UNIVERSITÉ CAEN NORMANDIE	BUT Réseaux et Télécommunications - Ifs	2024/2025
	R405	Clément Brisacier
	TP 1 - pipenv et jinja2	

Cet TP sera effectué sous Debian (pas de Windows) avec un accès réseau classique IUT en 802.1x. Il n'est pas nécessaire - ni recommandé - de se loguer en tant qu'utilisateur root mais plutôt en tant qu'adminetu. Sauf indication contraire, sur le PC de TP, les commandes ne nécessitent pas de droit root.

Dans cette série de TP, nous allons mettre en place une solution et piloter une solution LAMP avec Ansible. Des serveurs DNS serviront une zone « lab.lan » permettant de faire l'association entre le nom des VMs et leur IP.

Dans ce premier temps, nous allons nous concentrer sur deux briques que nous utiliserons dans les TP suivant, à savoir le gestionnaire de packages Python `pipenv` et le moteur de templates Jinja2.

Le but de se TP sera d'installer le moteur de template Jinja2 et de générer un fichier de zone DNS grâce à lui.

1. Préparation de l'environnement de TP

Sur Ecampus, récupérez l'archive « TP1.tar.gz ». Celle-ci contient deux fichiers : un script Python et un fichier `Pipfile`. Décompressez cette archive dans un dossier TP1 ne contenant que ces deux fichiers. Ce dossier sera nommé dossier de travail plus tard dans le TP.

Décompressez cette archive dans un dossier adapté (bureau, documents, clef USB, etc.). À noter qu'il faudra bien sauvegarder et conserver ce dossier entre les séances puisque tout le dossier contiendra l'ensemble de votre travail.

Tout au long de ce TP, les commandes seront à lancer systématiquement depuis ce dossier. Pensez donc bien à utiliser la commande cd pour vous déplacer votre session shell dans le bon dossier !

Dans ce TP, et les suivants, je vous conseille d'utiliser Visual Studio Code (VSCode) comme IDE. L'avantage réside dans la coloration syntaxique, l'intégration d'un terminal dans l'interface et les nombreuses extensions disponibles pour faciliter le debug. Je vous conseille d'ailleurs d'installer l'extension YAML ! Si vous utilisez encore nano, c'est à vos risques et périls…

2. Installation de Jinja2 et de ses dépendances

Documentation utile :

- https://docs.python.org/fr/3/library/venv.html
- https://pipenv.pypa.io/en/latest/
- https://pipenv.pypa.io/en/latest/basics/#example-pipfile-pipfile-lock
- https://www.commentcamarche.net/faq/3585-bash-la-variable-d-environnement-path

2.1. Un peu de contexte…

Les distributions Linux comme Debian proposent un système de paquets, en l'occurrence APT, permettant de fournir facilement des logiciels mais aussi des bibliothèques de développement applicatif pour tout type de langage dont Python. Outre leur facilité d'utilisation, ces paquets fournis au travers APT sont réputés stables. À la fois dans le sens de la stabilité du paquet qui a eu le droit à une nouvelle passe de corrections de bugs avant d'entrer dans les dépôts Debian mais aussi et surtout une stabilité dans le temps : sur une installation Debian donnée, on peut passer sereinement les mises à jour. En effet, tant qu'on ne change pas de version de Debian, les paquets ne changent pas de version non plus.

Seuls des patchs de sécurités sont appliqués et il n'y aura donc pas de « breaking changes » cassant les logiciels s'appuyant dessus. Cela permet d'exploiter en toute sérénité des serveurs applicatifs en entreprise.

Cependant rien n'est jamais tout blanc ou tout noir ! La contrepartie est qu'il n'existe qu'une version de telle ou telle bibliothèque sur le système et elle n'est pas forcément récente ! Or, certaines bibliothèques (ou logiciels comme c'est le cas avec Ansible) évoluent très vite dans le temps et ne sont pas forcément compatibles entre elles. Cette rapidité amène son lot de « breaking changes », c'est à dire des modifications qui peuvent générer des problèmes sur des projets existants. Conséquences, certains développements ont des contraintes fortes sur les versions de bibliothèques et l'utilisation des bibliothèques présentes dans APT peuvent être insuffisantes car certains logiciels ne sont compatibles qu'avec certaines versions des bibliothèques (surtout dans le monde dev web, devops, etc...). Enfin, comment gérer sur une même machine plusieurs logiciels ayants des contraintes de versions de bibliothèques contradictoires ? Par exemple, le logiciel 1 a besoin d'une version de bibliothèque en version 2.X alors qu'un logiciel 2 a besoin de la version 3 minimum.

Bref, comment faire ? Plusieurs réponses existent à cette question et nous n'aborderons qu'une des réponses proposées pour Python. En effet, Python offre une solution élégante aux développeurs de logiciels dans ce langage : les `venv` ou `virtualenv` que l'on va coupler avec le gestionnaire de package Python `pip`. Les `virtualenv` permettent de créer des environnements virtuels isolés dans lesquels nous allons pouvoir installer des bibliothèques Python spécifiques. De son côté, le gestionnaire de package Python `pip3` qui permet d'installer des packages et logiciels Python localement tout en spécifiant pour chaque projet une version différente. Dans notre cas, ils seront installés dans notre `virtualenv`.

Dans cette partie, nous allons apprendre à utiliser `pip`, `virtualenv` et `pipenv` qui permet de concilier les deux facilement.

2.2. Installation de pip3 et pipenv

En `root`, installez `pip3` et `pipenv` :

```
apt update && apt -y install python3-pip pipenv
```

`pipenv` est un outil qui agit comme une surcouche au-dessus de `pip3`, il facilite la mise en place de `venv` et l'installation des packages dedans.

