UNIVERSITÉ CAEN NORMANDIE	BUT Réseaux et Télécommunications - Ifs	2024/2025
	R405	Clément Brisacier
	TP 4 - automatisation d'une infrastructure LAMP	

Généralités :

- se connecter en `adminetu` sous Debian
- récupérez votre dossier de travail habituel (starter kit)
- idéalement, utilisez un dépôt `git` ! à chaque modification de votre code fonctionnelle, pensez à faire un `commit` et un `push`
- de préférence, utiliser VSCode avec l'extension YAML en ouvrant votre répertoire de travail
- on a besoin de `pipenv` pour travailler :

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

- installer : Vagrant si ce n'est pas déjà fait

Maintenant que vous savez utiliser Vagrant pour le déploiement de VM avec un `Vagrantfile`, Ansible pour la création de playbooks et `git` pour la gestion des versions de code, il est temps de rentrer dans le vif du sujet ! On va progressivement réaliser des actions pour déployer automatiquement une solution LAMP.

1. Mise en place du service DNS

Documentation utile :

- playbook de base
- réutilisation de playbook
- variables de playbook
- templating Jinja2
- module package
- module template
- module shell
- module service
- module blockinfile

Pour cette partie et les suivantes, nous travaillons sur un unique playbook `setup_infra.yml` :

- à la racine de votre répertoire de ce TP, créez le fichier `setup_infra.yml`
- la première ligne de ce fichier sera 3 tirets : `---`
- pour exécuter ce playbook, il faudra utiliser la commande :

```
ansible-playbook -i inventory/hosts.ini setup_infra.yml
```

1.1. Installation de bind9 sur les DNS

1. Dans le playbook `setup_infra.yml` :
   - Créez un groupe de tâches à appliquer aux hôtes du groupe `dns`
   - La première tâche à écrire est l'installation de `bind9` à l'aide du module `package`
2. Exécutez une première fois votre playbook !
   - En cas de problème lors de l'exécution, déboguez le problème puis relancez le playbook. Le playbook s'est bien exécuté lorsqu'il n'y a plus aucun hôte avec des tâches en `unreachable` ou `failed`

- Notez le nombre de tâches en `ok` et en `changed`
- Une fois que le playbook est terminé, vérifiez que `bind9` est bien installé et fonctionnel sur les serveurs du groupe `dns` en vous connectant manuellement en SSH dessus.

3. Dans le `Vagrantfile`, modifiez le nombre de serveurs DNS à deux et démarrez `dns-2` avec Vagrant.
4. Ajoutez ce serveur DNS dans votre inventaire Ansible dans le bon groupe. Vérifiez que l'hôte est bien joignable avec `ansible -m ping -i inventory/hosts.ini dns-2`
5. Relancez le playbook :
   - Comparez le nombre de tâches en `ok` vs `changed` entre les deux hôtes. Qu'en déduisez-vous ?
   - Que se passe-t-il lorsqu'on exécute ce playbook sur des hôtes qui ont déjà bind9 d'installé ? Ansible va-t-il réinstaller `bind9` à chaque fois ?

1.2. Reconfiguration des VM pour utiliser les DNS internes

Maintenant que nous avons des serveurs DNS opérationnels, et même s'ils ne sont pas encore configurés avec la zone DNS interne de notre maquette, ils sont fonctionnels pour résoudre des noms de domaine publiques. Nous devons donc reconfigurer l'ensemble de nos VMs pour utiliser ces serveurs DNS internes et non les serveurs DNS fournis par défaut par Vagrant.

S'agissant ici de configurations visant à contourner les configurations automatiques de Vagrant (rappelez-vous qu'il s'agit d'un outil de maquettage et non un outil pour déployer des VMs de productions), les tâches de configurations à effectuer sont déjà écrites dans le fichier `tasks/set_dns.yml`

- Créez un nouveau bloc de tâches à appliquer à l'ensemble des hôtes
- Comme unique tâche, utilisez le module `import_tasks` pour importer les tâches du fichier `tasks/set_dns.yml`

Ce fichier de tâches utilise la variable `domaine_lab` pour définir le suffixe DNS des VMs. Configurez cette variable avec la valeur `lab.lan` dans le fichier de variable appliqué à tous les hôtes dans l'inventaire
- Exécutez votre playbook à nouveau et gardez de côté le contenu du « play recap »
- Vérifiez que les VM utilisent bien les serveurs `dns-1` et `dns-2` comme DNS
- Exécutez une deuxième fois ce playbook et gardez de côté le contenu du « play recap »

Attardons-nous un petit peu sur le contenu du fichier de tâches `tasks/set_dns.yml` :
- Lisez le fichier et décrivez, concrètement, les actions que ces tâches réalisent
- À quoi servent les mots clefs « register » et « when » ?
- Maintenant, comment expliquez-vous la différence entre les « play recap » des deux exécutions du même playbook comme vu juste avant ?

