**Damien OLLIER 5ETI**

**Compte rendu TP : Automatisation Réseau et conteneurisation Docker**

**TP1 :** Générer des configurations réseau avec Python

Diagram

Description automatically generated

Figure 1: Architecture du réseau

**Question 1°)**

Pour que les équipements du vlan 10 et 20 puissent communiquer, il faut configurer le routeur R2 et le switch ESW2. Pour cela, on doit défini le nom de l’interface, une description, son adresse IP, son masque de sous-réseau et son vlan id.

**Question 2°)**

Les variable Jinja sont les mêmes que celles présentes dans le fichier de configuration Json. Il y a donc le nom de l’interface, une description, son adresse IP, son masque de sous-réseau et vlan ID.

**Question 3°)**

Les structures de données R2.JSON et ESW2.JSON sont disponible dans le dossier data du TP1-DEVOPS-MAIN.

**Question 4°)**

def load\_json\_data\_from\_file(file\_path):

    with open(file\_path) as json\_file:

            data = json.load(json\_file)

            return(data)

Pour appeler la fonction

load\_json\_data\_from\_file("data/R2.json")

load\_json\_data\_from\_file("data/ESW2.json")

**Question 5°)**

**Résultat de la console**

python3 \_\_main\_\_.py

Traceback (most recent call last):

File "/home/cpe/workspace/TP-01/scripts/\_\_main\_\_.py", line 36, in <module>

f = open('./data/ESW2.json') # Opening JSON file

FileNotFoundError: [Errno 2] No such file or directory: './data/ESW2.json'

Erreur fichier inexsitant

**Le script va générer une erreur car le chemin du fichier est introuvable. (FileNotFoundError: [Errno 2] No such file or directory.)**

**Question 6°)**

Il faut créer une exception pour lever l’erreur de la question précédente.

def load\_json\_data\_from\_file(file\_path):

    try:

        with open(file\_path) as json\_file:

            data = json.load(json\_file)

            return(data)

    except FileNotFoundError:

        print(f"ERREUR | Le fichier {file\_path} n'existe pas.")

    pass

**Question 7°)**

Le template R2.j2 est disponible dans le dossier templates du TP1-DEVOPS-MAIN.

**Question 8°)**

Le template ESW2.j2 est disponible dans le dossier templates du TP1-DEVOPS-MAIN.

**Question 9°)**

Définition du répertoire par défaut contenant les Jinja :

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

**Question 10°)**

def render\_network\_config(template\_name, data):

    template = env.get\_template(template\_name)

    return(template.render(data))

**Question 11°)**

def save\_built\_config(file\_name, data):

    fichier = open(f"config/{file\_name}", "w")

    fichier.write(data)

    fichier.close()

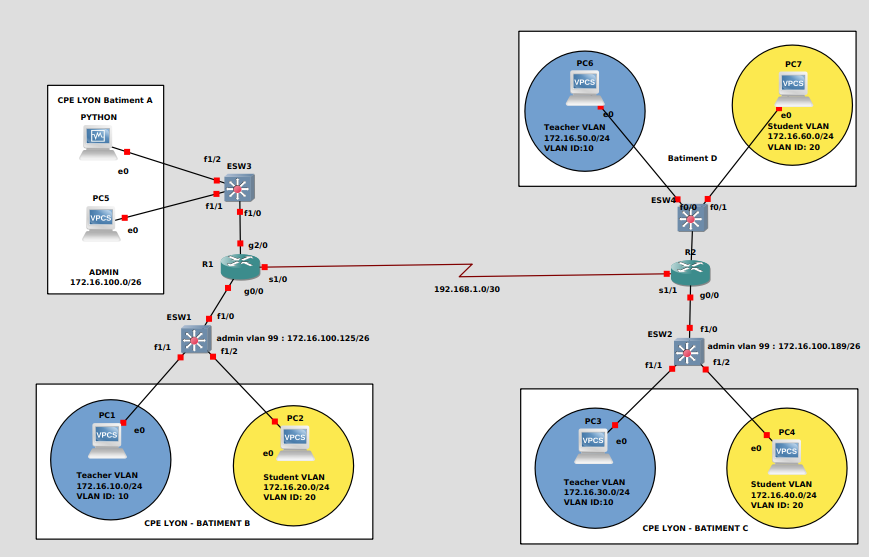
    pass

**Question 12°)**

J’ai copié les fichiers configurations dans le routeur et le switch et fait un ping pour tester si les deux sous réseau étaient joignable.

**Ajout d’un nouveau bâtiment D à l’architecture**

**Question 13°)**



**Question 16°)**

def load\_yaml\_data\_from\_file(file\_path):

    try:

        with open(file\_path) as yaml\_file:

            data = yaml.safe\_load(yaml\_file)

    except FileNotFoundError:

        print(f"ERREUR | Le fichier {file\_path} n'existe pas.")

    return data

**Question 17°)**

Les données sont au format JSON

**Question 18°)**

Il n’est pas nécessaire de modifier les templates car ils fonctionnent pour des fichiers JSON et YAML.

**Question 20°)**

De même que la question 12, j’ai copié le fichiers configurations sur le routeur et fait un ping pour tester si les deux sous réseau étaient joignable.

