
ANSIBLE

Ansible, industrialiser vos déploiements

Tables des matieres

1. Introduction

Ansible est un outil d'automatisation de gestion de configurations, l'installation de logicielles, la configuration via des modules d'infrastructure (Openstack, vmware, Cisco, Linux, Microsoft windows, ...)
Ansible permet le déploiement multi-nœuds d'infrastructure.
Ansible se démarque de la plupart des autres outils comme Puppet, Chef, .. Par la non utilisation d'agent (agentless). Pour ce faire, Ansible utilise le protocole OpenSSH pour déployer des configurations, installer des paquets, ...
La plate-forme Ansible est écrite en python et a été créée par Michael DeHaan. Il est inclus dans le cadre de la distribution Linux Fedora, propriété de Red Hat inc., et est également disponible pour Red Hat Enterprise Linux, CentOS et Scientific Linux via les paquets supplémentaires "Extra Packages for Enterprise Linux" (EPEL).

Ansible Inc. était la société derrière le développement commercial de l'application Ansible. Red Hat rachète Ansible Inc. en octobre 2015.

Le nom *Ansible* a été choisi en référence au terme Ansible choisi par Ursula Le Guin dans ses romans de science-fiction pour désigner un moyen de communication plus rapide que la lumière.

2. Présentation du LAB

Sur votre poste de travail vous trouverez le logiciel virtualBox contenant une VM, nommée Ansible.

Le login et mot de passe sont :
root
root

Dans le répertoire /root/ vous trouverez deux fichiers. Le fichier create-infra.sh et delete-infra.sh.
Ces derniers vont nous permettre de créer et de supprimer notre environnement de travail sous Docker.

```
cat  /root/create-infra.sh
```

```
#creation des containers pour Ansible

docker run --detach --name server00 --network ansible-net --ip
192.168.0.10 local/sshd
```

```
docker run --detach --name server01 --network ansible-net --ip 192.168.0.11 local/sshd
docker run --detach --name server02 --network ansible-net --ip 192.168.0.12 local/sshd


echo "liste des containers"
docker container ls
```

Pour créer le lab, exécutez la commande

```
sh create-infra.sh
```

Vous pouvez maintenant « pinger » ces machines et se connecter en ssh.
Le mot de passe est : pass

```
ping server00

ssh server00
```

3. Installation

L’installation du logiciel est vraiment très simple.
Via le gestionnaire de paquet de votre distribution ou via PIP. On préférera l’installation via le gestionnaire de paquet pour les updates.

```
yum install -y ansible
```

```
pip install ansible
```

4. Premier avec Ansible

Ansible n'utilise pas d'agent pour « discuter » avec les clients. Pour ce faire Ansible utilise le protocole openSSH.
Il va donc nous falloir générer des clés ssh et copier ces dernières sur les différents clients Ansible.

```
ssh-keygen #on ne créer pas de passphrasse, touche entrée
ssh-copy-id root@server0x
#x reprensente 0, 1, 2

ssh root@server
#..
```

On peut utiliser de deux manières :

- En ligne de commande
- Avec des playbooks

En ligne de commande

```
ansible -m ping -i inventaire
. . .
. . .
ERROR! Missing target hosts
```

Spécifier les hôtes

Lorsqu'on utilise Ansible en ligne de commande, il faut lui spécifier sur quelles machines on souhaite effectuer nos actions.

Créons un ficher pour renseigner les hôtes qui seront concernées par l'action

```
#vim inventaire

. . .
server00
server01
server02
```

Avec le fichier d'inventaire des hôtes

```
ansible -m ping -i inventaire server00
server00 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

Installons maintenant un paquet sur une cible

```
#ansible -i inventaire server00 -m yum -a "name=net-tools
state=present"

server00 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "msg": "",
    "rc": 0,
    "results": [
        "net-tools-2.0-0.24.20131004git.el7.x86_64 providing net-
tools is already installed"
    ]
}
```

Le paquet net-tools est déjà installé.
On peut donc mieux comprendre le comportement d'Ansible. Ansible fonctionne par état. Il vérifie l'état d'une demande. Dans l'exemple précedant, on demande de vérifier que le paquet net-tools est **présent**. Le paquet etant présent, il n'y a alors aucun changement sur l'hote cible.

Supprimons le paquet net-tools

