ECOLE SUPERIEURE D'INGENIEURS DE RENNES

Université de Rennes

Introduction à Ansible

29 août 2023

Introduction

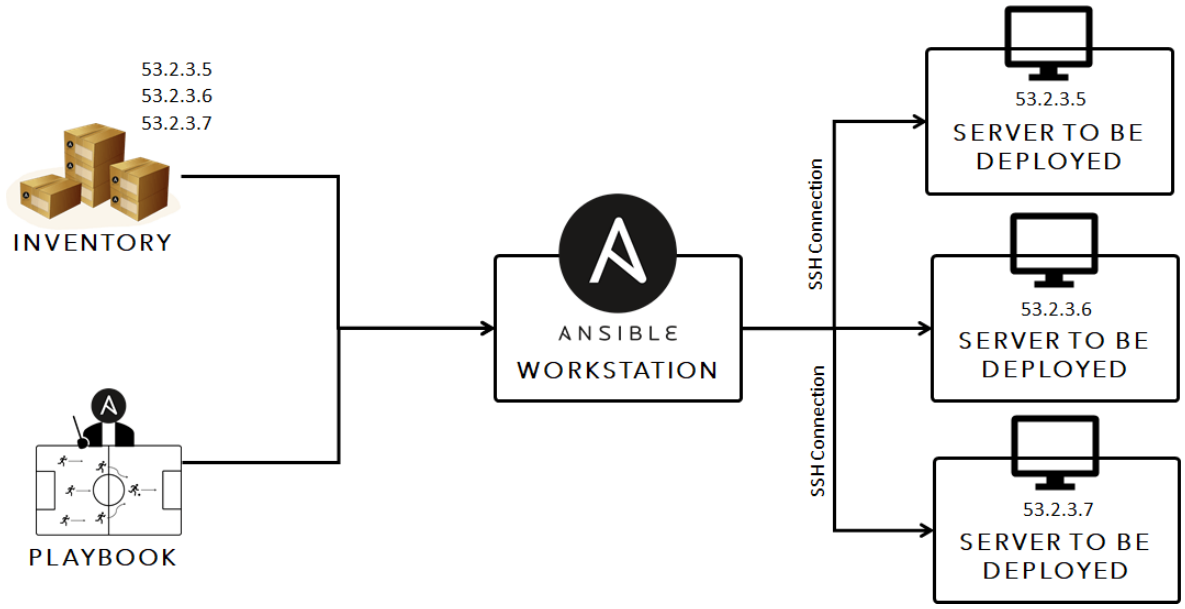

Figure 1: Ansible Architecture

Qu'est-ce qu'Ansible ? Il s'agit d'une plateforme de gestion de configuration et d'automatisation open source. Ansible permet notamment de gérer des configurations et d'approvisionner des machines ou bien même déployer des programmes de manière automatisée et multi-environnements pour plus de flexibilité. Les cas d'utilisations sont multiples et variés, que ce soit pour l'automatisation cloud hybride, en passant par l'automatisation réseau jusqu'à l'automatisation sécurité.

L'architecture d'Ansible est composée de deux parties : Ansible Workstation et Ansible Nodes. Ansible Workstation correspond à la machine qui va permettre de gérer les configurations et d'automatiser les tâches. Ansible Nodes correspond aux machines qui vont être gérées par la partie Ansible Workstation. La Workstation comprend deux artéfacts : l'Inventaire et un/plusieurs Playbook(s). L'inventaire est un fichier qui contient la liste des machines à gérer. Tandis que les playbooks sont des fichiers YAML qui contiennent les différentes tâches d'automatisation à effectuer sur les machines renseignées dans l'inventaire.

Ansible est agentless, ainsi il n'y a rien besoin d'installer sur les nœuds cibles, il suffit simplement d'avoir un accès SSH vers ces différents nœuds pour que les tâches puissent être effectuées.
Enfin, Ansible est très intéressant d'un point de vue modularité et réutilisabilité notamment grâce aux nombreux modules pré-existants, mais également grâce à une nouvelle notion : les Roles. Les rôles permettent d'apporter de la clareté dans les playbooks et de diviser ces derniers en sous-parties qui auront chacune un rôle bien défini. Ces rôles très spécifiques peuvent alors être réutilisés dans n'importe quels playbooks, mais sont aussi très largement partagés sur GitHub, ce qui permet de gagner du temps et de ne pas réinventer la roue.

