07/11/2025

TP 01 Module Devnet

Automatisation Réseau

Ronan FOURNEUVE
CPE LYON

Table des matières

Partie 1 Générer des configurations réseau avec Python

1 – Automatiser la création de configuration réseau pour switchs et routeurs Cisco

1.6 - Préparation des variables pour le template JINJA

Question 1.6.1 : Quels sont les éléments de configuration à réaliser pour que les équipements du vlan 10 et du vlan 20 du bâtiment C puissent communiquer ? (ne faites pas la configuration pour l'instant)

Configuration requise pour le routeur R2

Les éléments de configuration suivants doivent être mis en place sur le routeur R2 :

- Configuration des sous-interfaces pour le routage inter-VLAN :
    - Création des sous-interfaces logiques :
        - g0/0.10 pour le VLAN 10
        - g0/0.20 pour le VLAN 20
    - Attribution d'une adresse IP passerelle à chaque sous-interface
    - Configuration de l'encapsulation dot1Q avec les numéros de VLAN correspondants
    - Ajout de descriptions claires pour identifier chaque sous-interface
- Activation du routage IP et vérification de l'état des interfaces

Configuration requise pour le switch ESW2

Les éléments de configuration suivants doivent être mis en place sur le switch ESW2 :

- Création des VLANs :
    - VLAN 10 → réseau utilisateurs teacher bâtiment C
    - VLAN 20 → réseau utilisateurs student bâtiment C

    (Le VLAN 99 d'administration n'est pas nécessaire ici, car la question ne concerne que les VLANs 10 et 20.)
- Configuration des interfaces :
    - Affectation des ports d'accès aux VLANs correspondants (mode access)
    - Configuration du port trunk vers le routeur R2 pour transporter les trames des VLANs 10 et 20
    - Ajout de descriptions sur les interfaces pour une meilleure documentation

.

Question 1.6.2 : Parmi les éléments de configuration identifiés à la question 1, quelles sont les variables Jinja que vous avez pu identifier ?

Les variables Jinja identifiées permettent de rendre la configuration des équipements réseau dynamique et adaptable :

- **{{vlan_id}}** : identifiant du VLAN
- **{{name}}** : nom de l'interface
- **{{description}}** : description de l'interface
- **{{ip}}** : adresse IP de l'interface
- **{{mask}}** : masque de sous-réseau associé à l'adresse IP
- **{{hostname}}** : nom de la machine
- **{{mode}}** : définit le rôle d'un port dans le transport des vlans

Question 1.6.3 : Définissez une structure de données JSON pour recenser les variables identifiées ainsi que leurs valeurs. Créez les fichiers R2.json et ESW2.json dans le dossier data.

Ces fichiers JSON recensent les variables de configuration des interfaces pour le routeur R2 et le switch ESW2, incluant noms, VLAN, descriptions, et adresses IP le cas échéant.

R2.json :

```
{
  "hostname": "R2",
  "interfaces": [
    {
      "name": "g0/0.10",
      "description": "Passerelle pour le reseau enseignants",
      "ip": "172.16.30.254",
      "mask": "255.255.255.0",
      "vlan_id": "10"
    },
    {
      "name": "g0/0.20",
      "description": "Passerelle pour le reseau etudiants",
      "ip": "172.16.40.254",
      "mask": "255.255.255.0",
      "vlan_id": "20"
    }
```

```
  ]
}
```

ESW2.json

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

1.7 - Développer la fonction python permettant de lire les données définis dans les fichiers R2.json et ESW2.json

Question 1.7.4 : Créez une fonction dans le fichier scripts/ __main__.py permettant de retourner le contenu du fichier JSON passé en paramètre.

```
def load_json_data_from_file(file_path):
    with open(file_path) as json_file:
        data = json.load(json_file)
    return data
```

La fonction load_json_data_from_file permet de lire un fichier JSON et de retourner son contenu. Voici son fonctionnement : elle prend en paramètre le chemin du fichier à lire, ouvre le fichier, charge les données JSON et les retourne.

```
if __name__ == "__main__":

    #process R2

    r2_data = load_json_data_from_file(file_path='data/R2.json')

    esw2_data = load_json_data_from_file(file_path='data/ESW2.json')

    print(r2_data)
```

Le fichier complet est fourni dans le dossier Zip.

J'exécute le script en exécutant la commande suivante : `python3 -m scripts`

Le `-m` permet d'executer un module comme un scripts, Python va chercher le fichier `__main__.py` à l'intérieur et l'exécuter.