**Partie 2 Prise en main Docker**

Le but de cette deuxième partie du TP est de se familiariser à l’utilisation de conteneurs pour le développement d'applications ou l’installation d'applications métier. L’objectif est de profiter de la puissance et la flexibilité de Docker pour déployer des applications dans les mêmes conditions d'exécution utilisées pendant la phase de développement.

**Question 3°)**

Après avoir bien configuré docker sur la vm, on la redémarre et on run « docker run hello-word ».

$ docker run hello-world

Hello from Docker.

This message shows that your installation appears to be working correctly.

**Question 8°)**

**a°)**

Avec les données suivantes :

**Nom du container: cpe\_lyon\_mysql**

**Port d’écoute: 3306**

**Mode: detached**

**Mot de passe: cpelyon123**

On créer notre premier conteneur MySQL avec la commande suivante :

**docker run --name cpe\_lyon\_mysql -e MYSQL\_ROOT\_PASSWORD=cpelyon123 --detach -p 3306:3306 -d mysql:latest**

**b°)**

Commande : **docker stop f8517a670a56,** avec f8517a670a56 l’id du conteneur.

**c°)**

Commande : **docker start f8517a670a56,** avec f8517a670a56 l’id du conteneur.

**d°)**

On créer notre deuxième conteneur MySQL avec la commande suivante :

Nom du container: cpe\_lyon\_phpmyadmin

Port d’écoute: 8080

Mode: detached

Link: linkez votre instance à votre container MySQL

Commande : **docker run --name cpe\_lyon\_phpmyadmin -e MYSQL\_ROOT\_PASSWORD=cpelyon123 --detach -p 8080:80 --link cpe\_lyon\_mysql -d mysql:latest**

Commande : **docker run --name cpe\_lyon\_phpmyadmin -e MYSQL\_ROOT\_PASSWORD=cpelyon123 -d --link cpe\_lyon\_mysql:db mysql:latest phpmyadmin**

Commande **: docker run --name cpe\_lyon\_phpmyadmin3 -p 8080:80 --link cpe\_lyon\_mysql:db -d phpmyadmin**

**Question 10°)**

**d°)**

Text

Description automatically generatedQuand on tape «Docker images »

**f°)**

Ça ne marche pas car la web app est dans un conteneur

**TP2 :** Générer des configurations réseau avec Python

Diagram

Description automatically generated

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.

**Question 1°)**

Les templates vlan-routeur.j2 et vlan-switch.j2 sont disponibles dans le dossier template du TP2-DEVOPS-MAIN.

**Question 2°)**

Les structures de données vlan-R02.JSON et vlan-ESW2.JSON sont disponibles dans le dossier data du TP2-DEVOPS-MAIN.

**Question 2,3,4,5°)**

Le code qui permet de créer les configurations pour les routeurs et switch pour le site de Marseille et Paris est disponible sur le fichier create\_config.py dans scripts.

**Automatisation réseau avec netmiko**

**Question 23°)**

# net\_connect = ConnectHandler(\*\*r01)

**Question 24°)**

Je print le contenu de net\_connect

# print("\*\* Content of net\_connect \*\*")

# print(net\_connect)

Et le résultat est le suivant :

\*\* Begin \*\*

\*\* Content of net\_connect \*\*

<netmiko.cisco.cisco\_ios.CiscoIosSSH object at 0x7fd46cdc65b0>

\*\* Type of net\_connect \*\*

<class 'netmiko.cisco.cisco\_ios.CiscoIosSSH'>

\*\* End \*\*

**Question 25°)**

def question\_25(net\_connect):

    command = "show ip int brief"

    with ConnectHandler(\*\*r01) as net\_connect:

        output = net\_connect.send\_command(command)

    # Automatically cleans-up the output so that only the show output is returned

    print()

    print(output)

    print()

    pass

C’est la méthode send\_command qui permet d’envoyer des commandes

**Question 26°)**

Le résultat semble être formaté en JSON. Le code :

command = "show ip int brief"

with ConnectHandler(\*\*r01) as net\_connect:

output = net\_connect.send\_command(command, use\_textfsm=True)

# Automatically cleans-up the output so that only the show output is returned

print()

print(output)

print()

**Question 27°)**

**Table de routage command = "show ip route"**

[{'protocol': 'S', 'type': '', 'network': '172.16.100.192', 'mask': '26', 'distance': '', 'metric': '', 'nexthop\_ip': '', 'nexthop\_if': 'Serial1/1', 'uptime': ''}, {'protocol': 'S', 'type': '', 'network': '172.16.100.128', 'mask': '26', 'distance': '', 'metric': '', 'nexthop\_ip': '', 'nexthop\_if': 'Serial1/0', 'uptime': ''}, {'protocol': 'C', 'type': '', 'network': '172.16.100.64', 'mask': '26', 'distance': '', 'metric': '', 'nexthop\_ip': '', 'nexthop\_if': 'GigabitEthernet0/0.99', 'uptime': ''}, {'protocol': 'C', 'type': '', 'network': '172.16.20.0', 'mask': '24', 'distance': '', 'metric': '', 'nexthop\_ip': '', 'nexthop\_if': 'GigabitEthernet0/0.20', 'uptime': ''}, {'protocol': 'C', 'type': '', 'network': '172.16.10.0', 'mask': '24', 'distance': '', 'metric': '', 'nexthop\_ip': '', 'nexthop\_if': 'GigabitEthernet0/0.10', 'uptime': ''}, {'protocol': 'C', 'type': '', 'network': '172.16.100.0', 'mask': '26', 'distance': '', 'metric': '', 'nexthop\_ip': '', 'nexthop\_if': 'GigabitEthernet2/0', 'uptime': ''}, {'protocol': 'C', 'type': '', 'network': '10.1.3.0', 'mask': '30', 'distance': '', 'metric': '', 'nexthop\_ip': '', 'nexthop\_if': 'Serial1/1', 'uptime': ''}, {'protocol': 'C', 'type': '', 'network': '10.1.1.0', 'mask': '30', 'distance': '', 'metric': '', 'nexthop\_ip': '', 'nexthop\_if': 'Serial1/0', 'uptime': ''}]