Installation d'Ansible

Disclaimer : Ansible est Unix/Linux-based, pour les utilisateurs Windows il faudra donc passer par WSL :)

Il vous faudra tout d'abord vous assurer d'avoir pip sur votre machine pour pouvoir installer Ansible :

```
python3 -m pip install --user ansible
```

Si certains ont déjà Ansible d'installé, une mise à jour s'impose :

```
python3 -m pip install --upgrade --user ansible
```

Il ne reste plus qu'à vérifier que l'installation s'est bien déroulée !

```
ansible --version
```

Un dernier point à effectuer, avant de pouvoir commencer, est d'aller modifier le fichier de configuration d'Ansible qui se situe dans */etc/ansible/ansible.cfg* et d'y ajouter :

```
[defaults]
host_key_checking = False
```

pour éviter d'avoir à valider les clés SSH des nœuds cibles à chaque fois que l'on souhaite exécuter un playbook (surtout que nous risquons d'en créer et détruire quelques uns).

Déploiement de nœuds

Comme dit plus tôt, il va falloir déployer des nœuds qui seront les terrains de jeux de nos playbooks ! Le TP n'étant pas dédié au déploiement de VMs, vous allez donc pouvoir récupérer un script à l'adresse suivante : https://github.com/MariusLD/tp_ansible.git

Pour exécuter le script il faudra cependant disposer de : *podman*.

```
sudo apt-get -y install podman
```

Il est également nécessaire de disposer d'une clé SSH pour vous connecter aux nœuds avec Ansible. Le nom de la clé par défaut appelée par le script est *id_rsa*. Si vous ne possédez pas de clé portant ce nom, petit rappel, vous pouvez la générer avec :

```
ssh-keygen -t rsa -b 2048
```

Vous devriez désormai pouvoir créer vos nœuds aisément en effectuant la commande :

```
./deploy_vm_tp.sh -c <nombre de machines à générer>
```

La liste des machines créées sera affichée à la fin de l'exécution du script, où vous pourrez également voir l'adresse IP attribuée à chacune d'entre elles. Vous pourrez stopper les machines avec la commande :

```
./deploy_vm_tp.sh -t
```

Avec la possibilité de les relancer par la suite :

```
./deploy_vm_tp.sh -s
```

Ou bien (toutes) les supprimer définitivement avec :

```
./deploy_vm_tp.sh -d
```

Toutes ces options sont également disponibles en exécutant le script sans arguments.

Création de l'inventaire

Pour cette étape, l'objectif est de définir un fichier qui va donner les instructions aux playbooks pour savoir « chez qui » (hosts ou groupes) ils vont devoir effectuer les tâches. Il peut alors s'agir d'un nœuds ou bien d'un groupe de nœuds.
Par défaut le groupe **all** est le groupe racine qui comprendra tous nos nœuds.
Il existe différents formats d'inventaire : init, yaml, json... Nous allons d'abord voir la structure de l'inventaire au format init, puis nous passerons au format yaml qui est plus modulable et plus facile à manipuler.
Avec votre éditeur de texte préféré vous allez pouvoir créer un fichier nommé « inventory.yml » où la structure par défaut sera :

```
[all]
ansible_host_1: <adresse IP du nœud ansible_host_1>
ansible_host_2: <adresse IP du nœud ansible_host_2>
```

L'inventaire ci-dessus indique donc aux playbooks qu'il y a deux machines à traiter.

Les groupes permettent d'isoler certains nœuds pour leur attribuer un playbook spécifique. Pour créer un groupe il suffit de définir un nom entre crochets et de lister les nœuds qui en feront partie, par exemple nous pouvons avoir :

```
[parent1]
ansible_host_1: <adresse IP du nœud ansible_host_4>

[enfant1]
ansible_host_2: <adresse IP du nœud ansible_host_1>
ansible_host_3: <adresse IP du nœud ansible_host_2>

[enfant2]
ansible_host_4: <adresse IP du nœud ansible_host_3>

[enfant3]
ansible_host_5: <adresse IP du nœud ansible_host_3>

[parent1:children]
enfant1
enfant2

[enfant2:children]
enfant3
```

Ici **parent1** contient :
- les nœud *ansible_host_1*

enfant1 contient :
- le nœud *ansible_host_2*
- le nœud *ansible_host_3*

enfant2 contient :
- le nœud *ansible_host_4*

enfant3 contient :
- le nœud *ansible_host_5*

parent1 contient :
- le groupe **enfant1**
- le groupe **enfant2**

enfant2 contient :
- le groupe **enfant3**

Nous retrouvons alors une structure en forme d'arbre, où les groupes peuvent être imbriqués les uns dans les autres.

