TP 03

Automatisation Réseau et conteneurisation Docker

5ETI : Module Automatisation Réseau et conteneurisation Docker

Amar KANTAS

Partie 1
Générer et déployer des configurations réseau avec Python et Nornir

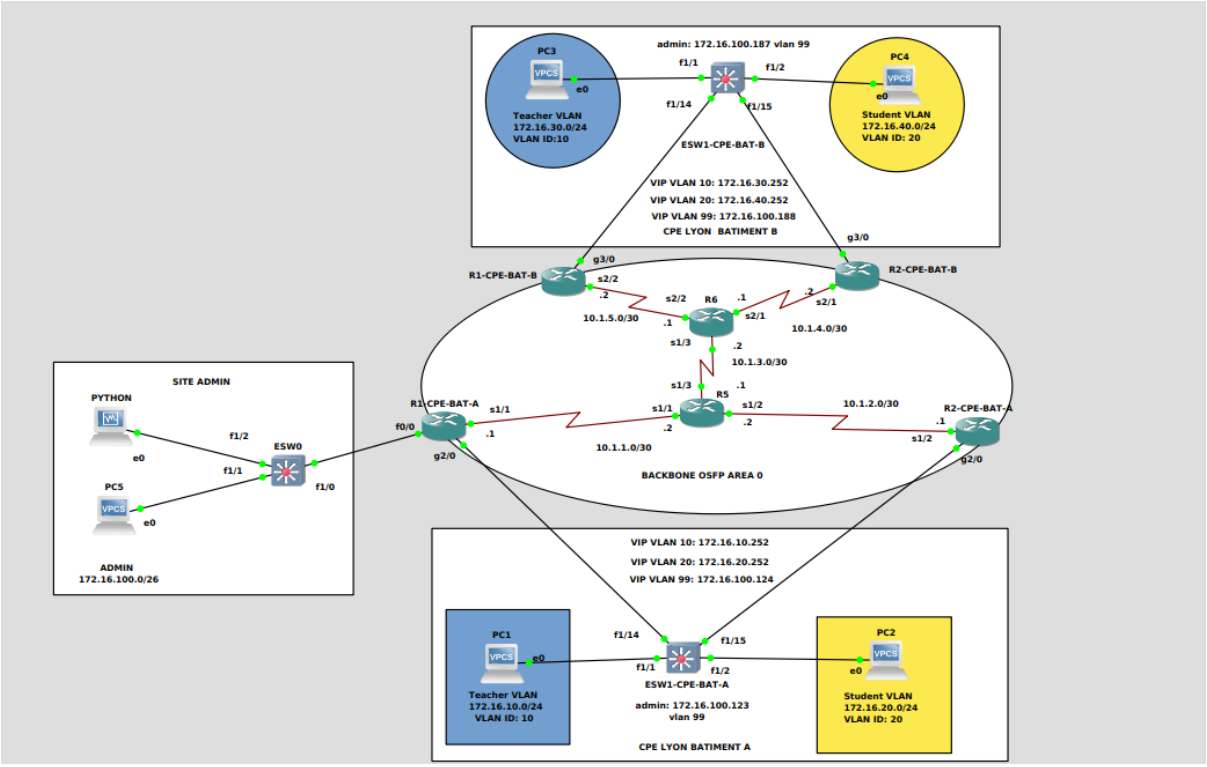
1-Automatiser le déploiement de configuration réseau sur des équipements Cisco

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 réseau sur les équipements nous utiliserons la librairie Nornir.

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 Batiment A:
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-LYON Batiment B:
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 non utilisé:
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 bâtiment.

1891.4 Liste des équipements

1.4.1 - liste des équipements réseau et de leur adresse ip d’admin

Hostname	Ip address	SITE
PC5	172.16.100.1	ADMIN
PYTHON	172.16.100.2	ADMIN
R1-CPE-BAT-A	f0/0 : 172.16.100.62 g2/0.99 : 172.16.100.125	CPE-LYON Batiment A
R2-CPE-BAT-A	g2/0.99 : 172.16.100.126	CPE-LYON Batiment A
ESW1-CPE-BAT-A	172.16.100.123 (vlan 99)	CPE-LYON Batiment A
R1-CPE-BAT-B	g3/0.99 : 172.16.100.189	CPE-LYON Batiment B
R2-CPE-BAT-B	g2/0.99 : 172.16.100.190	CPE-LYON Batiment B
ESW1-CPE-BAT-B	172.16.100.187 (vlan 99)	CPE-LYON Batiment B

1.4.2 - liste des équipements clients et de leur adresse ip

Hostname	Ip address	SITE
PC1	172.16.10.1 (vlan 10 - teacher)	CPE-LYON Batiment A
PC2	172.16.20.1 (vlan 20 - student)	CPE-LYON Batiment A
PC3	172.16.30.1 (vlan 10 - teacher)	CPE-LYON Batiment B
PC4	172.16.40.1 (vlan 20 - student)	CPE-LYON Batiment B

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 PC client est configuré
- Les liaisons point à point entre les routeurs du backbone OSPF sont également configurées
- Le routage OSPF est configuré pour le subnet ADMIN seulement, permettant à la machine Python d’accéder à l’ensemble des équipements des deux bâtiments via leur adresse ip d’admin
- La haute disponibilité VRRP (HA) des bâtiments A et B est configurée sur le subnet d’admin seulement

Travail demandé (résumé):

- Développez l’ensemble des scripts nécessaires pour générer la configuration réseau (Jinja2) du routage inter-vlan des Bâtiments A et B:
    - Développez l’ensemble des scripts nécessaires pour déployer automatiquement la configuration sur les équipements cibles (Routeurs et Switchs)
    - Testez le résultat de l’implémentation et faites vérifier par votre responsable pédagogique
- Développez l’ensemble des scripts nécessaire pour générer la configuration réseau (JInja2) du backbone OSPF (area 0) pour les subnets des vlans 10 (teacher) et 20 (student)
    - Développez l’ensemble des scripts nécessaire 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 nécessaire pour générer la configuration réseau (JInja2) du HA (VRRP) de chaque bâtiment pour les vlans 10 (teacher) et 20 (student)
    - Développez l’ensemble des scripts nécessaire 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 nécessaire pour générer des fichiers de backup de 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 PYTHON

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 .