**C’est une liste**

**Question 28°)**

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

**Question 29°)**

C’est la méthode send\_config\_set qui permet d’executer des commandes en mode config.

**Question 30°)**

command = ['interface loopback1','no interface loopback 1']

**Question 31°)**

**a°)**

cf code question31

**b°)**

Une autre option possible serai de parcourir un fichier text (ou JSON) comportant toutes les informations nécessaires. Puis il faut envoyer chaque information avec la méthode send\_config\_set. La variable commande change pour chaque interface.

command = ['interface loopback1','ip address 192.168.1.1 255.255.255.255','description loopback interface from netmiko']

    with ConnectHandler(\*\*r01) as net\_connect:

        output = net\_connect.send\_config\_set(command)

**Question 32,33,34,35,36°)**

Cf code dans le fichier Python run\_netmiko.py

**Question 37°)**

Les pings entre les vlan 10 et 20 du site de Marseille sont censés fonctionner

**Question 38°)**

Les pings entre les vlan 10 et 20 du site de Paris sont censés fonctionner

**Automatisation réseau avec Napalm**

**Question 41°)**

C’est la méthode device.cli() qui permet d’envoyer une commande.

On obtient le résultat suivant en sortie de la commande « show ip int brief » :

\* Begin \*

{' show ip int brief': 'Interface IP-Address OK? Method Status Protocol\nEthernet0/0 unassigned YES NVRAM administratively down down \nGigabitEthernet0/0 unassigned YES NVRAM up up \nGigabitEthernet0/0.10 172.16.10.254 YES NVRAM up up \nGigabitEthernet0/0.20 172.16.20.254 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 NVRAM administratively down down \nSerial1/3 unassigned YES NVRAM administratively down down \nGigabitEthernet2/0 172.16.100.62 YES NVRAM up up'}

\* End \*

**Question 42°)**

Le format de sortie de la question 42 est un dictionnaire.

**Question 43°)**

La seconde méthode est device.get\_arp\_table() sans argument et elle nous retourne.

\* Begin \*

\* question 43 \*

[{'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': '08:00:27:80:1B:A1', 'ip': '172.16.100.2', 'age': 19.0}, {'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}]

<class 'list'>

\* End \*

**Question 44°)**

C’est une liste, comme on peut le voir à la dernière ligne.

**Question 45°)**

**b°)**

Résultat de la commande show ip interface brief

Loopback1 192.168.1.1 YES TFTP up up

Loopback2 192.168.2.1 YES TFTP up up

La configuration est bien déployée.

**Question 46°)**

J’ai généré les 3 fichiers de configuration ospf\_r01.conf, ospf\_r02.conf, ospf\_r03.conf grâce au template ospf.j2 et données des fichiers ospf\_r01.json, ospf\_r02.json, ospf\_r03.json.

**Question 47°)**

On déploie les configurations précédemment créées. Il faut faire attention au hostname.

**Partie 2 Docker : Manipulation de conteneurs et de volumes**

**TP03 : Automatisation Réseau et conteneurisation Docker**

**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°) à 5°)

Les scripts pour déployer les configurations des routeurs et switchs du bâtiment A et B sont disponible dans la partie create\_config du TP3.

6°) à 12°)

Création des configurations et fichier afin d’initialiser nornir.

13°)

Les attributs sont : Data / Inventory / Config / Processors / Runner Inventory

C’est Runner Iventory qui permet de lire l'inventaire

{'data': <nornir.core.state.GlobalState object at 0x7f39645d2070>, 'inventory': <nornir.core.inventory.Inventory object at 0x7f3963b5ab40>, 'config': <nornir.core.configuration.Config object at 0x7f3963bb41d0>, 'processors': [], 'runner': <nornir.plugins.runners.ThreadedRunner object at 0x7f39640efe80>}

14°)

Ce sont tous les noms des hosts de hosts.yaml

{'R1-CPE-BAT-A': Host: R1-CPE-BAT-A, 'R2-CPE-BAT-A': Host: R2-CPE-BAT-A, 'ESW1-CPE-BAT-A': Host: ESW1-CPE-BAT-A, 'R1-CPE-BAT-B': Host: R1-CPE-BAT-B, 'R2-CPE-BAT-B': Host: R2-CPE-BAT-B, 'ESW1-CPE-BAT-B': Host: ESW1-CPE-BAT-B}

Ça ressemble à un dictionnaire mais python retourne un objet de type classe « nornir.core.inventory.Host ».

