Si l'accès au réseau d'admin ne fonctionne pas, vérifiez l'état des interfaces de la machine depuis virtualbox .

- La première interface doit être connectée à GNS3 (GenericDriverl)
- La seconde interface doit être en NAT

Vérifiez que le câble est "branché" au niveau des cartes réseaux de la VM :



Vérifiez également que la machine virtuelle PYTHON dispose d'une route vers le réseau d'admin : 172.16.100.0/24. Si ce n'est pas le cas , ajoutez la route :

sudo ip route add 172.16.100.0/24 via 172.16.100.62 dev <interface_name>

La machine virtuelle python doit pouvoir joindre les équipements suivants :

Hostname	Ip address	SITE
PC5	172.16.100.1	ADMIN
PYTHON	172.16.100.2	ADMIN
R1	g2/0 : 172.16.100.62 g0/0.99 : 172.16.100.126	CPE-LYON
ESW1	172.16.100.125 (vlan 99)	CPE-LYON
R2	g0/0.99 : 172.16.100.190	CPE-MARSEILLE
ESW2	172.16.100.189 (vlan 99)	CPE-MARSEILLE
R3	g0/0.99 : 172.16.100.254	CPE-PARIS
ESW4	172.16.100.253 (vlan 99)	CPE-PARIS

Vérifiez que la machine virtuelle peut accéder à internet. Si l'accès à internet ne fonctionne pas, vérifiez l'état des interfaces de la machine depuis virtualbox.

1.7 - Environnement de développement

Prenez connaissance de l'environnement de développement en ouvrant l'éditeur de code VsCode installé dans la machine virtuelle. Ajoutez le dossier workspace/tp02 à votre éditeur si ce n'est pas déjà fait.

Activez l'environnement virtuel en vous plaçant à la racine du projet et en entrant : **pipenv shell**

Installez les dépendances pour utiliser les fonctions du package Jinja2:

pipenv install Jinja2

Note : Vous pouvez directement utiliser le terminal depuis l'éditeur VsCode
Créez un fichier test.py dans le dossier scripts et affichez simplement un helloworld sur la sortie standard.

Exécutez le script (python3 -m scripts.test) pour vous assurer que tout fonctionne correctement. En cas de problème, n'hésitez pas à solliciter le responsable pédagogique.

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

A ce stade vous avez pris connaissance de l'architecture et des éléments de configuration en place sur les équipements réseaux de l'architecture. Vous avez également vérifié le bon fonctionnement de votre environnement de développement.

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.)

   L'objectif est que les machines du site de Marseille puissent communiquer entre elles.

   Nommez les fichiers de templates de la manière suivant:
   - vlan_router.j2
   - vlan_switch.j2

   Note : Appuyez-vous sur ce qui a été réalisé en TP-01 et en cours

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.)

   Nommez les fichiers de data de la manière suivante :
   - vlan_R02.json ou vlan_R02.yaml
   - vlan_ESW2.json ou vlan_ESW2.yaml

   Note : Appuyez-vous sur ce qui a été réalisé en TP-01 et en cours

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

Note : Appuyez-vous sur ce qui a été réalisé en TP-01 et en cours. Vous pouvez récupérer la fonction render_network_config du TP-01.

Le code permettant de créer la config à partir du template jinja2 et du jeux de données (JSON / YAML) doit être placé dans une fonction nommée : create_vlan_config_cpe_marseille() qui retournera la config générée de chaque équipement (Routeur et Switch).

Cette fonction peut également faire appel à la fonction render_network_config() si vous l'avez récupéré du tp01.

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

   Nommez les fichiers de config de la manière suivante :
   - vlan_R02.conf
   - vlan_ESW2.conf

   Le code permettant de créer un fichier de conf et de sauvegarder le résultat de la fonction de la question 3 doit être placé dans une fonction nommée : save_built_config() qui prend en paramètre la config générée et le nom du fichier de sauvegarde.