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>

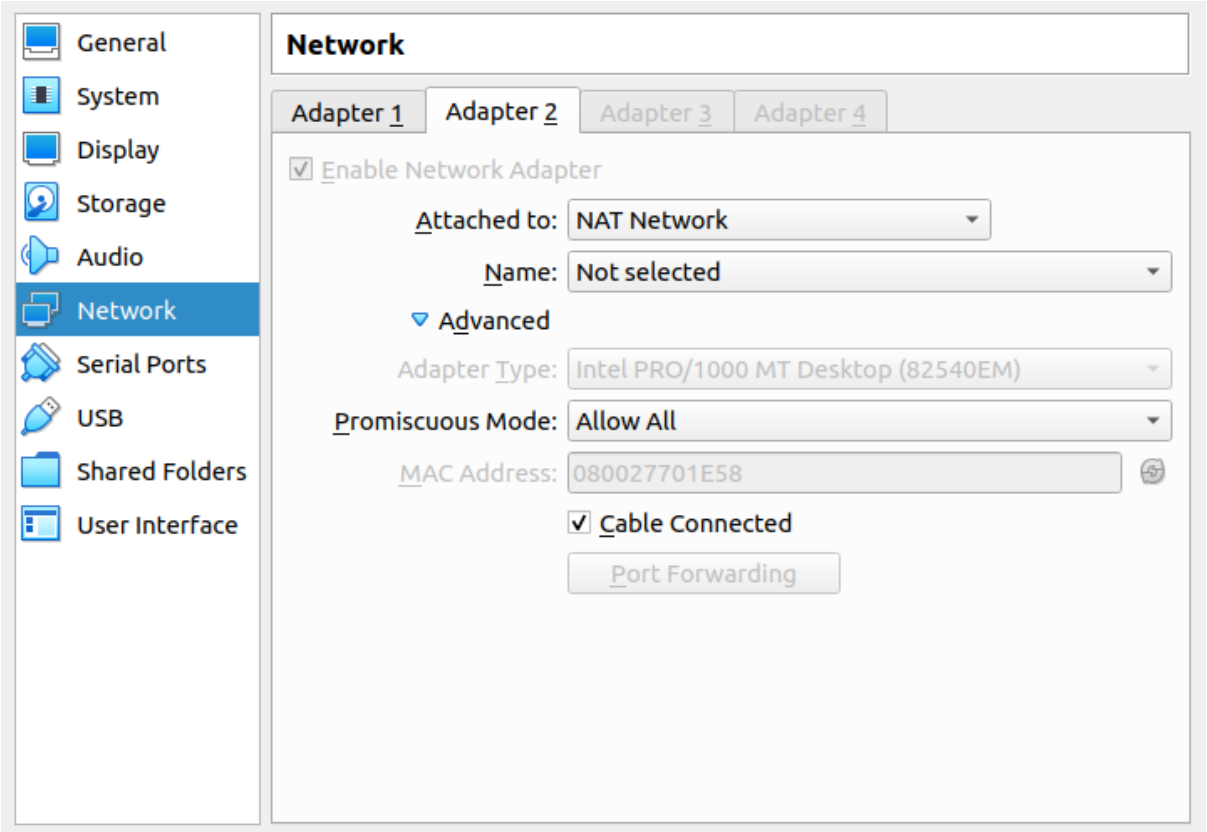
La machine virtuelle doit pouvoir joindre les équipements suivant :

Hostname	Ip address	SITE
PC5	172.16.100.1	ADMIN
PYTHON	172.16.100.2	ADMIN
R1-CPE-BAT-A	f0/0 : 172.16.100.62 g2/0.99 : 172.16.100.125	CPE-LYON Batiment A
R2-CPE-BAT-A	g2/0.99 : 172.16.100.126	CPE-LYON Batiment A
ESW1-CPE-BAT-A	172.16.100.123 (vlan 99)	CPE-LYON Batiment A
R1-CPE-BAT-B	g3/0.99 : 172.16.100.189	CPE-LYON Batiment B
R2-CPE-BAT-B	g2/0.99 : 172.16.100.190	CPE-LYON Batiment B
ESW1-CPE-BAT-B	172.16.100.187 (vlan 99)	CPE-LYON Batiment B

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.

- 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 :



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/TP03 à votre éditeur si ce n'est pas déjà fait.

Installez les dépendances pour utiliser les fonction du package Jinja2 dans votre workspace (TP03):

pipenv install Jinja2

Activer l'environnement virtuel en vous plaçant à la racine du projet (TP03) et en entrant : **pipenv shell**

Note: Vous pouvez directement utiliser le terminal depuis l'éditeur VsCode

Importez la librairie Jinja depuis le fichier scripts/create_config.py et définissez le dossier “templates” par défaut de Jinja dans une variable env:

```
from jinja2 import Template, Environment,  FileSystemLoader

env = Environment(loader=FileSystemLoader("templates"))
```

Exécutez le fichier create_config à l’aide de la commande : python -m scripts.create_config. Si vous avez une erreur de package qui s'affiche, sollicitez le responsable pédagogique.

1.8 -Génération automatique de la configuration à l’aide de Jinja2 (TP-01 et TP-02)

A ce stade vous avez pris connaissance de l’architecture et des éléments de configuration en place sur les équipements réseau 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 bâtiment A. (Pour rappel, le vlan d’admin (99) est déjà configuré sur les équipements.) Les templates JInja2 doivent être sauvegardés dans le dossier templates.

   L’objectif est que les machines du bâtiment A puissent communiquer entre elles. Nommez les fichiers de templates de la manière suivante:
   - vlan_router.j2
   - vlan_switch.j2
   - vrrp_router.j2

   Note: Appuyez-vous sur ce qui a été fait en TP-01/02 et en cours

2) Définissez les structures de données dans des fichiers JSON ou YAML (au choix) nécessaires pour remplir votre template JInja2 développé à la question 1. (Pour rappel, le vlan d'admin (99) est déjà configuré sur les équipements.) Les fichiers doivent être stockés dans le dossier data.