```
#ansible -i inventaire server00 -m yum -a "name=net-tools
state=absent"
server00 | CHANGED => {
   "ansible_facts": {
      "discovered_interpreter_python": "/usr/bin/python"
   },
   "changed": true,
   "changes": {
```

```
      "removed": [
          "net-tools"
      ]
   },
   "msg": "",
   "rc": 0,
   "results": [
      "Loaded plugins: fastestmirror, ovl\nResolving Dependencies\n--> Running transaction check\n---> Package net-tools.x86_64 0:2.0-0.24.20131004git.el7 will be erased\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n================================================================================\n Package        Arch        Version                     Repository                 Size\n================================================================================\nRemoving:\n net-tools      x86_64      2.0-0.24.20131004git.el7    @base      918 k\n\nTransaction Summary\n================================================================================\nRemove  1 Package\n\nInstalled size: 918 k\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\n  Erasing    : net-tools-2.0-0.24.20131004git.el7.x86_64                 1/1 \n  Verifying  : net-tools-2.0-0.24.20131004git.el7.x86_64                 1/1 \n\nRemoved:\n  net-tools.x86_64 0:2.0-0.24.20131004git.el7                                  \n\nComplete!\n"
   ]
}
```

Installons maintenant un paquet sur toutes les cibles du fichiers inventaire

```
ansible -i inventaire all -m yum -a "name=net-tools state=present"

server02 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "msg": "",
    "rc": 0,
    "results": [
        "net-tools-2.0-0.24.20131004git.el7.x86_64 providing net-tools is already installed"
    ]
}
server01 . . .
```

```
server00 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "changes": {
        "installed": [
            "net-tools"
```

Le paquet est seulement sur la machine dans lequel le paquet est ABSENT.

5. L’inventaire

Ansible contient un fichier d’inventaire, communément appelé **/etc/ansible/hosts** et dans lequel on renseigne les cibles sur lesquels on effectue nos actions.

Nous avons déjà défini un fichier inventaire dans notre répertoire root.

Les paramètres d’inventaire

Ansible dispose de paramètres pour le fichier d’inventaire et pour vous donner un exemple, on peut prendre le port pour la connexion en SSH.

Lorsqu’Ansible se connecte à l’une des machines que vous avez spécifié dans l’inventaire, il tente de se connecter sur le port 22. Ce dernier est le port par défaut pour les connections en SSH. Bien évidemment, vous pouvez changer cela à l’aide du paramètre **ansible_ssh_port**.

Découvrons ensemble d’autres paramètres :

- **ansible_ssh_port :** le port à utiliser pour se connecter en SSH. Par défaut, à 22,
- **ansible_ssh_user :** l’utilisateur avec lequel on s’authentifie. Par défaut, il est placé à root,
- **ansible_ssh_pass :** le mot de passe à utiliser pour s’authentifier,
- **ansible_ssh_private_key_file :** la clé privée SSH à utiliser pour s’authentifier.

Les groupes

Lorsqu’on effectue nos actions, de manière général, on travaille sur plusieurs serveurs. On définit ces machines dans notre inventaire et en principe, on regroupe ces hôtes par groupe.

Prenons l’exemple suivant, nous souhaitons installer le serveur web apache sur certains de nos serveurs. On créera alors dans le fichier d’inventaire un groupe appelé apache, par exemple et on y rajoutera tous les serveurs où l’on veut l’installer :

```
[apache]
myserver.com
192.168.0.11
yourserver.com
```

Comme dans un fichier .ini, on spécifie un nom de groupe entre crochets : **[apache]**.

Une fois définit le groupe, On informe ansible sur quelle groupe de machines on effectue l’action désirée.