15°)

La valeur du premier élément : R1-CPE-BAT-A

Le type de l’objet : <class 'nornir.core.inventory.Host'>

C’est normalement une chaine de charactère.

16°)

L’adresse ip : 172.16.100.125

username : Cisco

Ces informations proviennent du fichier hosts.yaml et defdaults.yaml

17°)

On voit les paramètres présent dans hosts.yaml.

key(): dict\_keys(['device\_name', 'device\_type', 'device\_model', 'locality', 'building', 'room', 'vendor'])

Apès avoir ajouté room:001 dans le fichier on peut récupérer la valeur grâce nornir.

room: 1

19°)

{'ios': Group: ios}

20°)

[Group: ios]

21°)

dict\_keys(['vendor'])

22°)

Cisco

23°)

Hostname: R1-CPE-BAT-A

adresse IP: 172.16.100.125

Hostname: R2-CPE-BAT-A

adresse IP: 172.16.100.126

Hostname: ESW1-CPE-BAT-A

adresse IP: 172.16.100.123

Hostname: R1-CPE-BAT-B

adresse IP: 172.16.100.189

Hostname: R2-CPE-BAT-B

adresse IP: 172.16.100.190

Hostname: ESW1-CPE-BAT-B

adresse IP: 172.16.100.187

24°)

dict\_keys(['R1-CPE-BAT-A', 'R2-CPE-BAT-A', 'R1-CPE-BAT-B', 'R2-CPE-BAT-B'])

25°)

dict\_keys(['ESW1-CPE-BAT-A', 'ESW1-CPE-BAT-B'])

26°)

AggregatedResult (hello\_world): {'R1-CPE-BAT-A': MultiResult: [Result: "hello\_world"], 'R2-CPE-BAT-A': MultiResult: [Result: "hello\_world"], 'ESW1-CPE-BAT-A': MultiResult: [Result: "hello\_world"], 'R1-CPE-BAT-B': MultiResult: [Result: "hello\_world"], 'R2-CPE-BAT-B': MultiResult: [Result: "hello\_world"], 'ESW1-CPE-BAT-B': MultiResult: [Result: "hello\_world"]}

27°) La méthode type retourne: <class 'nornir.core.task.AggregatedResult'>

Meme si le résultat ressemble à un dictionnaire.

28°)

29°) On reçoit le message qui est envoyé par tous les diffents équipement, switchs et routeurs.

30°) Seul les switchs envoient les messages.

hello\_world\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ESW1-CPE-BAT-A \* changed : False \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

vvvv hello\_world \*\* changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO

ESW1-CPE-BAT-A says hello world!

^^^^ END hello\_world ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

\* ESW1-CPE-BAT-B \* changed : False \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

vvvv hello\_world \*\* changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO

ESW1-CPE-BAT-B says hello world!

^^^^ END hello\_world ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**Partie 2 : Manipulation des différents réseaux Docker et création d’un cluster Docker Swarm**

**1°) 2°)** Création du compte docker, et connection à la sandbox puis connection en ssh sur mon terminal Windows avec la commande ssh ip172-18-0-93dllem60qau000cggelg@direct.labs.

play-with-docker.com

**3°)** Après avoir cliqué effectué la commande suivante :

docker run --name nginx-01 -d -p 8080:80 nginx

Puis on ouvre le port 8080, et on peut voir le message «welcom to nginx » sur la page web.

**4°)** Pour rentrer dans le bash: docker exec -it nginx-01 bash

Une fois qu’on est dans le bash, on peut normalement ping, mais ce n’est pas le cas.

**5°)** Pour que le commande ifconfig soit fonctionnel il faut exécuter les commandes suivantes :

apt-get update

apt-get install net tools

ifconfig

**6°)** la commande pour que le ping soit fonctionnel :apt-get install iputils-ping

Puis on essai si ça marche, on se connecte au bash pour ping : docker exec -it nginx-cpe bash

et on teste si le ping marche, mais avant on affiche l’adresse IP du conteneur.

ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 172.17.0.3 netmask 255.255.0.0 broadcast 172.17.255.255

ether 02:42:ac:11:00:03 txqueuelen 0 (Ethernet)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536

inet 127.0.0.1 netmask 255.0.0.0

loop txqueuelen 1 (Local Loopback)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

On ping:

root@670bfed7e520:/# ping google.fr

PING google.fr (172.253.122.94) 56(84) bytes of data.

64 bytes from bh-in-f94.1e100.net (172.253.122.94): icmp\_seq=1 ttl=101 time=2.34 ms

64 bytes from bh-in-f94.1e100.net (172.253.122.94): icmp\_seq=2 ttl=101 time=2.18 ms

64 bytes from bh-in-f94.1e100.net (172.253.122.94): icmp\_seq=3 ttl=101 time=2.09 ms

^Cè

Ça fonctionne !

**7°) 8°)**

Il faut créer un docker file dans root : vim dockerfile

Le contenu du Dockerfile

1 FROM nginx

2 RUN apt-get update -y

3 RUN apt-get install net-tools -y

4 RUN apt-get install iputils-ping -y

puis on build l’image: docker build -t nginx-cpe .