   Nommez les fichiers de data de la manière suivante:
   - R1_CPE_LYON_BAT_A (.json ou .yaml)
   - R2_CPE_LYON_BAT_A (.json ou .yaml)
   - ESW1_CPE_LYON_BAT_A (.json ou .yaml)

Voici les adresses ip à renseigner dans vos fichiers de données :

Hostname	Ip address
R1_CPE_LYON_BAT_A	g2/0.10 : ● IP: 172.16.10.253 - /24 ● VIP: 172.16.10.252 (backup priority 100) g2/0.20 : ● IP: 172.16.20.253 - /24 ● VIP: 172.16.20.252 (backup priority 100)
R2_CPE_LYON_BAT_A	g2/0.10 : ● IP: 172.16.10.254 - /24 ● VIP: 172.16.10.252 (master priority 110) g2/0.20 : ● IP: 172.16.20.254 - /24 ● VIP: 172.16.20.252 (master priority 110)

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 définis à la question 1 et 2. Écrivez votre code dans le fichier scripts/create_config.py

   Note: Vous pouvez récupérer la fonction render_network_config du TP-01/02.

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_config_cpe_lyon_batA() qui retournera la config générée pour chaque équipement.

4) Développez les fonctions vous permettant de sauvegarder automatiquement les configurations générées à la question 3 dans le dossier config de votre workspace (TP03).

   Nommez les fichiers de config de la manière suivant:
   - R1_CPE_LYON_BAT_A.conf
   - R2_CPE_LYON_BAT_A.conf
   - ESW1_CPE_LYON_BAT_A.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.

   Faites vérifier le résultat par votre responsable pédagogique

5) Répétez les questions 1 à 4 pour le bâtiment B en vous appuyant sur les données du tableau ci-dessous

Hostname	Ip address
R1_CPE_LYON_BAT_B	g2/0.10 : ● IP: 172.16.30.253 - /24 ● VIP: 172.16.30.252 (backup priority 100) g2/0.20 : ● IP: 172.16.40.253 - /24 ● VIP: 172.16.40.252 (backup priority 100)
R2_CPE_LYON_BAT_B	g2/0.10 : ● IP: 172.16.30.254 - /24 ● VIP: 172.16.30.252 (master priority 110)

	g2/0.20 : ● IP: 172.16.40.254 - /24 ● VIP: 172.16.40.252 (master priority 110)

Faites vérifier le résultat par votre responsable pédagogique

A ce stade le __main__ de votre script python ne doit contenir que les appels aux fonctions suivantes:

- create_config_cpe_lyon_batA()
- create_config_cpe_lyon_batB()
- 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 Nornir

Nornir est un framework d'automatisation réseau qui se différencie des autres librairies parcourues tout au long du TP02 (Paramiko, Netmiko, Napalm) car il fournit une abstraction pour l'inventaire (les hosts et les groupes d'hosts), l'exécution des tâches simultanées (nous n'avons pas besoin d'écrire nos propres threads) et se veut être flexible / extensible par l'intégration et l'écriture de plug-ins .

6) Installez nornir dans votre workspace (TP03): **pipenv install nornir**

7) Créez un fichier nommé run_nornir.py dans le dossier scripts de votre workspace (TP03) et importez le package nornir en en-tête du fichier :

```
from nornir import InitNornir
```

Nornir peut être initialisée de trois manières différentes :

1. Depuis un fichier de config YAML
2. Depuis le code python (manuellement)
3. Depuis un fichier de config YAML et le code python

Nous allons utiliser la première méthode , car c'est ce qui fait sa force par rapport aux autres librairies et c'est la méthode la plus "clean" pour initialiser l'environnement Nornir.

8) Créez un fichier config.yaml dans le dossier inventory de votre workspace et ajoutez la config suivante :

```yaml
inventory:
  plugin: SimpleInventory
  options:
    host_file: "inventory/hosts.yaml"
    group_file: "inventory/groups.yaml"
    defaults_file: "inventory/defaults.yaml"

runner:
  plugin: threaded
  options:
    num_workers: 20
```

9) Créez le fichier hosts.yaml dans le dossier inventory et ajoutez la structure de données suivante:

```yaml
R1-CPE-BAT-A:
  hostname: 172.16.100.125
  port: 22
  groups:
    - ios
  data: # Anything under this key is custom data
    device_name: R1-CPE-BAT-A
    device_type: router
    device_model: C7200
    locality: lyon
    building: A

R2-CPE-BAT-A:
  hostname: 172.16.100.126
  port: 22
  groups:
    - ios
  data: # Anything under this key is custom data
```

```
    device_name: R2-CPE-BAT-A
    device_type: router
    device_model: C7200
    locality: lyon
    building: A

ESW1-CPE-BAT-A:
  hostname: 172.16.100.123
  port: 22
  groups:
    - ios
  data: # Anything under this key is custom data
    device_name: ESW1-CPE-BAT-A
    device_type: router_switch
    device_model: C3725
    locality: lyon
    building: A

R1-CPE-BAT-B:
  hostname: 172.16.100.189
  port: 22
  groups:
    - ios
  data: # Anything under this key is custom data
    device_name: R1-CPE-BAT-B
    device_type: router
    device_model: C7200
    locality: lyon
    building: B

R2-CPE-BAT-B:
  hostname: 172.16.100.190
  port: 22
  groups:
    - ios
  data: # Anything under this key is custom data
    device_name: R2-CPE-BAT-B
    device_type: router
    device_model: C7200
    locality: lyon
```

```
    building: B
ESW1-CPE-BAT-B:
  hostname: 172.16.100.187
  port: 22
  groups:
    - ios
  data: # Anything under this key is custom data
    device_name: ESW1-CPE-BAT-B
    device_type: router_switch
    device_model: C3725
    locality: lyon
    building: B
```

10) Créez le fichier groups.yaml dans le dossier inventory et ajoutez la structure de données suivante:

```
ios:
  platform: ios
  data:
    vendor: Cisco
```

11) Créez le fichier defaults.yaml dans le dossier inventory et ajoutez la structure de données suivante:

```
  username: cisco
  password: cisco
```

12) Vous pouvez à présent initialiser Nornir dans le __main__ du script run_nornir.py de la manière suivante :

```
nr = InitNornir(config_file="inventory/config.yaml")
```

13) Quels sont les attributs de l'objet nr ? Pour connaître les attributs de l'objet nr utilisez la méthode __dict__ (utilisable sur tout objet python). Quel est le format de données de sortie ? Quelles sont les attributs de l'objet nr ? A votre avis lequel nous permettrait d'aller lire l'inventaire que nous avons précédemment défini (question 9) ?