   Faites vérifier le résultat par votre responsable pédagogique

5) Répétez les questions 1 à 4 pour le site CPE-PARIS

   Faites vérifier le résultat par votre responsable pédagogique

A ce stade le main de votre script python ne doit contenir que l'appel aux fonctions :

- create_vlan_config_cpe_marseille()
- create_vlan_config_cpe_paris()
- save_built_config()

Aucune configuration ne doit être implémentée sur les équipements pour le moment. Nous reviendrons sur ces éléments de configurations un peu plus tard dans le TP.

1.9 -Automatisation réseau avec Paramiko

Paramiko est définit comme une implémentation du protocole SSHv2 en python qui permet de fournir les fonctionnalités ssh client et serveur.

Autrement dit, paramiko est une bibliothèque python permettant d'interagir avec des équipements au travers du protocole ssh.

6) Installez paramiko à votre workspace (/TP02):

   pipenv install paramiko

7) Créez un fichier nommé run_paramiko.py dans le dossier scripts de votre workspace (TP02) et importez le package Paramiko en en-tête du fichier :

   ```
   import time
   import paramiko
   ```

   Note : Nous aurons également besoin du package time qui est indépendant à paramiko

8) Définissez une variable device de type dictionnaire avec l'adresse ip d'admin de R1 et le login / password du user ssh (cisco / cisco)

   Les clés du dict à utiliser sont les suivantes : ip, username, password

9) Initiez la connexion avec le routeur R1 depuis le script paramiko (dans le__name__)

   ```
   ssh = paramiko.SSHClient() #initialization of SSHClient class

   #Set policy to use when connecting to servers without a known host key
   ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
   ```

```
#init connection with the remote device
ssh.connect(device.get('ip'),
            username=device.get('username'),
            password=device.get('password'),
            look_for_keys=False, allow_agent=False,
            timeout=5)
```

10) Invoquez le shell distant pour préparer l'envoi de commandes à l'équipement distant

```
nbytes = 65535

#get the remote shell
remote_conn = ssh.invoke_shell()
```

11) Envoyez la commande "show ip interface brief" à l'équipement et afficher le résultat sur la sortie standard

```
remote_conn.send(".....") # à compléter
time.sleep(.5)
output = remote_conn.recv(nbytes) #Get output data from the channel

ssh.close()
```

12) Utilisez la méthode **decode("utf8")** pour avoir un affichage "propre" sur le terminal

13) Commentez la ligne suivante de votre script : time.sleep(.5) et exécutez le script. Que se passe-t-il ? Pourquoi ?

14) Modifiez la valeur de la variable nbytes à 500 et exécutez le script. Que se passe-t-il ? Pourquoi ?

15) Créez une interface de loopback depuis le script python sur le routeur R1 en utilisant la connexion ssh

nom de l'interface : loopback 1

ip de l'interface : 192.168.1.1/32
description: "loopback interface from paramiko"

Faites vérifier par le responsable pédagogique

16) Affichez la configuration de l'interface de loopback 1 depuis le script paramiko en utilisant la connection ssh

Faites vérifier par le responsable pédagogique

17) Modifiez votre script afin d'afficher l'état des interfaces du routeur R2 et R3

Faites vérifier par le responsable pédagogique

18) Modifiez votre script afin d'ajouter une description à la sous-interface d'admin des routeurs R1, R2 et R3

description:
"sub-interface for admin vlan access - set by paramiko"

Faites vérifier par le responsable pédagogique

19) Développez une fonction (save_config par exemple) permettant de sauvegarder la configuration d'un équipement passé en paramètre

20) Exécutez le script de manière à lancer une sauvegarde de configuration sur les équipements: R1, R2 et R3

Comme vous avez pu le voir Paramiko vous fourni les outils nécessaires pour interagir en SSH avec les équipements compatibles mais dans certains cas et selon les constructeurs Paramiko peut ne pas être le choix adéquat.

Il est également limité dans son interface avec les équipements réseaux, il faut à chaque fois définir dans quel mode on se trouve avant de lancer les commandes (enable, conf t, etc..).

Paramiko répond à des besoins d'automatisations de tâches SSH spécifiques mais n'est pas la solution la plus adaptée pour une automatisation sur des équipements réseaux à grande échelle.