```
#ansible -i inventaire apache  -m yum -a "name=httpd
state=present"
server00 | CHANGED => {
   "ansible_facts": {
      "discovered_interpreter_python": "/usr/bin/python"
   },
   "changed": true,
   "changes": {
      "installed": [
         "httpd"
```

L'action d'installer le serveur web apache s'effectuera sur toutes les machines de la section [apache]

Les alias

Si l'on souhaite utiliser à notre , il va falloir que je saisisse à chaque fois son adresse IP et d'autres paramètres associées comme par exemple le **ansible_ssh_port**. Il est évident que je ne vais pas répéter cette opération ; nous pouvons alors utiliser les **alias :**

```
my_server ansible_ssh_host=192.168.0.10
[webservers]
my_host
```

Comme vous pouvez le voir, on définit un alias qui s'appelle **my_host**. A chaque fois que l'on souhaitera déployer un application sur ce serveur. On saisira son alias et non plus son adresse IP.

Les groupes de groupes

Prenons par exemple, trois applications

- Patrol : outil de supervision, permet de remonter des alertes sur votre système,
- Elasticsearch : bases de données qui contiendra les logs,
- Kibana : Une interface graphique pour Elasticsearch qui permettra de visualiser vos logs sous forme de graphiques.

Prenons le scénario suivant, on souhaite utiliser ces outils en interne :

- Nous aurons nos 3 premiers serveurs (host1 à host3) où l'on installera Kibana,
- Nous aurons 3 autres machines (host4 à host6) qui contiendront Elasticsearch,

- Enfin, on installera Patrol sur tous les hôtes pour surveiller l'état de nos machines et s'assurer que Kibana et Elasticsearch fonctionnent correctement.

Par conséquent, nous allons avoir un fichier d'inventaire qui ressemblera à cela :

```
host1 ansible_ssh_host=192.168.0.18
host2 ansible_ssh_host=192.168.0.19
host3 ansible_ssh_host=192.168.0.20
host4 ansible_ssh_host=192.168.0.28
host5 ansible_ssh_host=192.168.0.29
host6 ansible_ssh_host=192.168.0.30
[kibana]
host1
host2
host3
[elasticsearch]
host4
host5
host6
[patrol]
host1
host2
host3
host4
host5
host6
```

On peut voir beaucoup de lignes ! Beaucoup de doublons. Afin d'organiser au mieux notre inventaire. On peut alors utiliser des groupes de groupes en ajoutant la notation **:children** à son nom. Pour illustrer, reprenons le précédent fichier et créons le groupe de groupes patrol :

```
host1 ansible_ssh_host=192.168.0.18
host2 ansible_ssh_host=192.168.0.19
host3 ansible_ssh_host=192.168.0.20
host4 ansible_ssh_host=192.168.0.28
host5 ansible_ssh_host=192.168.0.29
host6 ansible_ssh_host=192.168.0.30
[kibana]
host1
host2
host3
[elasticsearch]
host4
host5
host6
[patrol:children]
kibana
elasticsearch
```

Avec l'ajout du suffixe **:children** au nom du groupe pour le passer en groupe de groupes puis nous avons utilisé les groupes précédemment créé, kibana et elasticsearch. Cela permet d'alléger le fichier puis de simplifier la compréhension.

Les variables d’hôte et de groupe

Nous avons vu précédemment les variables d’inventaire comme

ansible_ssh_host et **ansible_ssh_port** :

```
my_host ansible_ssh_host=192.168.0.30 ansible_ssh_port=22
```

De la même manière, il est possible de déterminer des variables qui seront associées à des hôtes. Une entreprise qui offre ses services informatiques via un site web et qui dispose de 5 serveurs répartis sur 5 continents différents :

```
192.168.0.18
192.168.0.19
192.168.0.20
192.168.0.21
192.168.0.22
```

Ce que souhaite faire l’administrateur système, c’est attribuer comme nom (hostname) à chaque serveur, le nom du continent sur lequel il se trouve. Ainsi, le serveur se trouvant en Europe s’appellera europe, celui se trouvant en Asie s’appellera asia et ainsi de suite. Il va alors déterminer une variable d’hôte nommée hostname et lui attribuera le nom du continent :