Créez un dossier « test » sur le Bureau. Dans ce dossier, créez un fichier nommé `Pipfile` (attention, la casse est importante !). Dans ce fichier, copiez le contenu suivant :

```
[[source]]
url = "https://pypi.python.org/simple"
verify_ssl = true
name = "pypi"
[packages]
cowsay = "*"
[dev-packages]
```

Ici, le « * » sert à indiquer que l'on souhaite n'importe quelle version. Mais on pourrait remplacer l'étoile par des indications de versions. NB : l'URL de la source restera toujours la même tout au long des TP. Il s'agit du miroir par défaut du gestionnaire de packages Python pip. Cette URL n'est modifiée que dans certains cas spécifiques que nous ne verrons pas dans ce TP. Ouvrez une invite de commande et allez dans le dossier test à l'aide de la commande cd.

Installez l'environnement virtuel à partir du `Pipfile` :

```
pipenv install
```

Enfin, vous pouvez entrer dans le `venv` :

```
pipenv shell
```

Observez la variable d'environnement `PATH` : quel est son contenu ? Pouvez-vous utiliser la commande `cowsay` dans le `venv` ? Pouvez-vous utiliser la commande `cowsay` dans un autre terminal (environnement par défaut) ?

Avec `pipenv`, on peut donc simplement indiquer une version de Python et un ensemble de packages Python désirés dans le `Pipfile` et travailler sur un environnement virtuel séparé de toute contrainte de versions de paquets !

2.3. Installation de Jinja2

Retournez dans votre dossier de travail (celui où vous aviez extrait l'archive `TP1.tar.gz`) Regardez le fichier `Pipfile` : que permet-il d'installer ?

Il ne vous reste plus qu'à rentrer dans votre environnement virtuel avec :

```
pipenv install
pipenv shell
```

NB : en cas de soucis, il suffit juste de modifier la version de Python par rapport à celle installée sur la machine !

3. Templating Jinja2

Allez reprendre votre fichier `data.yaml` réalisé au premier TD ! En cas de soucis, il y a une correction sur Ecampus…

Maintenant que nous avons d'un côté un exemple de ce à quoi doit ressembler la zone, et d'un autre côté une structure de données au format YAML contenant toutes les informations pour générer cette zone, voyons pour écrire un template de zone DNS en utilisant la syntaxe du moteur de template `jinja2`.

Pour vous aider, vous avez sur Ecampus, un dossier cheatsheets comprenant une fiche récapitulative de `jinja2`.

De même, vous pouvez vous référer à la documentation officielle de `jinja2`, partie template designer : https://jinja.palletsprojects.com/en/stable/templates/

À noter que vous aurez besoin de mobiliser vos compétences acquises lors des différents cours d'algorithmique et de programmation de l'année dernière !

3.1. Autour de Jinja2

Les questions ci-dessous vont vous servir d'étapes intermédiaires et vous guider.

1. Quelle est la syntaxe vous permettant de parcourir, sous forme d'une boucle, une liste/un tableau ?
2. Quelles est la syntaxe pour demander à `jinja2` d'afficher le contenu de la variable `var` ?
3. Si var est une map/un dict, quelle est la syntaxe pour accéder à la valeur de la propriété IPv4 de `var` ?
4. Quelle syntaxe utiliseriez-vous pour écrire une entrée NS si et seulement si la propriété `isAuthoritative` de var est vraie ?

Une fois des questions réalisées, nous allons pouvoir construire un template qui construise notre zone DNS à partir des données stockées au format YAML.

Créez un fichier template nommé `template.j2` dans le même répertoire que le script Python et le fichier `data.yaml`.

À chaque étape, vous pourrez utiliser le script `zone_generator.py` pour générer le rendu de votre template à partir du vos données dans `data.yaml`. Vous pouvez le tester dans un terminal dans un shell qui est dans la `venv` avec la commande :

```
python3 zone_generator.py
```

3.2. Début de la zone, mise en place du SOA

À partir de vos cours sur les DNS, mettez dans votre fichier `jinja2` l'enregistrement SOA (vous pouvez faire un copier-coller depuis un exemple existant). Repérez les parties du SOA correspondant spécifiquement au nom de la zone et remplacez-les par une balise `jinja2` permettant d'afficher le contenu de la variable zone.

Générez un rendu avec le script Python. Vous devriez voir le contenu de votre fichier template avec les balises remplacées par le contenu de la variable zone.

3.3. Enregistrements A et AAAA

À partir des réponses données aux questions préparatoires, ajoutez les enregistrements A et AAAA en positionnant une boucle sur le tableau `zone_dns`.

Générez un rendu avec le script Python. Vous devriez voir le contenu de votre fichier template avec les balises remplacées par une série d'enregistrement DNS A et AAAA.

3.4. Enregistrements NS

Entre l'enregistrement SOA et les enregistrements A/AAAA, ajoutez une boucle pour créer les enregistrements NS. Pour cela, il faut boucler sur le tableau `zone_dns` et ne créer un enregistrement NS si et seulement si la valeur `isAuthoritative` est vraie.

Une fois que vous avez pu voir que votre template génère une sortie valide dans le terminal, vous pouvez utiliser la commande suivante pour l'enregistrer dans un fichier, ici le fichier `zone.db` :

```
python3 zone_generator.py --out zone.db
```

Enfin, vous pouvez tester que la zone est valide grâce à `named-checkzone` :

```
sudo apt install -y bind9utils
sudo named-checkzone lab.lan zone.db
```

Si le template est bon, la zone sera valide… sinon corrigez !