Puis on run le conteneur en back ground: docker run -d -name nginx-cpe

**9°)**

On arrête le docker: Docker stop + id du docker

Puis on lance le nouveau docker : docker run -d --name nginx-cpe-01 nginx-cpe

**10°)**

On lance la commande pour entrer dans le bash :docker exec -it nginx-cpe-01 bash

1ere commande :

root@59d1ceca0442:/# ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 172.17.0.3 netmask 255.255.0.0 broadcast 172.17.255.255

ether 02:42:ac:11:00:03 txqueuelen 0 (Ethernet)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536

inet 127.0.0.1 netmask 255.0.0.0

loop txqueuelen 1 (Local Loopback)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

2eme commande :

root@59d1ceca0442:/# ping 8.8.8.8

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.

64 bytes from 8.8.8.8: icmp\_seq=1 ttl=54 time=1.17 ms

64 bytes from 8.8.8.8: icmp\_seq=2 ttl=54 time=1.07 ms

64 bytes from 8.8.8.8: icmp\_seq=3 ttl=54 time=1.11 ms

^C

--- 8.8.8.8 ping statistics ---

3 packets transmitted, 3 received, 0% packet loss, time 2001ms

rtt min/avg/max/mdev = 1.066/1.115/1.165/0.040 ms

tout fonctionne !

**11°)**

docker run -d --name nginx-cpe-02 -p 8081:80 nginx-cpe

pour nginx-cpe-02 :

root@57059c494f30:/# hostname --all-ip-addresses

172.17.0.4

pour nginx-cpe-01:

$ docker exec -it nginx-cpe-01 bash

root@59d1ceca0442:/# hostname --all-ip-addresses

172.17.0.3

C’est le meme subnet.

**12°)** les deux conteneurs se trouve sur le même réseau 172.17.0.0/16.

**13°)** la passerelle par défaut est la 172.17.0.1.

**14°)** Le ping marche entre les deux conteneurs :

oui ça ping

$ docker exec -it nginx-cpe-01 bash

root@59d1ceca0442:/# ping 172.17.0.4

PING 172.17.0.4 (172.17.0.4) 56(84) bytes of data.

64 bytes from 172.17.0.4: icmp\_seq=1 ttl=64 time=0.220 ms

64 bytes from 172.17.0.4: icmp\_seq=2 ttl=64 time=0.091 ms

64 bytes from 172.17.0.4: icmp\_seq=3 ttl=64 time=0.158 ms

^C

--- 172.17.0.4 ping statistics ---

3 packets transmitted, 3 received, 0% packet loss, time 2006ms

rtt min/avg/max/mdev = 0.091/0.156/0.220/0.052 ms

root@59d1ceca0442:/# exit

exit

sur l’autre conteneur :

[node1] (local) root@192.168.0.8 ~

$ docker exec -it nginx-cpe-02 bash

root@57059c494f30:/# ping 172.17.0.3

PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.

64 bytes from 172.17.0.3: icmp\_seq=1 ttl=64 time=0.169 ms

64 bytes from 172.17.0.3: icmp\_seq=2 ttl=64 time=0.109 ms

64 bytes from 172.17.0.3: icmp\_seq=3 ttl=64 time=0.097 ms

64 bytes from 172.17.0.3: icmp\_seq=4 ttl=64 time=0.095 ms

^C

--- 172.17.0.3 ping statistics ---

4 packets transmitted, 4 received, 0% packet loss, time 3000ms

rtt min/avg/max/mdev = 0.095/0.117/0.169/0.030 ms

[root@57059c494f30](mailto:root@57059c494f30):/#

**15°)** docker network create cpe-bridge-network --attachable -d bridge –subnet=192.168.1.0/24

**16°)**

$ docker network connect cpe-bridge-network nginx-cpe-01

**17°)**

nouvelle interface $ docker exec -it nginx-cpe-01 bash

root@59d1ceca0442:/# ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 172.17.0.3 netmask 255.255.0.0 broadcast 172.17.255.255

ether 02:42:ac:11:00:03 txqueuelen 0 (Ethernet)

RX packets 16 bytes 1232 (1.2 KiB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 16 bytes 1232 (1.2 KiB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.1.3 netmask 255.255.255.0 broadcast 192.168.1.255

ether 02:42:c0:a8:01:03 txqueuelen 0 (Ethernet)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536

inet 127.0.0.1 netmask 255.0.0.0

loop txqueuelen 1 (Local Loopback)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

**Il y a deux adresse ip.**

**18°)** "NetworkSettings": {

"Bridge": "",

"SandboxID": "fc26017390d5c7ab00d7e2ce4d48e3ccaf4d3bca9ff4a7b403c38809e1af6d55",

"HairpinMode": false,

"LinkLocalIPv6Address": "",

"LinkLocalIPv6PrefixLen": 0,

"Ports": {

"80/tcp": null

},

"SandboxKey": "/var/run/docker/netns/fc26017390d5",

"SecondaryIPAddresses": null,

"SecondaryIPv6Addresses": null,

"EndpointID": "6394b997796d5bd59b1f5d0621823abfcae36a973ea7e2a8a38ca2a50c5404f6",