Question 1.7.5 : Essayez d'exécuter votre script avec un chemin d'un fichier inexistant. Que se passe-t-il ? Quel est le type d'erreur qui est levé ?*

```
Traceback (most recent call last):

  File "<frozen runpy>", line 198, in _run_module_as_main

  File "<frozen runpy>", line 88, in _run_code

  File "/home/cpe/workspace/devnet/TP-01/scripts/__main__.py", line 36, in <module>

    fichier_inconnu_data = load_json_data_from_file(file_path='data/file-unknown.json')

                           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/home/cpe/workspace/devnet/TP-01/scripts/__main__.py", line 5, in
load_json_data_from_file

    with open(file_path) as json_file:

         ^^^^^^^^^^^^^^^

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

Pour tester la gestion des fichiers inexistants, j'ai ajouté dans le script une ligne qui tente de charger un fichier JSON inexistant (data/file-unknown.json).

Lors de l'exécution, le script s'arrête sur cette ligne car Python ne trouve pas le fichier et lève une erreur `FileNotFoundError`. Le message `Errno 2` signifie « No such file or directory », ce qui indique que le chemin fourni n'existe pas. L'erreur apparaît dès que la fonction `open()` tente d'ouvrir le fichier.

Question 1.7.6 : Améliorez votre code pour gérer les exceptions (erreurs) dans le cas ou le chemin du fichier n'est pas correct.

```
def load_json_data_from_file(file_path):

    try:

        with open(file_path, "r", encoding="utf-8") as json_file:

            return json.load(json_file)

    except FileNotFoundError:

        print(f"Erreur : le fichier '{file_path}' n'existe pas.")
```

```
    return None
```

La fonction essaie d'ouvrir et de lire le fichier, et affiche un message d'erreur si le chemin n'est pas correct.

1.8 - Définition des templates JInja pour la configuration du routeur R2 et du switch ESW2

Question 1.8.7 : Créez le template R2.j2 dans le dossier templates et définissez le contenu de la config et ses variables

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

```
hostname {{ hostname }}

{% for interface in interfaces %}

interface {{ interface.name }}

 description {{ interface.description }}

 ip address {{ interface.ip }} {{ interface.mask }}

 encapsulation dot1Q {{ interface.vlan_id }}

 no shutdown

{% endfor %}
```

Nous avons utilisé une boucle for pour parcourir toutes les interfaces du fichier JSON. Chaque interface est configurée avec ses attributs (description, adresse IP, masque, …)

Question 1.8.8 : Créez le template ESW2.j2 dans le dossier templates et définissez le contenu de la config et ses variables

```
hostname {{ hostname }}


! Configuration des interfaces

{% for interface in interfaces %}

interface {{ interface.name }}

 description {{ interface.description }}

{% if interface.mode == "access" %}

 switchport mode access

 switchport access vlan {{ interface.vlan_id }}

{% elif interface.mode == "trunk" %}

 switchport mode trunk

 switchport trunk allowed vlan {{ interface.vlan_id }}

{% endif %}

 no shutdown

{% endfor %}
```

Nous avons utilisé une boucle for pour parcourir toutes les interfaces du switch définies dans le fichier JSON, en configurant automatiquement pour chacune son nom, sa description, son mode (access ou trunk) et les VLANs associés, tout en conservant une structure de commandes uniforme.

1.9 - Génération automatique de la configuration réseau

Question 1.9.9 : Installez les dépendances nécessaires pour utiliser les fonctions du package Jinja2

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

```
pipenv install Jinja2
```

Question 1.9.9.a : Importez-les packages en début de fichier de votre script python (__main__.py)

Nous importons les packages nécessaires de Jinja2, à savoir Template, Environment et FileSystemLoader, pour pouvoir créer et gérer nos templates

```
from jinja2 import Template, Environment, FileSystemLoader
```

Question 1.9.9.b : Définissez le répertoire par défaut contenant vos templates Jinja

La ligne de commande permet de créer un environnement Jinja2 qui va charger les templates depuis le dossier templates.

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

Question 1.9.10 : Complétez la fonction render_network_config pour que celle-ci retourne le résultat de configuration générée pour chaque équipement (R2 / ESW2).

```
def render_network_config(template_name, data):

    template = env.get_template(template_name)

    return template.render(data)
```

La fonction render_network_config permet de générer automatiquement une configuration réseau à partir d'un template Jinja2 et du dictionnaire de données (fichier JSON).

Question 1.9.11 : Développez une fonction save_built_config permettant de sauvegarder la configuration générée par la fonction de la question 10. Les fichiers de conf doivent automatiquement être stockés dans le dossier config :