1.3. Génération de la zone DNS avec Jinja2

Maintenant que nous avons des serveurs DNS opérationnels et utilisés par les différents serveurs de notre maquette, nous allons provisionner la zone DNS `lab.lan` dans nos serveurs DNS. Le but sera qu'à chaque exécution du playbook `setup_infra.yml`, la zone DNS soit re-générée avec les informations à jours. Pour cela, nous générerons le fichier de zone DNS à l'aide d'un template Jinja2. Tous les serveurs DNS seront masters de la zone `lab.lan`.

Afin de configurer la zone DNS `lab.lan`, nous allons nous appuyer sur le template du TP1 que nous allons légèrement adapter afin d'utiliser les données variables présentes dans l'inventaire Ansible plutôt que le fichier `data.yaml`.

1.3.1. Gestion du numéro de série

Pour rappel, le numéro de série de la zone doit être changé à chaque modification de celle-ci avec un nombre de plus en plus grand. Afin de contourner simplement ce problème, je vous propose d'utiliser le timestamp Unix du moment de la génération de la zone comme numéro de série. Ce nombre est codé sur 32 bits comme le numéro de série DNS et ne fait qu'augmenter (du moins jusqu'en 2038).

En bash, la commande suivante permet d'obtenir le timestamp de l'heure actuelle : `date +%s&`

1. Dans le bloc de tâche exécutées sur les serveurs DNS, rajoutez une tâche de type `shell` permettant d'exécuter cette commande. Enregistrez (`register`) dans ce résultat dans une variable appelée `timestamp`
   - Le résultat de la commande `shell` (sortie standard de la commande `date`) sera alors accessible dans `timestamp.stdout`
2. En sachant qu'Ansible exécute les playbooks en parallèle sur plusieurs machines cibles en même temps dans la limite du nombre de tâches parallèles paramétrées dans le fichier `ansible.cfg`, quel est le risque si chaque hôte utilise son propre timestamp du moment où la commande date est réellement exécutée ?
   - Pour contourner ce problème, je vous propose de regarder du côté de l'option `run_once`. En quoi cette option va nous aider dans notre cas ?
   - Modifiez le playbook pour utiliser cette option sur la tâche générant le timestamp

1.3.2. Template Jinja2

Rappel de quelques variables disponibles :

Nom de la variable	Contenu
`domaine_lab`	Nom de la zone / suffixe DNS
`groups['<nom_groupe>']`	Liste des tous les hôtes du groupe `<nom_groupe>`
`hostvars[<nom_hôte>]['ansible_host']`	IP de l'hôte `<nom_hôte>`
`groups['dns']|first`	Nom du premier hôte hôte du groupe DNS
`timestamp.stdout`	Timestamp généré par la tâche précédente

1. Copiez le fichier template Jinja2 que vous avez écrit durant le TP 1 dans le fichier `templates/labzone.j2` puis adaptez le :
   - Le nom de la zone doit être uniquement présent sous forme de variable
   - Le serveur DNS master spécifié dans la SOA est le premier serveur DNS du groupe DNS
   - Les entrées NS et A sont générés dans deux boucles « for » parcourant respectivement la liste des hôtes des groupes `dns` et `all` de l'inventaire (NB : en plus de la documentation Jinja2 indiquée dans les liens en début de cette partie, vous pouvez regarder ce qui a été fait dans le fichier `templates/lb-apache2-conf.j2`)
2. Enfin, dans le playbook `setup_infra.yml`, pour les hôtes `dns`, ajoutez une nouvelle tâche utilisant le module template pour générer la zone. Le fichier de zone devra appartenir à l'utilisateur `bind` et au groupe `bind`.
3. Exécutez le playbook et vérifiez sur l'un des serveurs DNS le fichier de zone ainsi généré !