14) Affichez l'attribut hosts de l'attribut que vous avez trouvé à la question 13. Quelles sont les données retournées ? Quel est le format de données retourné ?

15) Affichez la valeur du premier élément de l'objet à la question 14. Quelle est la valeur retournée ? Quel est le type de cette valeur (fonction type())?

16) Utilisez la méthode dir() sur l'objet de la question 15. Parmi les attributs affichés, lequel permet d'afficher l'adresse ip et le username / password du host en question? Affichez les valeurs de ces attributs dans la console. Depuis quel fichier ces données ont été récupérées ? Dans quelle section de ce fichier plus précisément?

17) Utilisez la méthode keys() sur l'objet de la question 15. Que voyez-vous ? Depuis quel fichier ces données ont été récupérées ? Dans quelle section de ce fichier plus précisément ? Ajoutez une nouvelle entrée à ce fichier, par exemple: room: 001 et exécuter de nouveau le script.

18) Affichez la valeur de "room" sur le terminal

19) Nornir nous fournit également la possibilité d'accéder aux données du fichier groups.yaml à l'aide de l'attribut groups de l'objet inventory. Affichez les groupes définis dans le fichier groups.yaml à l'aide du code suivant:

```
print(nr.inventory.groups)
```

20) Il est possible d'afficher les groupes rattaché à un host à l'aide de l'attribut groups de l'objet:

```
nr.inventory.hosts.get('R1-CPE-BAT-A').groups
```

21) Par exemple, si on veut afficher les keys définis dans le fichier groups.yaml du host RP1-CPE-BAT-A, il suffit d'exécuter :

```
nr.inventory.hosts.get('R1-CPE-BAT-A').groups[0].keys()
```

22) Affichez le vendor du host R1-CPE-BAT-A en partant du code de la question 21

23) Affichez le hostname (adresse ip) de chaque host définis dans le fichier hosts.yaml en utilisant l'objet nr définis à la question 12. Aidez-vous des questions précédentes.

24) Nornir nous donne la possibilité d'appliquer des filtres sur notre inventory pour récupérer par exemple un host à partir de son hostname par exemple. A l'aide du code suivant affichez la liste des hosts de type router.

```
nr.filter(key_example=".....").inventory.hosts.keys()
```

N'hésitez pas à vous aider de la doc : Nornir

25) Affichez à présent la liste des hosts de type router_switch.

Nous avons vu comment initialiser Nornir et comment parcourir notre inventory (hosts, groups, defaults) . L'inventory est une partie importante de Nornir car c'est ce qui fait l'une de ses forces par rapport aux autres librairies que nous avons vu.

Nous allons à présent aborder la notion de Tasks qui sont tout simplement des instructions à exécuter sur un groupe d'hosts (inventory).

Il est nécessaire d'avoir bien compris la partie du TP qui précède cette section pour aborder les tasks dans de bonnes conditions.

Si vous avez des questions ou un doute ou bien encore des zones d'ombres sur les notions précédentes n'hésitez pas à solliciter votre responsable pédagogique avant d'aborder la suite.

26) Importez les classes Task et Result et définissez une première task nommée hello_world

```
from nornir.core.task import Task, Result

def hello_world(task: Task) -> Result:
    return Result(
        host=task.host,
        result=f"{task.host.name} says hello world!"
    )
```

Pour exécuter la task hello_world il vous suffit de faire appel à la méthode run de l'objet nornir:

```
result = nr.run(task=hello_world)
```

Affichez le résultat : print(result)

27) Que retourne la variable result à la question 27 ? Aidez-vous de la méthode type()

28) Nornir fournit une fonction print_result qui permet d'afficher des logs sur les événements retournés par la task. Utilisez print_result pour afficher la variable result.

Vous devez installer le package nornir_utils : **pipenv install nornir_utils** et importer print_result depuis ce package:

```
from nornir_utils.plugins.functions import print_result
```

29) Après avoir affiché result à l'aide de la fonction print_result, qu'avez-vous remarqué ? Sur quel équipement la task s'est exécutée par défaut ?

30) Faites en sorte que la task hello_world s'exécute uniquement sur les équipements de type router_switch.

Comme décrit dans le cours et dans le TP, Nornir est un framework flexible et extensible. Il a la possibilité d'utiliser des plugins pour faire appels à certaines méthodes de la librairie Netmiko et Napalm (TP-02) pour interagir avec les équipements.

31) Installez les packages nornir_napalm et nornir_netmiko à votre environnement de développement (TP03):

pipenv install nornir_napalm
pipenv install nornir_netmiko

Importez les méthodes de chaque package:

```
from nornir_napalm.plugins.tasks import napalm_get, napalm_configure, napalm_cli

from nornir_netmiko.tasks import netmiko_send_config, netmiko_send_command, netmiko_save_config, netmiko_commit
```

32) Développez une task permettant d'afficher l'état des interfaces de chaque **routeur** à l'aide de la fonction napalm_cli.

```
result =  nr.run(task=....., commands=["....."])
print_result(result)
```

33) Développez une task permettant d'afficher la table arp de chaque **switch** référencé dans l'inventory. Utilisez la fonction napalm_get pour cela.

```
result =  nr.run(task=....., getters=["....."])
print_result(result)
```

Liste des getters napalm : Napalm doc

34) Développez deux tasks permettant de créer une interface de loopback sur les routeurs R1 et R2 du bâtiment A . (Une task par routeur) à l'aide de la task napalm_configure

interface loopback 1 R1: 1.1.1.1 255.255.255.255
interface loopback 1 R2: 2.2.2.2 255.255.255.255

```
result = nr.run(task=....., configuration=".....”])
print_result(result)

result = nr.run(task=....., configuration=".....”])
print_result(result)
```

35) Développez une fonction contenant une task permettant de sauvegarder la running-config sur les équipements (routeurs / switchs) depuis napalm_cli.