```
def save_built_config(file_name, data):

    with open(file_name, "w") as f:

        f.write(data)

    return file_name
```

Cette fonction `save_built_config` permet de sauvegarder la configuration générée dans un fichier. Elle prend en entrée le nom du fichier (file_name) et le texte de configuration (data), écrit ce texte dans le fichier spécifié, et renvoie le chemin du fichier créé. Tous les fichiers de configuration sont sauvegarder dans le dossier config.

Question 1.9.12 : Implémentez la configuration sur les équipements et testez la communication entre les machines du vlan 10 et du vlan 20 du bâtiment C :

Je configure les équipements réseau et je teste la communication entre les machines du VLAN 10 et du VLAN 20 dans le bâtiment C, et la communication fonctionne correctement.

PC3 :

```
PC3> ping 172.16.40.1

84 bytes from 172.16.40.1 icmp_seq=1 ttl=63 time=29.492 ms

84 bytes from 172.16.40.1 icmp_seq=2 ttl=63 time=18.128 ms
```

ESW2 :

```
ESW2#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
ESW2(config)#hostname ESW2
ESW2(config)#
ESW2(config)#! Configuration des interfaces
ESW2(config)#
ESW2(config)#interface f1/1
ESW2(config-if)# description Connexion vers le VLAN 10
ESW2(config-if)#
ESW2(config-if)# switchport mode access
ESW2(config-if)# switchport access vlan 10
ESW2(config-if)#
ESW2(config-if)# no shutdown
ESW2(config-if)#
ESW2(config-if)#interface f1/2
ESW2(config-if)# description Connexion vers le VLAN 20
ESW2(config-if)#
ESW2(config-if)# switchport mode access
ESW2(config-if)# switchport access vlan 20
ESW2(config-if)#
ESW2(config-if)# no shutdown
ESW2(config-if)#
ESW2(config-if)#interface f1/0
ESW2(config-if)# description Connexion vers le routeur R2
ESW2(config-if)#
ESW2(config-if)# switchport mode trunk
ESW2(config-if)# switchport trunk allowed vlan add 10,20
ESW2(config-if)#
ESW2(config-if)# no shutdown
ESW2(config-if)#end
```

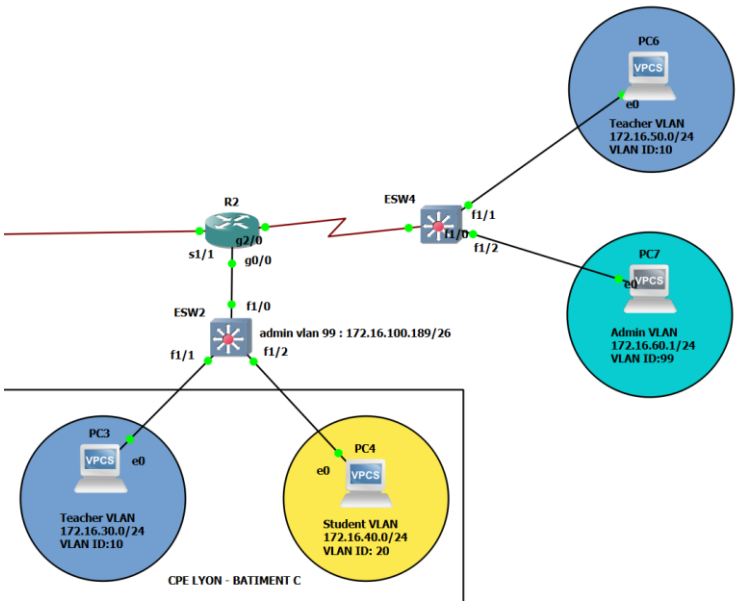
R2 :

```
R2#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#hostname R2
R2(config)#
R2(config)#interface g0/0.10
R2(config-subif)# description Gateway pour le reseau 172.16.30.0/24
R2(config-subif)# ip address 172.16.30.254 255.255.255.0
R2(config-subif)# encapsulation dot1Q 10
R2(config-subif)# no shutdown
R2(config-subif)#
R2(config-subif)#interface g0/0.20
R2(config-subif)# description Gateway pour le reseau 172.16.40.0/24
R2(config-subif)# ip address 172.16.40.254 255.255.255.0
R2(config-subif)# encapsulation dot1Q 20
R2(config-subif)# no shutdown
R2(config-subif)#
R2(config-subif)#end
```

1.10 - Ajout d'un nouveau bâtiment D à l'architecture

Question 1.10.13 : Connectez un deuxième switch au routeur R2 avec deux VPCs connectés au switch :

Ci-dessous le schéma avec les modifications ainsi que les commandes exécutées sur les deux VPCs.



PC5 :

```
PC5> ip 172.16.50.1/24 172.16.50.254

Checking for duplicate address...

PC5 : 172.16.50.1 255.255.255.0 gateway 172.16.50.254
```

PC6 :

```
PC6> ip 172.16.60.1/24 172.16.60.254

Checking for duplicate address...