"Gateway": "172.17.0.1",

"GlobalIPv6Address": "",

"GlobalIPv6PrefixLen": 0,

"IPAddress": "172.17.0.3",

"IPPrefixLen": 16,

"IPv6Gateway": "",

"MacAddress": "02:42:ac:11:00:03",

"Networks": {

"bridge": {

"IPAMConfig": null,

"Links": null,

"Aliases": null,

"NetworkID": "17fd26c0db0245d945d9e8cfd303cb892154d5f6d3b32f75e509131f7f70306a",

"EndpointID": "6394b997796d5bd59b1f5d0621823abfcae36a973ea7e2a8a38ca2a50c5404f6",

"Gateway": "172.17.0.1",

"IPAddress": "172.17.0.3",

"IPPrefixLen": 16,

"IPv6Gateway": "",

"GlobalIPv6Address": "",

"GlobalIPv6PrefixLen": 0,

"MacAddress": "02:42:ac:11:00:03",

"DriverOpts": null

},

"cpe-bridge-network": {

"IPAMConfig": {},

"Links": null,

"Aliases": [

"59d1ceca0442"

],

"NetworkID": "8342d7748eb5d5939ece07c66f3f768333adb43249542fe04924e792491315cf",

"EndpointID": "73b21667b590cba07926a7a95812f79ed0e97200bd4c4f1e185a8a6e3674a617",

"Gateway": "192.168.1.1",

"IPAddress": "192.168.1.3",

"IPPrefixLen": 24,

"IPv6Gateway": "",

"GlobalIPv6Address": "",

"GlobalIPv6PrefixLen": 0,

"MacAddress": "02:42:c0:a8:01:03",

"DriverOpts": {}

}

}

}

nouvelle catégorie cpe-bridge-network

**19°)** $ docker network ls

NETWORK ID NAME DRIVER SCOPE

17fd26c0db02 bridge bridge local

8342d7748eb5 cpe-bridge-network bridge local

85bf5be09cd3 host host local

afc9e97a5a62 none null local

puis: $ docker network disconnect bridge nginx-cpe-01

Il existe encore mais déconnecté

**20°)**

$ docker exec -it nginx-cpe-01 bash

root@59d1ceca0442:/# ping 172.17.0.4

PING 172.17.0.4 (172.17.0.4) 56(84) bytes of data.

^C

--- 172.17.0.4 ping statistics ---

4 packets transmitted, 0 received, 100% packet loss, time 3000ms

Le ping ne marche pas.

**21°)**

On connecte le bridge

[node1] (local) root@192.168.0.8 ~

$ docker network connect cpe-bridge-network nginx-cpe-02

on va dans le batch

[node1] (local) root@192.168.0.8 ~

$ docker exec -it nginx-cpe-02 bash

on ping l’addresse du bridge d’en face

root@57059c494f30:/# ping 192.168.1.3

PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.

64 bytes from 192.168.1.3: icmp\_seq=1 ttl=64 time=0.187 ms

64 bytes from 192.168.1.3: icmp\_seq=2 ttl=64 time=0.086 ms

--- 192.168.1.3 ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 1003ms

rtt min/avg/max/mdev = 0.086/0.136/0.187/0.050 ms

ça marche !

**22°)**

Cf colonne driver

$ docker network ls

NETWORK ID NAME DRIVER SCOPE

17fd26c0db02 bridge bridge local

8342d7748eb5 cpe-bridge-network bridge local

85bf5be09cd3 host host local

afc9e97a5a62 none null local

**23°)**

Il n’y a pas de subnet :

$ docker network inspect none

[

{

"Name": "none",

"Id": "afc9e97a5a6272d8435285b61b738a690daced78ad7a9fb5f780766a9628dac4",

"Created": "2022-11-09T07:34:06.713922597Z",

"Scope": "local",

"Driver": "null",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": null,

"Config": []

},

"Internal": false,

"Attachable": false,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {},

"Options": {},

"Labels": {}

}

]

**24°)**

on deconnecte le bridge

$ docker network disconnect bridge nginx-cpe-02

[node1] (local) root@192.168.0.8 ~

$ docker network disconnect cpe-bridge-network nginx-cpe-01

[node1] (local) root@192.168.0.8 ~

$ docker network connect none nginx-cpe-01

[node1] (local) root@192.168.0.8 ~

$ docker network disconnect cpe-bridge-network nginx-cpe-02

[node1] (local) root@192.168.0.8 ~

$ docker network connect none nginx-cpe-02

[node1] (local) root@192.168.0.8 ~

**25°)** La section network est vide.

**26°)**

$ docker exec -it nginx-cpe-01 bash

root@59d1ceca0442:/# ifconfig

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536

inet 127.0.0.1 netmask 255.0.0.0

loop txqueuelen 1 (Local Loopback)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@59d1ceca0442](mailto:root@59d1ceca0442):/#

plus qu’un réseau

**27°)**

$ docker exec -it nginx-cpe-01 bash

root@59d1ceca0442:/# ifconfig

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536

inet 127.0.0.1 netmask 255.0.0.0

loop txqueuelen 1 (Local Loopback)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@59d1ceca0442:/# ping google.fr

ping: connect: Cannot assign requested address

root@59d1ceca0442:/# ping 8.8.8.8

ping: connect: Network is unreachable

rien ne marche car il n’y a plus de port.