36) Développez une fonction permettant d'afficher l'état des interfaces de chaque **routeur** à l'aide de la task netmiko_send_command.

```
result = nr.run(task=....., command_string="....")
print_result(result)
```

37) Développez une fonction permettant de créer une interface de loopback 2 sur les routeurs R1 et R2 du bâtiment A . (Une task par routeur) en utilisant la task netmiko_send_config

interface loopback 2 R1: 1.1.1.2 255.255.255.255
interface loopback 2 R2: 2.2.2.3 255.255.255.255

```
result = nr.run(task=....., config_commands=["....",”....”])
print_result(result)
```

38) Développez une fonction contenant une task permettant de sauvegarder la running-config de la question précédente à l'aide de la task netmiko_save_config

A ce stade du TP vous êtes en mesure d'utiliser Nornir pour parcourir les hosts de l'inventory et appliquer des filtres de recherche afin d'exécuter des tasks en fonction du type de host ou de l'adresse ip etc.

Vous êtes également en capacité d'interagir avec les équipements en lecture et écriture à l'aide des plugins nornir_netmiko et nornir_napalm.

39) Reprenez la première partie du TP afin de déployer les configurations générées sur les équipements du bâtiment A et B (vlan, routage inter-vlan, vrrp pour les vlans 10 et 20. Vous avez le choix d'utiliser nornir_netmiko ou nornir_napalm (ou les deux) pour le déploiement de la configuration. Pensez à sauvegarder automatiquement (depuis nornir) vos configurations après le déploiement. Aidez-vous de la doc de nornir_napalm et nornir_netmiko pour le déploiement de config via un fichier de conf.

a) Vérifiez que la communication entre le vlan 10 et 20 du bâtiment A est fonctionnelle après déploiement (même chose pour le bâtiment B)

b) Vérifier l'état du HA entre les routeurs de chaque bâtiment : show vrrp brief. Le routeur R1 de chaque bâtiment doit être le backup et R2 doit être le master.

c) Testez manuellement votre HA vrrp sur chaque bâtiment

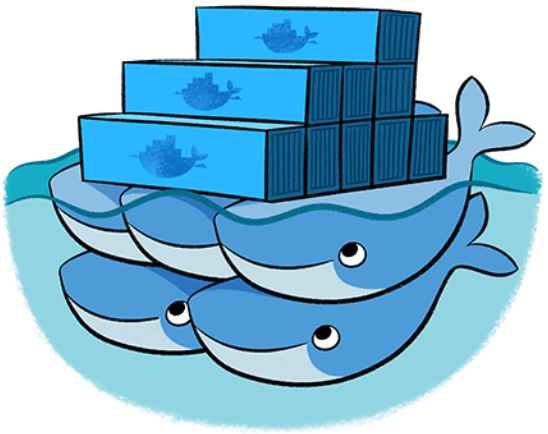
Faites vérifier par votre responsable pédagogique

d) Testez automatiquement (via nornir) votre HA vrrp sur chaque bâtiment

Faites vérifier par votre responsable pédagogique

40) Créez une task à l'aide de nornir_netmiko ou nornir_napalm permettant d'annoncer les subnets des vlans 10 et 20 du batiment A et B sur le backbone OSPF. Les vlans de chaque bâtiment pourront ainsi communiquer ensemble.. Assurez-vous de générer automatiquement la configuration OSPF à l'aide de Jinja2 (Voir TP02).

Partie 2
Manipulation des différents réseaux Docker et création d'un cluster Docker Swarm



Le but de cette deuxième partie du TP est de créer un ensemble de conteneurs attaché à différents types de réseau docker (driver) et d'observer les différences entre chaque type de réseau. L'utilisation des différents réseaux docker nous permettra de tester plusieurs scénarios comme : la communication entre différents conteneurs d'une même machine host, l'isolation des communications inter-conteneur et la communication entre plusieurs conteneurs hébergés sur différents hosts.

Dans un deuxième temps nous utiliserons docker swarm pour créer un cluster de conteneurs composé d'un manager et de deux workers. Nous effectuerons plusieurs tests afin d'observer le comportement du cluster et des services qui y seront déployés.

2.0 - Création de l'environnement de travail docker

Pour cette partie du TP nous n'aurons pas besoin d'utiliser GNS3 et virtualbox. Nous allons utiliser une sandbox gratuite, accessible sur internet et qui nous permettra de créer des instances (VMs) sur lesquelles nous allons réaliser nos actions.

La sandbox s'intitule Play With Docker (PWD) et est accessible depuis l'URL : https://labs.play-with-docker.com/. Il est nécessaire d'avoir un compte Docker hub pour utiliser la plateforme. Vous pouvez facilement créer votre compte à l'adresse suivante : https://hub.docker.com (pour les besoins du TP évitez de mettre un mot de passe personnel, utilisez plutôt un nouveau mot de passe random).

Attention : Comme il s'agit d'un lab en accès libre et gratuit, il se peut que vous rencontriez des instabilités, lenteurs ou des erreurs. Si les problèmes persistent sur vos instances je vous invite à faire le TP en utilisant VirtualBox. Référez-vous à la section en annexe **Création de l'environnement de travail Docker Swarm avec Virtualbox**.

1) Connectez-vous au lab à l'adresse https://labs.play-with-docker.com et créez votre première instance en cliquant sur "ADD NEW INSTANCE". Par défaut docker est déjà installé sur l'instance, pour vérifier son bon fonctionnement exécuter la commande : docker –version

   Comme vous pouvez le voir vous avez directement la possibilité de travailler sur la console intégrée mais celle-ci peut s'avérer moins pratique dans certains cas, par exemple il n'est pas possible de copier / coller sur le terminal directement.

2) Il est possible de se connecter en SSH à notre instance. Pour cela il vous suffit simplement de copier l'adresse SSH et de la coller dans un terminal de votre choix sur votre machine. Si vous obtenez une erreur de type Permission denied (publickey) il vous suffira de générer une paire de clé ssh sur votre machine à l'aide de la commande ssh-keygen (en laissant la passphrase vide) et de relancer la commande ssh.