1.3.3. Déclaration automatique de zone

Maintenant que nous avons un fichier de zone généré automatiquement par Ansible, nous devons également ajouter une tâche dans notre playbook pour aller modifier le fichier de configuration `/etc/bind/named.conf.local`

1. Tout d'abord, supprimez les configurations dans le fichier `/etc/bind/named.conf.local` que vous auriez pu faire par vous-même ou supprimez puis recréez les VM DNS dans Vagrant
2. Ajoutez une tâche aux serveurs `dns` utilisant le module `blockinfile` pour ajouter automatiquement la déclaration de la zone DNS dans `/etc/bind/named.conf.local` (NB : en complément de la documentation Ansible, vous pouvez regarder ce qui a été fait dans le fichier `tasks/set_dns.yml`)
3. Ajoutez une seconde tâche à ces mêmes serveurs utilisant le module `service` d'Ansible pour recharger la configuration du service `bind9`
4. Vérifiez le bon fonctionnement des serveurs DNS en testant de pinger les VM `web` depuis la VM `lb-1`

1.3.4. Utilisation des variables dans le playbook

Si vous ne l'avez pas fait lors de la sous-partie précédente, modifiez votre playbook afin que le nom de la zone DNS porte automatiquement le nom de notre zone interne stocké dans la variable `domaine_lab`

2. Mise en place du service load-balancer Web

- https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html
- https://docs.ansible.com/ansible/latest/modules/apache2_module_module.html
- https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html
- https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html

Dans cette partie, nous allons installer et configurer le load-balancer. Le template de configuration de celui-ci est fourni dans le « Starter Kit », on ne verra pas ici son fonctionnement mais si vous êtes curieux c'est assez compréhensible !

2.1. Installation du lb

1. Créez un nouveau groupe de tâches pour le groupe d'hôtes `lb`
2. Pour ce groupe, ajoutez une tâche pour installer `apache2`
3. Après l'installation d'Apache, nous devons activer plusieurs de ses modules : `proxy_http`, `proxy_http2`, `proxy_balancer` et `lbmethod_byrequests`. Pour cela nous allons utiliser le module Ansible `apache2_module`. Ce module Ansible n'acceptant qu'un seul nom de module Apache à la fois, vous allez utiliser une boucle Ansible pour activer ces 4 modules Apache dans la même tâche.
   - En vous référant à la documentation, vous verrez que ce module fait partie de la collection de module `community.general`. Installez ces modules à l'aide de la commande `ansible-galaxy` qui vous est proposée dans la documentation
   - Écrivez cette tâche puis exécutez votre playbook et regardez l'affichage de la tâche dans la sortie standard d'Ansible

2.2. Configuration du lb

4. Toujours dans les tâches pour les `lb`, ajoutez une tâche utilisant le module `template` pour générer le fichier de configuration à partir de `templates/lb-apache-conf.j2`
   - Le fichier de conf généré sera enregistré au chemin `/etc/apache2/sites-available/000-default.conf`
   - Le propriétaire et le groupe du fichier sera `www-data`
5. Ajoutez une autre tâche pour recharger le service `apache2`
6. Faites-en sorte, à l'aide d'une variable sur la tâche template et d'une condition (« when ») sur la tâche service, que la tâche `reload` s'exécute uniquement lorsque le fichier de configuration du `lb` change

7. Le fichier de configuration des `lb` change lorsque le nombre de serveurs web change. Pour tester le bon fonctionnement de votre playbook, vous pouvez ajouter puis supprimer des serveurs web en modifiant la bonne variable dans le Vagrantfile (cf début du TP). N'oubliez pas de mettre à jour également votre inventaire Ansible !

3. Mise en place des serveurs web

- https://docs.ansible.com/ansible/latest/modules/file_module.html
- https://docs.ansible.com/ansible/latest/modules/copy_module.html

Dans cette partie, nous installerons Apache2 et PHP sur les serveurs `web` et nous téléchargerons notre « site web » par défaut (dans le cas de ce TP, un simple fichier php) vers l'ensemble des serveurs web.

Ajoutez un groupe de tâches à appliquer aux hôtes du groupe `web` :

- Installation de Apache2 & PHP
- Suppression du fichier `/var/www/html/index.html`
- Copie du fichier `files/index.php` vers le chemin `/var/www/html/index.php` sur les serveurs `web`. Le fichier `index.php` appartiendra à l'utilisateur `www-data` et au groupe `www-data`

Encore une fois, testez le bon fonctionnement de votre Playbook !

4. Test du déploiement final

Si tout est bien configuré, vous pouvez tester le bon fonctionnement en vous connectant à l'URL http://127.0.0.1:8080 depuis un navigateur de votre PC. Une redirection de port effectuée par Vagrant renvoie vers le `load-balancer`.

Si vous rechargez plusieurs fois la page, vous verrez le nom des différents serveurs Web défiler les uns après les autres (sauf si vous n'en avez qu'un seul de configuré).

Pour tester le bon fonctionnement de notre Playbook, et tout l'intérêt d'Ansible, nous allons augmenter le nombre de serveurs : 2 DNS et 4 serveurs web.