**28°)**

$ docker swarm init --advertise-addr 192.168.0.28

Swarm initialized: current node (x02as83vmhhzm28m5vyqf8mm7) is now a manager.

To add a worker to this swarm, run the following command:

docker swarm join --token SWMTKN-1-5j5s556zlay4fa1p1pcki61tncefeca8eimo1v9ix1sp3vl42e-agucxh9n4ewn24fi13c078czi 192.168.0.28:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

**29°)**

$ docker swarm join --token SWMTKN-1-5j5s556zlay4fa1p1pcki61tncefeca8eimo1v9ix1sp3vl42e-agucxh9n4ewn24fi13c078czi 192.16

8.0.28:2377

This node joined a swarm as a worker.

**30°)**

$ docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION

x02as83vmhhzm28m5vyqf8mm7 \* node1 Ready Active Leader 20.10.17

viu1rjmej4subfs1u8xf59o4t node2 Ready Active 20.10.17

**31°)**

$ docker build -t nginx-cpe .

**32°)**

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

nginx-cpe latest 166b74c56b68 28 seconds ago 164MB

nginx latest 76c69feac34e 2 weeks ago 142MB

**33°)**

C’est vrai !

[node2] (local) root@192.168.0.27 ~

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

**34°)**

[node1] (local) root@192.168.0.28 ~

$ docker run -d --name nginx-cpe-01 -p 8080:80 nginx-cpe

**35°)**

[node2] (local) root@192.168.0.27 ~

$ vim Dockerfile

Build an image from a Dockerfile

[node2] (local) root@192.168.0.27 ~

$ docker build -t nginx-cpe .

**36°)**

Il faut faire if config pour connaitre l’adresse IP

Puis ping le slave depuis le worker et inversement.

On trouve la même adresse IP, le ping n’est donc pas censé marcher.

**37°)**

La solution est la création d’un nouveau network overlay sur le manager.

**38°)**

docker network create --attachable -d overlay --subnet=192.168.1.0/24 cpe-overlay-network

**39°)**

Sur le manager: docker network connect cpe-overlay-network nginx-cpe-01

docker network disconnect bridge nginx-cpe-01

**40°)**

Sur le worker: docker network connect cpe-overlay-network nginx-cpe-02

docker network disconnect bridge nginx-cpe-02

**42°)**

Le ping fonctionne dans les eux sens grâce au network overlay.

**43°)**

Sur le manager:

Cd

mkdir cpe\_api

**44°)**

pipenv install flask

Mais avant il faut installer pipenv avec pip install pipenv !