3) Maintenant que vous avez votre environnement de travail fonctionnel, nous allons créer un conteneur NGINX pour voir comment tester la redirection des ports avec les instances du lab. Pour cela, créez un conteneur ayant pour nom nginx-01, utilisant l'image nginx et écoutant sur le port 8080. (Par défaut NGINX écoute sur le port 80). Aidez-vous de la documentation suivante : https://hub.docker.com/_/nginx.

    a) Sur la page web de votre instance PWD, cliquez sur le bouton OPEN PORT et renseignez le port 8080. Vous serez automatiquement redirigé vers votre instance sur le port 8080 et si l'exécution de votre conteneur s'est correctement déroulée, vous obtenez à l'écran le message "welcome to nginx" .

Chaque fois que vous voudrez consulter au travers de HTTP les applications conteneurisées dans vos conteneurs de votre instance vous devrez toujours cliquer sur le bouton OPEN PORT de votre instance et renseigner le port souhaité.

Si vous utilisez une VM standard , rendez-vous simplement sur l'adresse localhost:PORT dans un navigateur.

2.1 - Manipuler les réseaux avec Docker Network

Dans cette partie de TP nous allons explorer et manipuler les réseaux au sein des conteneurs docker. L'objectif est d'observer les différents types de network docker (driver) et créer nos propres network avec leur propres configurations (subnets etc).

4) Invoquez un bash au sein de votre conteneur nginx-01 créé précédemment et afficher l'adresse ip de celui-ci à l'aide de la commande ifconfig. Quel est le résultat de la commande ?

5) Faites le nécessaire pour que l'exécution de la commande ifconfig soit fonctionnelle au sein de votre conteneur.

6) Affichez l'adresse ip du conteneur. Exécutez un ping vers google.fr. Faites le nécessaire afin que la commande ping soit également reconnue et relancer votre ping. Le ping est-il fonctionnel ? Pourquoi ? Aidez-vous de la commande docker inspect <container_id> pour argumenter votre réponse.

7) Par la suite nous serons amenés à utiliser la même image pour un deuxième container NGINX. Afin d'éviter de toujours devoir installer manuellement les packages nécessaires à la commande ifconfig et ping au sein du container NGINX, quelle serait la solution afin d'avoir une image NGINX disposant déjà de ces dépendances là ?

8) Faites le nécessaire à la question 7 et nommez votre nouvelle image : nginx-cpe

9) Arrêtez votre conteneur nginx-01 et exécutez un nouveau conteneur ayant pour nom nginx-cpe-01 et utilisant cette fois-ci l'image nginx-cpe.

10) Invoquez un bash sur le conteneur précédemment créé et exécutez les commandes ifconfig et ping 8.8.8.8. Les commandes doivent directement être fonctionnelles sans aucune intervention supplémentaire de votre part.

11) Créez un deuxième conteneur nginx-cpe-02 utilisant l'image nginx-cpe et écoutant sur le port 8081. Invoquez un bash et comparez l'adresse ip avec le précédent conteneur nginx-cpe-01. Quelles sont vos observations ?

12) Sur quel réseau (subnet) se trouvent vos deux conteneurs ? Aidez-vous de la commande docker inspect <container_id> et de la commande docker inspect network <network_name>.

13) Quelle est la passerelle par défaut (gateway) configurée sur vos conteneurs docker ?

14) Réalisez un ping entre les deux conteneurs. Quel est le résultat ? Expliquez.

15) Créez un nouveau réseau nommé cpe-bridge-network et qui a pour subnet le réseau 192.168.1.0/24. Aidez-vous de la commande docker network create et de la documentation en ligne ainsi que du cours.

    Liens de la doc : Docker network

    Faites en sorte que votre réseau soit “**attachable**” à un conteneur (voir les options dans la doc)

16) Connectez le réseau cpe-bridge-network à votre conteneur nginx-cpe-01. Aidez-vous de la commande docker network connect et de la documentation Docker network connect.

17) Invoquez un bash sur le conteneur nginx-cpe-01 et exécutez la commande ifconfig. Qu’observez-vous ?

18) Exécutez la commande docker container inspect nginx-cpe-01, que remarquez-vous dans la section NetworkSettings ?

19) Déconnectez le réseau par défaut “bridge” attaché au conteneur nginx-cpe-01 et exécutez de nouveau un ifconfig au sein du conteneur. Que remarquez-vous ?

20) Tentez de ping le conteneur nginx-cpe-02. Quel est le résultat ? Pourquoi ?

21) Faites en sorte que le ping entre le conteneur nginx-cpe-01 et nginx-cpe-02 soit de nouveau fonctionnel au sein du réseau cpe-bridge-network. Aidez-vous des questions précédentes pour les appliquer au container nginx-cpe-02. Indiquez les différentes étapes dans votre réponse.

22) Affichez la liste des networks à l’aide de la commande docker network ls. Quels sont les différents drivers affichés ?

23) Exécutez la commande docker network inspect sur le network nommé none. Que remarquez-vous ?

24) Déconnectez le network cpe-bridge-network de vos deux conteneurs à l'aide de la commande docker network disconnect. Connectez ensuite le network none à vos deux conteneurs à l'aide de la commande docker network connect.

25) Exécutez la commande docker container inspect sur vos deux conteneurs. Que remarquez-vous dans la section network ?

26) Invoquez un bash sur le conteneur nginx-cpe-01 et faites un ifconfig. Qu'observez-vous ?

27) Tentez de lancer un ping vers google.fr / 8.8.8.8. Quel est le résultat de la commande ? Quelles sont vos conclusions ?

Il nous reste encore un type de network à explorer qui est l'overlay. Le network overlay permet la communication inter-container situés sur différents hosts. Mais pour pouvoir manipuler ce type de network il nous faut initialiser le cluster à l'aide de docker swarm. Nous allons donc explorer cela à la section suivante.

2.3 - Découvrir et manipuler docker swarm

Dans cette section nous allons découvrir et manipuler le service d'orchestration docker swarm. A l'aide de la plateforme play with docker (PWD) nous serons en capacité de créer un cluster docker et de déployer nos différents services (containers) au sein du cluster et des workers. Avant de démarrer, fermer votre session PWD afin de partir sur de nouvelles instances.

Si vous utilisez virtualbox et non pas PWD pour le TP, je vous invite à créer au moins 2 machines virtuelles (un manager et un worker).

28) Créez une instance et connectez-vous en SSH sur celle-ci. Nous allons initier le cluster au sein de cette instance à l'aide de la commande : docker swarm init –advertise-addr <instance_ip_address>.