Pour de petites inventaires le format init est suffisant, mais pour des inventaires plus conséquents il est préférable d'utiliser le format yaml.
Pour cela il suffit de créer un fichier *inventory.yml*. Le début de l''équivalent structurel de l'exemple précédent est :

```
---
all:
  children:
    parent1:
      hosts:
        ansible_host_1:
          ansible_host:
```

Q1) Complétez cet inventaire pour correspondre au format init et créez 5 nouveaux nœuds, ajoutez-les à l'inventaire en reprenant la structure ci-dessus.
Puis testez votre inventaire grâce à la commande :

```
ansible <groupe> -m ping -i inventory.yml
```

Attention à l'indentation en YAML, un espace de moins ou de plus peut changer la signification d'un fichier de configuration ! Si tout se déroule comme prévu, vous ne verrez que les ***pongs*** *des nœuds faisant partie du groupe renseigné.*

Pour la suite, vous pouvez supprimer tous les nœuds créés pour réduire le temps d'exécution des prochains exercices, et en déployer un nouveau. Pour ce qui est de l'inventaire, maintenant que vous avez compris le principe, vous pouvez soit le refaire, soit profiter de l'option -a du script *deploy_vm_tp.sh* pour vous le générer automatiquement dans */ansible_dir/00_inventaire.yaml*

Prise en main des playbooks Ansible

Maintenant que nous avons survolé les principales notions d'Ansible, nous allons pouvoir passer à la pratique. Un playbook Ansible est un ficher YAML où nous allons devoir renseigner : les nœuds cibles et les tâches à effectuer.
Pour ça, un playbook Ansible aura la structure de base suivante :

```
---
- hosts: <groupe>
  tasks:
    - name: <nom de la tâche>
      <module/tâche>: <paramètres>
    - name: <nom de la tâche>
    [...]
```

Ansible playbook cheat sheet

Les modules suivants ne sont pas forcément tous nécessaires pour la suite mais restent des modules très utiles à connaître :

- **apt/yum/dnf** : pour l'installation de packages : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/apt_module.html

```
- name: Installation des packages listés
  apt/yum/dnf:
    name:
      - package1
      - package2
      - package3
    state: present
```

- **copy** : permet de copier des fichiers : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/copy_module.html

```
- name: Copie d'un fichier en local ou sur la machine distante, vers la machine distante
  copy:
    src: /path/du/fichier/à/copier
    dest: /path/sur/la/machine/distante
    owner: <utilisateur>
    group: <groupe>
    mode: <autorisations (valeur numérique ou via les droits)>
```

- **fetch** :

```
- name: Récupération d'un fichier
  fetch:
    src: /path/du/fichier/à/récupérer
    dest: /path/où/save/le/fichier
```

- **file** : permet de créer, supprimer ou modifier des fichiers : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/file_module.html

```
- name: Change les permissions d'un fichier
  file:
    path: /path/du/fichier
    owner: <utilisateur>
```

```
    group: <groupe>
    mode: <autorisations (valeur numérique ou via les droits)>
```

- **service/systemd** : permet de gérer les services : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/service_module.html - https://docs.ansible.com/ansible/latest/collections/ansible/builtin/systemd_module.html

```
-  name: Après avoir installé un package, il faut un module pour le lancer, service permet donc de start/stop/reload des packages
   service/systemd:
    name: <service à gérer>
    state: <started/stopped/restarted>
```

- **command** : permet d'exécuter des commandes, préciser des arguments, et bypass le shell : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/command_module.html

Si vous avez tout de même besoin d'une fonctionnalité du shell, vous pouvez vous référer au module shell : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/shell_module.html

```
- name: Exécution d'une commande
  command: <commande>
```

On peut également vouloir définir des variables dans notre playbook, on peut soit les déclarer en haut de ce dernier :

```
---
- hosts: <groupe>
  vars:
    variable_test: test
    [...]
  tasks:
```

Il existe également les variables multi-dimensionnelles :

```
---
- hosts: <groupe>
  vars:
    type: web
    service:
      web:
        name: apache
        rpm: httpd
      db:
        name: mariadb
        rpm: mariadb-server
    [...]
```

Une bien meilleure pratique est de définir un fichier de variables à part, communément appelé *vars.yml* qui contiendra les variables sous la forme :

```
---
variable_test: test
[...]
```

Puis l'intérêt est de l'inclure dans le playbook :

```
---
- hosts: <groupe>
  vars_files:
    - vars.yml
  tasks:
  [...]
```