**45°)**

pipenv shell

**46°)**

vim api.py

**47°)**

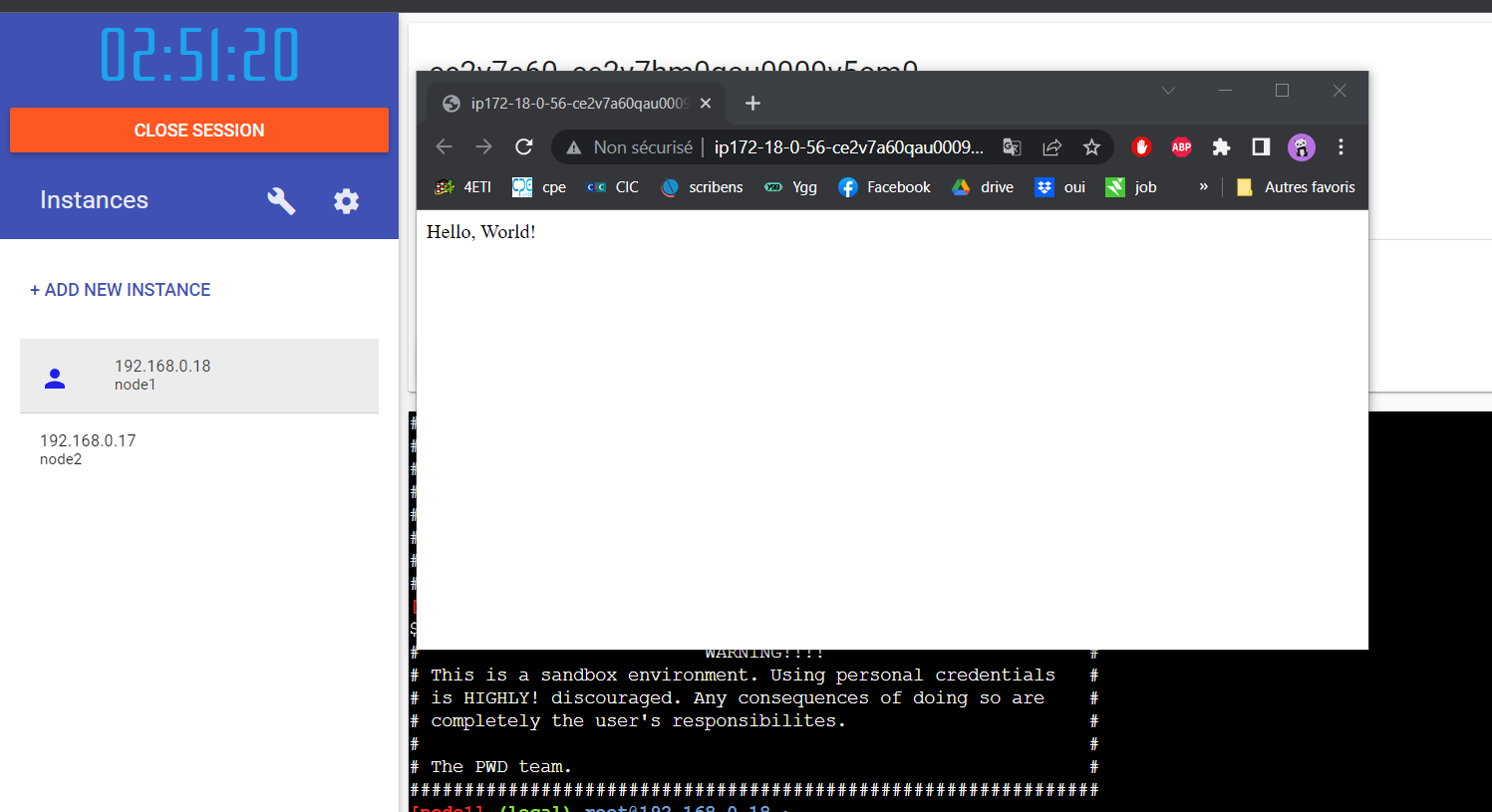
export FLASK-ENV=dev

export FLASK\_APP=api.py

**48°)**

flask run -host=0.0.0.0

**49°)**

On arrive bien à voir le message si on ouvre le port 5000

**50°)**

Affichage sous forme de fichier JSON

Graphical user interface, text, application, chat or text message

Description automatically generated

**51°)**

**a°)** la commande lock ne marche pas à cause -r qui n’existe pas.

Il faut faire pip freeze > requirements.txt à la place sinon le fichier reste vide.

**b°)**

vim dockerfile

**c°)**

docker build -t cpe\_api\_image .

**d°)**

docker run --name cpe\_api\_01 -d -p 8080:5000 cpe\_api\_image

**e°)**

Tout fonctionne normalement.

Graphical user interface, text, application

Description automatically generated

**53°)**

docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.

Username: damienollier

Password:

WARNING! Your password will be stored unencrypted in /root/.docker/config.json.

Configure a credential helper to remove this warning. See

https://docs.docker.com/engine/reference/commandline/login/#credentials-storeGraphical user interface, text, application, email

Description automatically generated

**54°)**

docker run --name damienollier/cpe\_api:1.0 -d -p 8080:5000 cpe\_api\_image

**55°)**

docker push damienollier/cpe\_api:1.0

**56°)**

Graphical user interface, application

Description automatically generated

**57°)**

docker service create --name cpe\_api -p 8080:5000 --with-registry-auth damienollier/cpe\_api:1.0

**58°)**

Le service est bien présent après docker service ls (j’ai oublié de prendre le screen)

**59°)**

**60°)**

$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

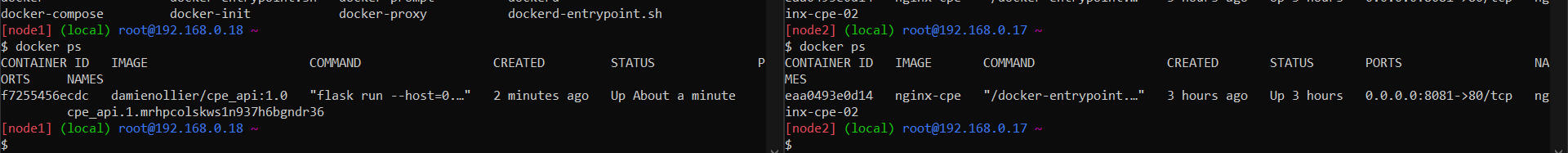
3810e9614fc8 damienollier/cpe\_api:1.0 "flask run --host=0.…" About a minute ago Up 58 seconds cpe\_api.1.pgn0hmwqjbymkvlaaimq15gb5

**61°)**

On a accès à l’affichage

Graphical user interface, text, application

Description automatically generated

**62°)**

Depuis le manager on voit le docker, depuis worker on ne voit rien

**63°)**

On n’observe rien.

Graphical user interface, text, application

Description automatically generated

**64°)**

$ docker container stop f7255456ecdc

f7255456ecdc

Même après avoir arrêté le docker, on peut toujours avoir accès à la page web.

Graphical user interface, text, application

Description automatically generated

65°) On génère les 10 instances.

Text

Description automatically generated

66°)

Text

Description automatically generated

69°)

$ cat api.py

from flask import Flask

from flask import jsonify

app = Flask(\_\_name\_\_)

@app.route("/")

def hello\_world():

return jsonify(message="Hello, World", version=”2.0”)

a°) docker built -t damienollier/cpe\_api\_image:2.0

b°) docker push damienollier/cpe\_api:2.0

c°) docker service update cpe\_api -image damienollier/cpe\_api\_image:2.0

**70°)** docker run -it -d -p 5000:8080 -v /var/run/docker.sock:/var/run/docker.sock dockersamples/visualizer