    Enregistrez l'output de la commande docker swarm init. Elle nous servira pour la suite.

29) Créez une deuxième instance et exécutez la commande en sortie de la question 28.

docker swarm join --token <swarm_token><advertised_ip>:<advertised_port>

30) Vérifiez que votre deuxième instance (worker) à bien rejoint votre cluster à l'aide de la commande : docker node ls (à exécuter sur le manager)

31) Reprenez votre fichier dockerfile à la question 8 de la section précédente du TP. Buildez une image et nommez la nginx-cpe (sur le manager).

32) Vérifiez sur le manager que votre image a bien été créée à l'aide de la commande docker images.

33) Exécuter la commande docker images sur votre worker. Vous devriez avoir aucune image docker sur cette instance.

34) Créez un conteneur sur votre manager avec les instructions suivantes:
- Nom du conteneur : nginx-cpe-01
- Image du conteneur: nginx-cpe
- Port mapping: 8080:80

35) Répétez la question 31 sur votre worker et créez un conteneur sur votre worker en reprenant les instructions de suivantes :
- Nom du conteneur : nginx-cpe-02
- Image du conteneur: nginx-cpe
- Port mapping: 8081:80

36) Tentez un ping entre le conteneur nginx-cpe-01 du host manager et le conteneur nginx-cpe-02 du host worker. Quel est le résultat ? Pourquoi?

37) D'après vous et les éléments vu en cours, quelle est la solution à mettre en place pour que le ping soit fonctionnelle à la question 36 ?

38) Créez un nouveau network de type overlay selon les instructions suivantes :
    - Nom du network: cpe-overlay-network
    - Créez ce network sur le manager uniquement
    - Le réseau doit être attachable

39) Connectez le network cpe-overlay-network à votre conteneur nginx-cpe-01 (manager) et déconnectez ensuite le network bridge

40) Faites la même chose (question 39) sur le worker

41) Sur le worker l'attachement du network doit correctement se dérouler et pourtant vous n'avez pas créer le network sur la machine comme mentionnée à la question 38. Expliquez cela en vous basant sur les explications données dans le cours sur la partie docker-swarm.

42) Réalisez un ping entre le conteneur nginx-cpe-01 (manager) et le conteneur nginx-cpe-02 (worker). Commentez et expliquez le résultat.

Nous allons à présent créer et déployer des services docker (conteneurs) à l'aide de notre cluster swarm. L'objectif est de déployer des services sur l'ensemble des workers de notre cluster et d'observer également les fonctions de load-balancer délivrées par docker-swarm.

2.4 - Déployer des services à l'aide de Docker-swarm

Dans cette partie nous allons créer une API web permettant simplement d'afficher un helloworld lorsqu'on interroge le service web depuis un navigateur par exemple.

43) Placez-vous sur votre manager et créez un dossier cpe_api à la racine de votre home.

44) Placez-vous à la racine du dossier cpe_api et installez les dépendances python suivantes:

```
pipenv install flask
```

45) Activez votre environnement virtuel à l'aide de la commande pipenv shell.

46) Créez un fichier api.py à la racine du dossier cpe_api et insérez le code suivant:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

47) Exportez les variables d'environnements suivantes dans votre terminal :

```
export FLASK_ENV=dev
export FLASK_APP=api.py
```

Ces variables d'environnement vous permettent d'exécuter le serveur de dev embarqué par le framework Flask pour exécuter votre serveur API en local.

48) Exécutez ensuite la commande suivante pour lancer le serveur de développement :

```
flask run
```

49) Ouvrez un navigateur et rendez-vous à l'adresse e votre api : localhost:5000. Vous devriez obtenir "hello world" à l'écran. Cela signifie que votre API est fonctionnelle et qu'elle répond correctement.

50) Pour le moment notre API retourne du HTML lorsqu'on l'interroge. Si on veut être plus réaliste et respecter les "standards" nous allons faire en sorte que notre API retourne de la donnée structurée en JSON lorsqu'on se rend à l'adresse localhost:5000.

a) Importez le package jsonify depuis le module flask dans votre fichier api.py

b) Modifiez la réponse de votre fonction hello_world en utilisant la fonction jsonify :

```
return jsonify(message="Hello world")
```

c) Redémarrez le serveur pour que vos modifications soient prises en compte et vérifiez le résultat à l'adresse localhost:5000.

PS: Si vous souhaitez que vos modifications soient prise en compte en live (live reload) il suffit simplement d'ajouter la variable d'environnement FLASK_DEBUG=TRUE

51) Maintenant que notre API est prête et fonctionnelle, nous allons la déployer en tant que service (swarm) au travers de notre cluster. Les services swarm sont en réalité des containers qui sont déployés au travers du cluster et des workers. Vous l'aurez donc compris que pour que notre API soit déployée il faut que celle-ci soit conteneurisée.

a) Générez à l'aide de la commande suivante le requirements.txt du projet . Ce fichier contient les dépendances python qui seront installées au sein de votre conteneur pour exécuter votre API.

```
pipenv lock -r > requirements.txt
```

b) Créez et complétez le Dockerfile permettant le build de l'image contenant notre api en vous basant sur l'exemple fournis :

```
FROM python:3.8-slim-buster

WORKDIR ........ À DÉFINIR ..........

ENV FLASK_APP=api.py
ENV FLASK_ENV=development

COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

COPY ............ À DÉFINIR ..................

CMD ["......", "....", "--host=0.0.0.0"]
```

c) Buildez l'image et nommez la cpe_api_image.

d) Testez l'image en exécutant un conteneur avec les instructions suivante:

- port : 8080:5000
- nom du container : cpe_api_01

e) Testez que votre API conteneurisée répond correctement

52) Nous allons à présent déployer notre API sous forme de service swarm. Pour cela, commencez par arrêter votre conteneur cpe_api_01.

53) Pour les besoins du TP qui vont suivre , vous allez devoir créer un compte docker hub (si ce n'est pas déjà fait). Docker hub est à l'image de ce qu'est Github / Gitlab pour la gestion du code source, sauf que dans le cas de docker notre gestion se porte sur des images.

a) Créez un compte sur https://hub.docker.com et connectez-vous à votre espace

b) Créez ensuite un nouveau repos avec pour nom : cpe_api

c) Placez-vous sur le manager et connectez votre process docker à votre repos docker hub à l'aide de la commande docker login.