1.10 - Automatisation réseau avec netmiko

Netmiko a été conçu pour simplifier le dialogue SSH avec les équipements d'un grand nombre de constructeurs d'équipements de réseaux (https://github.com/ktbyers/netmiko/blob/develop/PLATFORMS.md).

Netmiko est une solution pour contourner les difficultés d'utilisation de la librairie Paramiko et gagner du temps sur les changements de config.

Netmiko repose sur Paramiko pour l'interaction SSH avec les équipements mais fournit une surcouche qui simplifie l'exécution des commandes et le déploiement de configuration.

Note : Supprimez l'interface loopback configurée sur le routeur R1

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

pipenv install netmiko

22) Créez une variable de type dict représentant les informations nécessaires pour se connecter au routeur R1

```
from netmiko import ConnectHandler

r01 = {
    'device_type': 'cisco_ios',
    'host':   '10.10.10.10',
    'username': 'test',
    'password': 'password'
}
```

23) Initiez la connexion SSH au routeur R1 en instanciant la classe ConnectHandler

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

24) Affichez la variable net_connect. Que contient la variable ? Quels sont les attributs de la variable ?

Aide:
https://docs.python.org/3/library/stdtypes.html#object.__dict__

25) 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.

Documentation :
https://github.com/ktbyers/netmiko/blob/develop/EXAMPLES.md

Faites vérifier par le responsable pédagogique

26) Utilisez le paramètre use_textfsm=True à la commande de la question 25. Affichez le résultat, qu'observez-vous ?

Aide: https://pynet.twb-tech.com/blog/netmiko-and-textfsm.html

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

28) Affichez l'état des interfaces du routeur R1 et dans un second temps afficher la configuration de chacune des interfaces retournée par la première commande.

Faites vérifier par le responsable pédagogique

29) 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.

nom de l'interface : loopback 1
ip de l'interface : 192.168.1.1/32
description: "loopback interface from netmiko"

Documentation :
https://github.com/ktbyers/netmiko/blob/develop/EXAMPLES.md

Faites vérifier par le responsable pédagogique

30) Supprimez l'interface loopback 1 depuis le script netmiko

31) Créez un fichier de configuration dans le dossier config, nommé loopback_R01.conf. Dans ce fichier de configuration vous allez copier / coller la configuration suivante :

```
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 shut

interface loopback 3
   ip address 192.168.3.1 255.255.255.255
   description "interface loopback 3"
   no shut

interface loopback 4
   ip address 192.168.4.1 255.255.255.255
   description "interface loopback 4"
   no shut
```

a) Déployez la configuration du fichier loopback_r01.conf depuis netmiko.
Documentation :
https://github.com/ktbyers/netmiko/blob/develop/EXAMPLES.md

Faites vérifier par le responsable pédagogique

b) Quelle est l'autre option possible pour déployer ce fichier de configuration depuis netmiko?

32) Supprimez les interfaces loopback du routeur r01 depuis netmiko.

33) 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"
    },
    ......
]
```

34) Développez une fonction get_inventory qui retourne le contenu du fichier inventory/hosts.json

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

Faites vérifier par le responsable pédagogique

36) 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().

Faites vérifier par le responsable pédagogique

37) Testez que les communications entre les vlans 10 et 20 du site CPE-Marseille sont fonctionnelles.

38) Testez que les communications entre les vlans 10 et 20 du site CPE-Paris sont fonctionnelles.

A ce stade du TP vous avez pris en main les librairies Paramiko et Netmiko et avez pu observer des différences notables entre ces deux librairies.

Du côté de votre architecture réseau, chaque machine de chaque site doit pouvoir communiquer en local. Les machines ne peuvent cependant pas communiquer d'un site à un autre.

Vous allez devoir développer un script à l'aide de la librairie Napalm pour générer et déployer les configurations nécessaires pour le bon fonctionnement du backbone OSPF Area 0.

1.11- Automatisation réseau avec Napalm

Network Automation and Programmability (NAPALM) est une librairie python permettant essentiellement d'automatiser la gestion de configuration pour des équipements réseaux.

Par exemple, pour des équipements Cisco IOS, elle permet d'automatiser les actions suivantes:

- Replace : remplacer l'ensemble de la config en cours d'exécution par une configuration complètement nouvelle
- Merge: fusionner un ensemble de modifications contenues dans un fichier avec la configuration en cours
- Compare: comparer le contenu du nouveau fichier de conf proposé avec la config en cours d'exécution.
- Commit : confirmer le déploiement de la config
- Discard: revenir à la configuration running-config actuelle
- Rollback: ramener la config en cours d'exécution à une config qui a été enregistré avant le commit précédent

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

pipenv install napalm

40) 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