Il est alors possible d'appeler les variables aisément dans le playbook : *"{{ variable_test }}"*, par exemple pour l'exemple précédent sur la variable multi-dimensionnelle *service* → "{{ service[type]['name'] }}" a pour valeur *apache*.
On peut également vouloir les définir lors de l'exécution du playbook grâce au module set_fact : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/set_fact_module.html

```
- name: Initialisation de variables
  set_fact:
    <nom de la variable>: <valeur>
    [...]
```

Il est alors possible d'utiliser le module debug pour vérifier l'existence d'une variable : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/debug_module.html

```
- name: Affichage d'une variable
  debug:
    msg: "Blabla {{ variable }} blabla"
  # var: variable
```

Dernier point à ce sujet, il est également possible de définir des variables d'inventaire, soit pour limiter l'accessibilité de ces variables à certains nœuds, soit pour faire varier les valeurs de ces variables en fonction des nœuds. Pour reprendre l'exemple d'inventaire vu précédemment :

```
---
all:
  children:
    parent1:
      hosts:
        ansible_host_1:
      children:
        enfant1:
          hosts:
            ansible_host_2:
            ansible_host_3:
          vars:
            var1: "Hello"
        enfant2:
          hosts:
            ansible_host_4:
          var2: "World!"
```

Création d'un premier playbook

Q2) Créez un ficher playbook.yml *ainsi que son fichier de variables* vars.yml *(qu'il ne faudra pas oublier d'importer au début du playbook !) puis un fichier texte vierge. L'objectif sera maintenant de créer un nouvel utilisateur* cidre *avec un mot de passe classique, puis de créer un dossier /tmp/cidre qui lui appartiendra et donnez-lui les droits d'écriture sur ce dossier, ensuite copiez-y le fichier texte.*
L'intérêt ici étant également de n'avoir aucune variable définie dans le playbook, mais uniquement dans le fichier de variables !

Tips : il se peut que certaines tâches dans vos playbooks nécessitent une élévation de privilège (sachant que par défaut Ansible se connecte via SSH en tant que non-root user). Pour palier à cette contrainte, il faut utiliser le module become *: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/become_module.html soit pour une tâche spécifique, soit sur l'entièreté du playbook afin de devenir un superuser :*

```
---
- hosts: <groupe>
  become: true
  tasks:
  [...]
```

Vous allez maintenant pouvoir lancer votre premier playbook. Pour cela il suffit d'exécuter la commande :

```
ansible-playbook -i inventory.yml playbook.yml
```

Encore une fois, si tout se déroule comme prévu vous pourrez voir Ansible effectuer les différentes tâches sur les différents nœuds, l'un après l'autre. Une fois l'exécution terminée, vous pouvez vérifier que tout s'est bien passé en vous connectant sur l'un des nœuds cibles, puis en vérifiant la présence du fichier texte dans /tmp/cidre.

Q2.1) Exercice similaire, mais nous allons maintenant introduire les boucles d'Ansible : with_items.
Le but sera de créer plusieurs répertoires dans /tmp/(cidre1, cidre2...).

Q2.2) Même exercice, mais cette fois-ci nous allons itérer sur différents dictionnaires, pour copier des fichiers dans différents répertoires. On voudra par exemple avoir le fichier1 dans cidre1, le fichier2 dans cidre2, etc.

Approfondissement du playbook

Toutes les **propositions** de solution sont accessibles sur le repo cloné, dans la section : *solutions*
Par souci de clarté, il est recommandé de créer un nouveau playbook pour chaque question (et cela va grandement vous faciliter la tâche pour la dernière question à traiter !)
Maintenant que vous avez compris les bases structurelles d'un playbook, vous allez pouvoir effectuer des tâches un peu plus spécifiques avec Ansible.

Déploiment de clé SSH

Q3) Créez un nouvel utilisateur sur les nœuds cibles. Générez une clé SSH en local puis déployez la clé publique en tant que clé autorisée sur les nœuds cibles.
Pour cela vous aurez besoin de différents modules Ansible :

- *user : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/user_module.html*
- *openssh_keypair:https://docs.ansible.com/ansible/latest/collections/community/crypto/openssh_keypair_module.html*
- *authorized_key:https://docs.ansible.com/ansible/latest/collections/ansible/posix/authorized_key_module.html*

Tips : La clé SSH doit être générée localement, il faut donc trouver un moyen d'indiquer au playbook que la tâche doit être effectuée en local et non sur les nœuds cibles.