L'objectif du docker-hub est de pouvoir déposer (push) et / ou récupérer (pull) des images qu'on a build et de maintenir un versionning.

54) Rejouez la question 51-b) afin de builder une nouvelle image de votre API en respectant les spécification suivantes:

- nom de l'image avec la version : <dockerhub_username>/cpe_api:1.0

55) Poussez l'image générée dans votre repos à l'aide de la commande : docker push <dockerhub_username>/cpe_api:1.0

56) Vérifiez que votre image docker est bien présente dans votre repository. Si vous souhaitez importer l'image depuis une autre machine manuellement vous pouvez exécuter la commande : docker pull <dockerhub_username>/cpe_api:1.0 (à condition que sur cette machine vous ayez connecté votre compte dockerhub).

57) Créez à l'aide du cours et de la documentation un service swarm (sur votre manager) avec les informations suivantes:

- Nom du service: cpe_api
- Image du service : <dockerhub_username>/cpe_api:1.0
- Port du service : 8080
- Nombre d'instance : 1 seule pour le moment
- passez l'option : –with-registry-auth (recherchez sur la doc ce qu'elle signifie et expliquez dans quel cas cela servira au cluster swarm)

58) Vérifiez que votre service est correctement créé à l'aide de la commande : docker service ls

59) Depuis le navigateur de votre manager rendez-vous à l'adresse localhost:8080. Qu'observez-vous ?

60) Depuis le worker-01, exécutez la commande : docker ps. Que voyez-vous ?

61) Depuis le navigateur de votre worker-01 rendez-vous à l'adresse localhost:8080. Qu'observez-vous ? Comment l'expliquez ? Détaillez votre réponse à l'aide des notions abordées en cours

62) Depuis le worker-02, exécutez la commande : docker ps. Que voyez-vous ?

63) Depuis le navigateur de votre worker-02 rendez-vous à l'adresse localhost:8080. Qu'observez-vous ? Comment l'expliquez ? Détaillez votre réponse avec les notions abordées en cours

64) Trouvez sur quelle machine (manager / worker) est exécuté votre conteneur à l'aide de la commande docker ps et arrêtez le à l'aide de la commande docker container stop <container_id>. Rendez-vous ensuite à l'adresse localhost:8080. Comment expliquer que votre service répond toujours ? (Notion abordée en cours)

65) Modifiez votre service swarm afin que celui-ci exécute 10 instances de votre API au travers du cluster. Aidez-vous documentation en ligne :

https://docs.docker.com/engine/reference/commandline/service_scale/

66) Vérifiez depuis le manager que vos instances sont correctement créées à l'aide de la commande docker service ls et docker ps.

67) Sur quel workers les instances ont été déployées ? Vérifiez à l'aide de la commande docker ps sur chaque worker.

68) Comment les workers arrivent à récupérer l'image docker de votre API ? Donnez deux raisons pour lesquelles cela est possible.

69) Mettez à jour le code de l'API afin que celle-ci affiche l'output suivant :

{"message":"Hello World", version:"2.0"}

a) Buildez une nouvelle image: <dockerhub_username>/cpe_api_image avec pour version 2.0

```
docker build -t <dockerhub_username>/cpe_api_image:2.0 .
```

b) Sauvegardez votre image sur votre repos dockerhub

c) Mettez à jour votre service swarm avec la nouvelle image générée précédemment à l'aide de la commande suivante :

```
docker service update cpe_api -image <dockerhub_username>/cpe_api_image:2.0
```

d) Les instances se mettent à jour une par une, à votre avis pourquoi ?

70) Exécutez le conteneur suivant depuis votre manager swarm, cela vous permettra d'avoir une vue d'ensemble de votre cluster et des instances actives. Ouvrez ensuite un navigateur à l'adresse localhost:5000 (depuis votre manager)

```
docker run -it -d -p 5000:8080 -v /var/run/docker.sock:/var/run/docker.sock
dockersamples/visualizer
```

source : https://github.com/dockersamples/docker-swarm-visualizer

71) Modifiez le nombre d'instances de votre service afin d'observer les changements sur l'interface du visualizer de votre cluster.

Annexe - Création de l'environnement de travail docker swarm avec Virtualbox

Pour travailler dans de bonne condition l'idéal serait de créer 3 machines virtuelles pour pouvoir simuler un cluster docker swarm dans son ensemble. Une machine fera office de manager et les deux autres seront des workers pour l'exécution des services définis par le manager. Si votre machine ne peut pas supporter 3 machines virtuelles vous pouvez vous limiter à deux machines dans le but d'avoir au moins un manager et un worker swarm.

Il est également nécessaire que les trois machines virtuelles puissent communiquer entre elles, pour cela nous allons devoir faire quelques configurations sur virtualbox et les machines virtuelles.

1. Créez trois machines virtuelles sous Ubuntu ou clonez votre machine virtuelle d'automatisation réseau python avec laquelle vous avez travaillé tout au long des séances de TP.

2. Chaque machine virtuelle doit disposer de deux interfaces:
    a. Une interface interne (assurez-vous de bien cocher la case "câble branché") pour les communications entre les machines virtuelles.

       L'interface interne est connectée à un réseau virtuel interne à virtualbox , il simule un switch de niv2 ou chaque VM possédant une interface interne est connectée et peut directement être joignable depuis une autre machine connecté à ce réseau interne.

    b. Une interface NAT pour les communications vers l'extérieur (Internet)

3. Assurez-vous que chaque machine dispose d'une adresse ip sur l'interface interne (qui est attribuée automatiquement) . Il est possible que plusieurs VMs possèdent la même IP sur ce réseau interne et le cas échéant modifiez l'adresse ip de la VM pour que celle-ci possède une adresse ip unique dans le réseau interne. Testez à l'aide d'un Ping que vos trois machines communiquent bien entre elles.

4. Assurez-vous que chaque machine dispose de docker en exécutant la commande : docker –version. Sinon installez sur chaque machine docker en suivant le lien d'installation suivant : https://docs.docker.com/engine/install/ubuntu/