```
r01 = {
    'hostname':'ip_address',
    'username': user,
    'password': password
}

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

41) 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.

Aide : https://napalm.readthedocs.io/en/latest/base.html

Faites vérifier par le responsable pédagogique

42) Quel est le format de sortie de la commande à la question 42 ? Quelle est la clé utilisée ?

43) Quelle est la seconde méthode permettant de lire les données de configuration du routeur ? Affichez la table arp du routeur R1.

Aidez-vous de la documentation napalm : NAPALM_doc

Faites vérifier par le responsable pédagogique

44) Quel est le format de sortie de la commande à la question 43 ?

45) Créez un fichier de config nommé loopback R01.conf dans le dossier config et copiez / collez le contenu suivant:

```
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 shut
```

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.

AIdez-vous de la documentation napalm: Napalm doc

Faites vérifier par le responsable pédagogique

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 ?

46) 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.

a) Liste des réseaux OSFP:

Hostname	OSFP NETWORKs	SITE
R1	172.16.10.0/24 172.16.20.0/24 172.16.100.0/26	CPE-LYON
R2	172.16.30.0/24 172.16.40.0/24 172.16.100.64/26	CPE-MARSEILLE
R3	172.16.50.0/24 172.16.60.0/24 172.16.100.192/26	CPE-PARIS

b) Liste des templates jinja2 , fichiers de données et fichiers de conf **(option A - un fichier de data par routeur)**:

Template Jinja2	Data (Json ou yaml)	Ouput
templates/osfp.j2	data/ospf_r01.json ou data/ospf_r01.yaml	config/ospf_r01.conf
templates/osfp.j2	data/ospf_r02.json ou data/ospf_r02.yaml	config/ospf_r02.conf
templates/osfp.j2	data/ospf_r03.json ou data/ospf_r03.yaml	config/ospf_r03.conf

c) Liste des templates jinja2 , fichiers de données et fichiers de conf **(option B - un seul fichier de data global)**:

Template Jinja2	Data (Json ou yaml)	Ouput
templates/osfp.j2	data/ospf.json ou data/ospf.yaml	config/ospf_r01.conf
templates/osfp.j2	data/ospf.json ou data/ospf.yaml	config/ospf_r02.conf
templates/osfp.j2	data/ospf.json ou data/ospf.yaml	config/ospf_r03.conf

d) Faites vérifier vos templates Jinja2 et vos fichiers de données à votre responsable pédagogique

e) Générez la configuration OSPF pour chaque routeur depuis votre script python (en une seule fois) et stocker la configuration dans les fichiers de config (voir tableau du dessus)

Faites vérifier par votre responsable pédagogique

47) 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)

48) 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

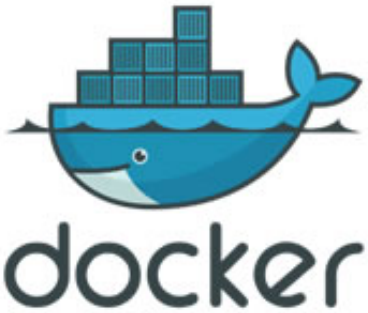
Faites vérifier par votre responsable pédagogique

49) 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 comme le modèle suivant :

Hostname	Backup
R1	R01.bak
ESW1	ESW1.bak
R2	R02.bak
ESW2	ESW2.bak
R3	R03.bak
ESW3	ESW3.bak

Faites vérifier par votre responsable pédagogique

Partie 2
Manipulation de conteneurs et de volumes



Le but de cette deuxième partie du TP est de créer nos propres images docker contenant notre stack python (environnement de développement) dans le but de pouvoir exécuter nos scripts de développement dans des conteneurs.