1. Modifiez les variables du `Vagrantfile` pour mettre 2 serveurs DNS et 4 serveurs WEB puis lancez la commande `vagrant up`
2. En attendant que les nouvelles VM démarrent, mettez à jour votre inventaire Ansible en ajoutant les nouvelles machines (avec leur IP et la bonne clé SSH pour se connecter)
3. Une fois que Vagrant a fini de démarrer toutes les VM, lancez votre playbook
4. Observez qu'il ne procède aux opérations d'installation et de configurations uniquement pour les nouvelles VM et les configurations qui ont changé sur les anciennes VM (comme la zone DNS ou la configuration du LB par exemple)
5. Une fois le playbook terminé, retournez sur http://127.0.0.1:8080 et rechargez plusieurs fois la page pour voir apparaître les 4 noms des 4 serveurs web au fur et à mesure des rechargements de la page

On pourrait rajouter 5, 6, 9 ou plus serveurs web. Grâce à Ansible, la mise en place de nouveaux serveurs est très simple car entièrement automatisée ! On a donc une infrastructure qui peut évoluer dynamiquement et supporter la montée en charge en cas de besoin.

5. Pour aller plus loin...

5.1. Segmentation du playbook en rôles

- https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html

Les rôles permettent de créer un ensemble de tâches modulaires et réutilisables par de multiples playbook. Même si le scénario de ce TP est plutôt simpliste, il est toujours bien de créer les rôles dès le début même lorsque les scénarios sont encore simples.

1. Créez 3 dossiers rôles dans le dossier roles : `dns`, `web`,

`lb`

1. Dans chacun des 3 rôles, créez un dossier `tasks` ; et dans ce dossier un fichier `main.yml` qui débutera par « — »
2. Dans les dossiers des rôles `lb` et `dns`, créez un dossier `templates` où vous déplacerez les templates associés à ces rôles
3. Dans le dossier du rôle `web`, créez un dossier `files` où vous déplacerez le fichier `index.php` utilisé dans ce rôle
4. Maintenant, déplacez les tâches spécifiques associées aux différents rôles dans les fichiers `tasks/main.yml` que nous avons créé dans les parties précédentes
5. Enfin, appliquez ces rôles dans le playbook `setup_infra.yml` à la place des tâches spécifiques

C'est plus propre en termes d'organisation ! Cela permet plus facilement de « grossir » le playbook dans le temps, surtout en cas de réutilisation des plays dans le futur.

5.2. Handlers

- https://docs.ansible.com/ansible/latest/user_guide/playbooks_handlers.html
- https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html

Dans les rôles `lb` et `dns`, nous sommes amenés à recharger, respectivement, les services `apache2` et `bind9`. Dans le rôle `lb`, nous réalisons cette action uniquement en cas de changement du fichier de configuration `apache2`. Pour cela nous utilisons une condition (clause `when`). Pour notre cas d'usage, cela fonctionne mais s'il y a plusieurs fichiers de configuration, cela pourrait vite devenir ingérable.

D'ailleurs, pour le rôle `dns`, nous modifions deux fichiers. Si nous voulons recharger `bind9` si l'un au moins des deux fichiers est modifié, alors il faut une clause `when` avec deux cas possibles. Si on avait 1000 zones... imaginez le bazar !

Pour ce de figure, Ansible propose une solution avec les `handlers`.

Dans l'idée, des tâches peuvent notifier des `handlers`. À la fin du `play`, si certains de ces `handlers` ont été notifiés, ils s'exécutent une fois.

Dans le cas des rôles, les `handlers` sont des tâches classiques si ce n'est qu'elles sont stockées dans le fichier `main.yml` du dossier `handler`.

5.2.1. Handler DNS

Basculez la tâche de rechargement du service `bind9` dans le `fichier handler/main.yml` du rôle `dns`. Puis, modifiez les deux tâches `template` et `blockinfile` pour qu'elles notifient la tâche de rechargement du service `bind9`.

Lancez le playbook plusieurs fois, que voyez-vous ?

5.2.2. Handler LB

Réalisez les mêmes changements dans le rôle `lb` : déplacement de la tâche de rechargement du service `apache2` et notification depuis la tâche de `template` du fichier de configuration `apache2`.

Lancez le playbook une fois. Le `handler` du rôle `lb` se lance-t-il ? Pourquoi ?

Ajoutez un serveur web en incrémentant la variable `$num_web` dans le `Vagrantfile` puis lancer la nouvelle VM à l'aide de la commande `vagrant up`. Puis ajoutez cet hôte à l'inventaire.

Relancez le playbook. Le `handler` du rôle `lb` se lance-t-il ? Pourquoi ? Relancez une dernière fois le playbook. Le `handler` du rôle `lb` se lance-t-il ? Pourquoi ?