(Cependant générer une clé localement nécessite un mot de passe root. Il faut donc ajouter une option à votre ligne de commande pour fournir ce mot de passe manuellement, jetez un coup d'œil à : https://docs.ansible.com/ansible/2.8/cli/ansible-playbook.html)

Découverte du module APT d'Ansible et gestion de service

Pour la prochaine étape, il s'agira d'installer un service grâce au module *apt* d'Ansible. Ici nous souhaitons tout d'abord utiliser un paramètre du module apt qui permet d'update et upgrade le cache des packages : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/apt_module.html

Q4.1) Installez le service apache2 *sur les nœuds cibles. Il y a deux manières de procéder :*

- *Soit vous pouvez simplement récupérer le package depuis les dépôts par défaut, en utilisant le module apt lui-même, voir le cheat sheet pour l'installation de package, le paramètre state est important !*
- *Soit vous pouvez ajouter le dépôt localement et installer le module depuis ce dépôt. Dans ce second cas il faudra faire usage du module apt_repository https://docs.ansible.com/ansible/latest/collections/ansible/builtin/apt_repository_module.html (le repo d'apache2 est ppa:ondrej/apache2)*

Q4.2) Une fois le service installé, il faut le lancer. Pour cela il faut utiliser le module service *d'Ansible : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/service_module.html. Une option requise pour assurer la liveness du service est le démarrage au boot de celui-ci. Cette option est indiquée dans la documentation.*

Q4.3) Nous pouvons également avoir envie de modifier le fichier de configuration du service. Nous découvrirons plus tard les templates qui facilitent grandement ce genre de modification, mais pour l'instant nous allons nous contenter de découvrir le module lineinfile *: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/systemd_module.html. Lineinfile se base sur les expressions regex :* ***regexp*** *ou bien sur une chaîne de caractères écrite à la main :* ***search_string*** *et remplace les occurences par* ***line****.*

Q4.4) Après ces modifications faites à la main, il faut penser à relancer le service pour qu'elles soient appliquées. Il faut donc utiliser une nouvelle fois le module apt, avec une légère variation pour répondre à notre besoin.

Voilà, vous venez de lancer votre premier service avec Ansible ! On peut être amené à se demander si le fait de lancer le service pour le redémarrer ensuite n'est pas redondant en termes de tâches, et la réponse est oui. À la place on peut ajouter une option au module systemd qui permet de recharger automatiquement le service si le fichier de configuration a été modifié grâce à un daemon.

Les handlers

Les handlers ne sont rien d'autre que des tâches qui sont exécutées que si une autre tâche a été exécutée précédemment et qu'elle s'est terminée correctement.
Par exemple, si nous souhaitons redémarrer un service après avoir modifié un fichier de configuration, il faut s'assurer que la modification du fichier de configuration s'est bien déroulée avant de redémarrer le service.
Les handlers doivent être rédigés sous cette forme dans le playbook :

```
---
- hosts: <groupe>
  become: true
  tasks:
  [...]
  handlers:
    - name: <nom du handler>
      <module/tâche>: <paramètres>
```

Pour cela, il faut utiliser le module *notify* : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/notify_module.html
Ce module permet d'appeler un handler, qui sera alors appelé après cette tâche.

*Q5) Reprenez le playbook d'installation et de lancement d'*apache2*, créez un handler qui permettra de redémarrer apache2, faites une modification du fichier de configuration qui se situe dans* /etc/apache2/apache2*.conf, puis notifiez le handler créé précédemment.*
Vérifiez que le service a bien été redémarré, vous pouvez même tenter de relancer le playbook plusieurs fois avec d'autres modifications pour vous assurer que toutes les modifications sont bien appliquées à chaque itération.

Protection des informations confidentielles

Il est possible que certaines tâches nécessitent des informations confidentielles, comme des mots de passe, des clés SSH, etc.
C'est pour ça qu'Ansible met à disposition une feature qui est *Ansible Vault* et qui permet de chiffrer ces données.
Pour cela, il faut dans un premier temps se servir de la commande permettant de chiffrer un fichier : https://docs.ansible.com/ansible/latest/vault_guide/vault_encrypting_content.html
Il existe aussi *encrypt_string* qui est pratique pour définir une variable et la chiffrer mais elle devient très peu modulable avec Ansible Vault (pour l'éditer ou la visualiser) par exemple :

```
ansible-vault encrypt_string 'Mon petit secret' --name 'variable_secret'
New Vault password: *****
Confirm New Vault password: *****
variable_secret: !
  $ANSIBLE_VAULT;1.1;AES256
  [...]

ansible -i "127.0.0.1" all --ask-vault -m debug -a "var=variable_secret"
Vault password: *****
# Aucun problème à relever
127.0.0.1 | SUCESS => {
  "variable_secret": "Mon petit secret"
}

# On crée un fichier variables.yml et on C/C la variable chiffrée qui a été print dans le terminal
ansible-vault edit variables.yml
Vault password: *****
ERROR! input is not vault encrypted data for variables.yml
ansible-vault view variables.yml
Vault password: *****
ERROR! input is not vault encrypted data for variables.yml
# La variable a été chiffrée mais n'est pas "atteignable" dans le fichier dans lequel elle a été stockée
```