L'avantage d'exécuter nos scripts de développement dans des conteneurs est de s'affranchir des contraintes système de la machine hôte autrement dit notre stack devrait être fonctionnelle sur n'importe quel host contenant ou pas un environnement python.

2.0 - Création d'une image docker contenant la stack de développement python mise en place durant la partie 1 du TP.

1) Créez un fichier Dockerfile à la racine de votre workspace (TP_02). L'objectif du Dockerfile est de construire (build) une image docker contenant un environnement de développement sous Python 3.8 et disposant du package pipenv. Appuyez-vous sur les indications et les liens suivants:

    - Lien à consulter: https://hub.docker.com/_/python
    - Version python : 3.8
    - Commandes à exécuter lors du build:
        - Copiez votre workspace lors du build (contenu du dossier tp_02) dans le dossier /tp02 (workdir du conteneur)
        - Définissez le workdir de votre conteneur sur /tp02
        - installez les dépendances de votre Pipefile au moment du build : pipenv install –skip-lock (pour ignorer le fichier Pipefile.lock) –systeme (pour installer les packages directement sur le système)

2) Vérifiez que votre image a correctement été build à l'aide de la commande docker images

3) Invoquez un shell pour interagir avec votre conteneur à l'aide des options et arguments suivants :

    -it (interactive) <your_image_name> bash

4) Dans quel dossier vous trouvez-vous lorsque vous avez l'affichage du bash sur votre terminal ? Comment l'expliquez-vous ?

    Si vous n'avez rien de spécifique au niveau du prompt (nom du dossier) c'est que le build de l'image n'a pas donné le résultat attendu et dans ce cas là retournez à la question 1 pour corriger le problème.

5) Exécutez la commande ls, que voyez-vous ? Comment l'expliquez-vous ?

Si vous n'avez rien à la sortie de cette commande c'est que le build de l'image n'a pas donné le résultat attendu et dans ce cas là retournez à la question 1 pour corriger le problème.

6) Vérifiez à présent que les packages de votre Pipefile ont bien été installés à l'aide de la commande : pip3 freeze

7) Exécutez le script run_netmiko qui normalement doit faire appel à la fonction "question_49()" permettant de réaliser des backups comme mentionné à la question 49 de la partie 1. Que se passe-t-il ? Comment l'expliquer ?

Si vous avez une erreur à l'exécution du script de type "no module found" alors c'est que le build de l'image n'a pas donné le résultat attendu et dans ce cas là retournez à la question 1 pour corriger le problème.

Assurez-vous que votre machine puisse joindre l'ensemble des machines de votre GNS3, si ce n'est pas le cas ajoutez la route suivante à votre machine virtuelle:

sudo ip route add 172.16.100.0/24 via 172.16.100.62 dev enp0s3 (ou autre si votre interface possède un autre nom que enp0s3)

8) Maintenant que vous avez compris comment votre conteneur est en mesure de communiquer avec les équipements réseaux de votre GNS3, nous allons voir comment passer en paramètre des scripts à notre conteneur au moment du run (démarrage) de celui-ci.

    a) Modifiez le main de votre script run_netmiko pour que celui-ci appelle la fonction question_25() qui permet d'afficher l'état des interfaces du routeur R01.

    b) Faites en sorte d'exécuter la nouvelle version de votre script au moment du run de votre container précédemment créé.

        i) Aidez-vous de la documentation suivante : https://docs.docker.com/storage/bind-mounts/

        ii) Indicateurs pour "ordonner" au conteneur d'exécuter le script passé en paramètre lors du bind (voir plus haut) : il suffit simplement de passer en paramètre la commande permettant d'exécuter le script cible lors du démarrage de votre container (voir le cours et / ou la doc sur google)

9) Modifier le nom de destination du fichier passé lors du bind (option -v) de la question 8, nommez le run_netmiko2.py et rejouez l'étape 2 de la question 8. A présent invoquez un bash lors du run de votre container et vérifiez si le fichier run_netmiko2.py est dans le dossier scripts. D'après vous pourquoi le fichier run_netmiko2.py n'est pas dans le dossier scripts du workdir de votre container ? Une notion du cours vous aide à répondre à cette question.