PC6 : 172.16.60.1 255.255.255.0 gateway 172.16.60.254
```

Question 1.10.14 : Reprenez le jeux de données JSON défini dans vos fichiers R2.json et ESW2.json et créez les jeux de données en YAML dans les fichiers R2.yaml et ESW4.yaml avec les données du tableau précédent.

Vous trouverez ci-dessous les versions YAML des fichiers R2 et ESW4, créées à partir des données du tableau précédent.

R2.yaml :

```
hostname: R2

interfaces:
```

```
  - name: g0/0
    description: Interface physique parent
    ip: null
    mask: null
  - name: g0/0.10
    description: Sous-interface g0/0.10
    ip: 172.16.30.254
    mask: 255.255.255.0
    vlan_id: 10
  - name: g0/0.20
    description: Sous-interface g0/0.20
    ip: 172.16.40.254
    mask: 255.255.255.0
    vlan_id: 20
  - name: g2/0
    description: Lien principal vers le switch ESW4 (trunk)
    ip: null
    mask: null
  - name: g2/0.10
    description: Gateway pour le réseau 172.16.50.0/24
    ip: 172.16.50.254
    mask: 255.255.255.0
    vlan_id: 10
  - name: g2/0.20
    description: Gateway pour le réseau 172.16.60.0/24
    ip: 172.16.60.254
    mask: 255.255.255.0
    vlan_id: 20
  - name: g2/0.99
    description: Interface de gestion VLAN 99
    ip: 172.16.100.254
    mask: 255.255.255.192
    vlan_id: 99
```

ESW4.yaml :

```
hostname: ESW4
management:
  ip_address: 172.16.100.253
  subnet: 255.255.255.192
```

```
  vlan: 99


interfaces:
  - name: f1/1
    mode: access
    vlan_id: 10
    description: Connexion vers VPC-5 (172.16.50.1/24)
  - name: f1/2
    mode: access
    vlan_id: 20
    description: Connexion vers VPC-6 (172.16.60.1/24)
  - name: f1/0
    mode: trunk
    vlan_id: "10,20,99"
    description: Connexion vers le routeur R2
```

Question 1.10.15 : Installez les dépendances pour utiliser les fonctions du package yaml et importer le package en début de fichier de votre script

Après avoir installé le package avec la commande pipenv install PyYAML, on importe le module dans le script à l'aide de import yaml.

Question 1.10.16 : Créez une fonction permettant de retourner automatiquement le contenu des fichiers R2.yaml et ESW4.yaml

La fonction ci-dessous prend en entrée le chemin d'un fichier YAML et retourne automatiquement son contenu.

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

Question 1.10.17 : Dans quel format (type) sont les données de sortie de la fonction à la question 16 ?

Les données de sortie de la fonction `load_yaml_data_from_file` sont de type dict en Python, car `yaml.safe_load()` transforme le contenu du fichier YAML en dictionnaire. Pour cela j'utilise la fonction type pour déterminer le type de la variable.

Lien vers la documentation

Question 1.10.18 : D'après-vous est-il utile de modifier les templates Jinja2 pour que votre configuration depuis yaml soit prise en compte ? Pourquoi?

Nous n’avons pas besoin de modifier les templates tant que la structure du dictionnaire reste la même avec les mêmes clés et valeurs, car le type ne change pas. Dans notre cas, il suffit d’ajouter l’IP de management.

Question 1.10.19 : Sauvegardez automatiquement la configuration dans les fichiers suivant : configs/R2_from_yaml.conf et configs/ESW2_from_yaml.conf

La configuration est automatiquement générée à partir des fichiers YAML R2.yaml et ESW2.yaml puis sauvegardée dans le dossier configs

```
r2_data_yaml = load_yaml_data_from_file(file_path='data/R2.yaml')

r2_config_yaml = render_network_config(template_name='R2.j2', data=r2_data_yaml)

save_built_config('config/R2_from_yaml.conf', r2_config_yaml)


esw2_data_yaml = load_yaml_data_from_file(file_path='data/ESW2.yaml')

esw2_config_yaml = render_network_config(template_name='ESW2.j2', data=esw2_data_yaml)

save_built_config('config/ESW2_from_yaml.conf', esw2_config_yaml)
```

Question 1.10.20 : Implémentez la configuration générée sur les équipements R2 et ESW4 (simple copier / coller) et testez que les communications fonctionnent entre les vlan 10 et 20 et vers les autres bâtiments.

Je configure les équipements réseau et je teste la communication entre les machines du VLAN, et la communication fonctionne correctement.

PC6 :

```
PC6> ping 172.16.30.1

84 bytes from 172.16.30.1 icmp_seq=1 ttl=63 time=29.702 ms

84 bytes from 172.16.30.1 icmp_seq=2 ttl=63 time=18.538 ms
```