Il faut donc préférer chiffrer directement notre fichier de variables *vars.yml* grâce à *encrypt* !

Q5) Reprenez le playbook avec lequel vous avez créé un utilisateur et ajoutez-lui un mot de passe dans la tâche dédiée, puis faites en sorte que ce mot de passe soit chiffré dans le fichier vars.yml*, puis testez d'y accéder.*

Les rôles : clarté et partage :)

Dans la gestion de machines distantes comme nous l'avons fait jusqu'ici, nous pouvons nous retrouver, à terme, avec des playbooks très longs, peu lisibles et où les différentes tâches s'entre-mêlent sans avoir de liens particuliers entre elles. C'est là que les rôles interviennent !

Les rôles ne sont rien d'autre que des librairies que nous pouvons rencontrer dans n'importe quel langage de programmation. Chaque rôle a sa mission et n'influe pas sur les autres rôles. Les rôles ont cette structure :

```
roles/
├─ my_role/
│  ├─ defaults/
│  ├─ handlers/
│  ├─ tasks/
│  ├─ vars/
│  ├─ templates/
```

Chaque sous-dossier du rôle *my_role* contient un ficher *main.yml*, qui aura une tâche bien précise :
- *defaults* : contient les variables, jugées *low priority*, par défaut du rôle
- *handlers* : contient les handlers du rôle, les handlers sont des tâches qui ne sont exécutées que si une autre tâche a été exécutée avant et a renvoyé un résultat positif
- *tasks* : contient les tâches à effectuer, le playbook que nous avons constitué lors des précédents exercices
- *vars* : contient les variables du rôle, celles qui seront appelées dans les tâches
- *templates* : contient les templates du rôle, les templates sont des fichiers de configuration Jinja2 qui vont nous épargner bien des expressions regex.

La création d'une telle structure pour un rôle se fait facilement à la racine d'un projet avec la commande :

```
ansible-galaxy init <nom du rôle>
```

L'objectif désormai est donc de découper notre playbook principal en différents rôles. Par exemple un rôle dédié au déploiement des clés, et un autre rôle dédié à la mise en place de Suricata.
Pour Suricata, nous allons pouvoir remplacer la tâche de modification du fichier de configuration suricata.yml par un template.

Comme dit précédemment, un template est un fichier Jinja2 qui viendra modifier le fichier de configuration sur les machines distantes, et il sera également customisable grâce aux variables.
Pour créer un template, il suffit de se rendre dans le rôle qui nous intéresse puis dans le dossier *templates* et de créer un fichier *suricata.yml.j2*.

Dedans, on viendra copier le contenu du fichier de configuration de Suricata (https://github.com/OISF/suricata/blob/master/suricata.yaml.in) et on pourra remplacer les champs que nous souhaitons modifier par des variables, puis il suffira de définir ces variables dans le dossier *vars* du rôle.

On peut également séparer les tâches des handlers maintenant, grâce au fichier *main.yml* dédié aux handlers, dans le rôle en question. \

Q6) Créez les deux rôles distincts, avec la spécificité du template pour le rôle Suricata, modifiez le playbook principal, puis testez le tout !

Une fois tous ces rôles constitués, il ne reste plus qu'à nettoyer le playbook principal et appeler les rôles dont nous avons besoin. Il est alors possible d'appeler un rôle dans un playbook avec la syntaxe, Ansible se chargera de récupérer toutes les informations concernant ce rôle :

```
---
- hosts: <groupe>
  roles:
    - <nom du rôle>
```

On peut généralement trouver les rôles dont nous avons besoin sur GitHub lorsqu'il s'agit d'automatiser le déploiement d'un programme ou d'un service, il suffit alors de les importer dans le dossier *roles* de notre projet et de l'appeler dans le playbook principal.