10) A présent nous allons faire en sorte de binder un espace de travail (dossier) de notre machine host (machine virtuelle) avec le workdir de notre conteneur. L'intérêt est d'avoir un binding bidirectionnel entre le container et le host lorsque celui-ci est démarré.

a) Créez un dossier my_docker_bind_workspace dans le dossier TP_02 et créez un fichier test.txt (vide).

b) Exécutez votre container de la manière suivante : docker run -v $PWD:/tp02 -d tp02_image:1.0 sleep infinity . L'instruction sleep infinity permet de garder en "vie" notre conteneur car celui-ci ne fait rien (aucune instruction le maintient dans le dockerfile) et s'arrête une fois qu'il a fini son exécution.

c) Assurez-vous que votre conteneur est en vie en exécutant la commande : docker ps

d) Récupérez l'id du container et invoquez un bash : docker run exec -it <container_id> bash et exécutez la commande ls dans le workdir. Que voyez-vous ? A votre avis que s'est-il passé ?

e) Créez de nouveau un fichier nommé test2.txt dans le sous dossier my_docker_bind_workspace et répétez la question précédente (d). Que voyez-vous ?

f) Créez un fichier nommé test3 à l'intérieur de votre container grâce à la commande suivante : touch text3.txt. Que remarquez-vous au niveau de votre dossier my_docker_bind_workspace ? Quelle conclusion en tirez-vous ?

g) A votre avis, pourquoi à la question (d) le contenu du workdir de votre container ne dispose que du fichier text.txt et que l'ensemble des autres fichiers et dossiers ont disparu ? Si vous ne parvenez pas à répondre à cette question , passez aux questions suivantes et vous serez en mesure d'y répondre.

11) Nous allons à présent faire en sorte de sauvegarder les données de notre container dans un volume. Le volume permet d'avoir un binding entre le host (votre machine virtuelle) et le conteneur et aura pour effet de sauvegarder les données de notre container (ajout d'un nouveau fichier, modification de fichier etc).

    a) Créez un volume nommé tp02_volume en vous aidant de la commande docker volume

    b) Exécuter la commande docker inspect volume tp02_volume. Ou seront stockées les données du volume au niveau de votre machine virtuelle ?

    c) Affichez le contenu du volume

    d) Arrêtez votre container à l'aide de la commande : docker stop <container_id>

    e) Exécutez votre container de la manière suivante pour s'assurer de lui passer le volume en paramètre: docker run -v tp02_volume:/tp02 -d test sleep infinity. Affichez le contenu du volume (question c). Que remarquez-vous ? Que s'est-il passé ?

    f) Comment expliquez vous les différences entre la première méthode de binding (question 10, partiellement expliquée à la question 10, g) et la méthode de binding par volume persistant vu à la question (e).

    g) Répondez à la question 10) g) si vous n'avez pas pu y répondre plutôt

Il est important de comprendre les manipulations réalisées précédemment sur la partie docker et les volumes. Pour s'assurer d'avoir compris l'enjeux des volumes nous allons rapidement réaliser deux tests.

12) Créez une nouvelle image docker nommée tp02_image:2.0 à partir du dockerfile définis dans la section précédente: docker build -t tp02_image:2.0 .

13) Démarrez votre container : docker run -it tp02_image:2.0 sleep infinity

14) Invoquez un bash : docker container exec -it <container_id> bash

15) Créez un fichier dans le workdir : touch test.txt et quittez le bash

16) Redémarrez votre container : docker restart <container_id> et vérifiez si votre fichier text.txt est toujours présent. Comment expliquez-vous vos observations ?

17) Créez un nouveau volume nommé tp02_volumev2

18) Répétez les questions 13 à 16 en prenant soin d'attacher le nouveau volume précédemment créé au workdir du conteneur. Comment expliquez-vous vos observations ?