```
192.168.0.18 hostname=europe
192.168.0.19 hostname=asia
192.168.0.20 hostname=africa
192.168.0.21 hostname=australia
192.168.0.22 hostname=america
```

On peut alors utiliser ces variables avec Ansible et par exemple attribuer leur valeur comme hostname de la machine concernée pour que ça soit plus parlant pour les utilisateurs travaillant dessus.

De la même manière que les hôtes, il est possible d’avoir des variables de groupe : on attribue des variables à des groupes en utilisant le suffixe **:vars**. Reprenons notre scénario de la partie précédente (groupes de groupes) où nous avions Kibana, Patrol et Elasticsearch.

L’authentification à ces outils nécessite un nom d’utilisateur ainsi qu’un mot de passe. De plus, vous voulez que le nom d’utilisateur et le mot de passe diffèrent selon l’outil. Vous devez alors prendre cela en compte et configurer la phase d’authentification. On va alors utiliser des variables de groupe. Dans notre exemple, nous aurons 2 variables pour chaque groupe : username (le nom d’utilisateur) et password (le mot de passe).

```
host1 ansible_ssh_host=192.168.0.18
host2 ansible_ssh_host=192.168.0.19
host3 ansible_ssh_host=192.168.0.20
```

host4 ansible_ssh_host=192.168.0.28
host5 ansible_ssh_host=192.168.0.29
host6 ansible_ssh_host=192.168.0.30
[kibana]
host1
host2
host3
[elasticsearch]
host4
host5
host6
[patrol:children]
kibana
elasticsearch
 # Nos variables de groupe
[kibana:vars]
username=kibana
password=k1i2b3a4n5a6
[elasticsearch:vars]
username=elastic
password=e1l2a3s4t5i6c7
[patrol:vars]
username=patrol
password=p1a2t3r4o5l6

N’oubliez pas, pour indiquer que nous allons définir des variables de groupe, on ajoute le suffixe **:vars** juste après le nom du groupe.

6. La notion de module

Jusqu'à présent nous avons utiliser Ansible en mode commande ou mode adhoc

```
#ansible -i inventaire server00 -m ping
#ansible --invetory inventaire server00 --module ping
```

L'option 'm' de la commande permet d'invoquer un module.
Mais qu'est qu'un module ?
Un module est un 'micro' programme qui permet d'effectuer un tache sur la ou les cibles.
Le module **ping** par exemple est un module qui nous permet de valider l'établissement d'une connexion entre le serveur de déploiement Ansible et les cibles.

Il existe énormément de modules :

https://docs.ansible.com/ansible/2.4/list_of_all_modules.html

Nous trouvons des modules pour des opérations basique. Comme l'Installation des paquets, la copie des fichiers, la modification du contenu des fichiers via des expressions régulières, l'exécution des commandes Shell sur les différentes cibles, monter des systèmes de fichiers, la manipulation de services...

Également des modules pour des cibles Microsoft Windows, des routeurs cisco, juniper,.. également des modules pour gérer notre cloud Amazon, Google ou Openstack. Des modules pour Vmware, pour Docker, des modules pour créer, manipuler des bases de données Mysql, Postgresql, ...

Comme vous pouvez le voir il existe de très nombreux modules.
Puis dans le cas ou vous ne trouvez pas de module pour notre besoin. Nous pouvons toujours pouvoir écrire ce dernier dans des langages aussi divers que le shell, Python, ruby et même en C !

Examinons un module simple.

Le module Copy, qui comme son l'indique permet d'effectuer des copie du serveur Ansible vers les machines cibles.

La documentation Ansible, indispensable ! Nous informe sur l'utilisation du module.
On trouvera un synopsys, des options, des exemples et des notes sur le status, …

Le synopsys du module Copy :

- The copy module copies a file from the local or remote machine to a location on the remote machine. Use the fetch module to copy files from remote locations to the local box. If you need variable interpolation in copied files, use the template module.
- For Windows targets, use the win_copy module instead.

Différentes options du module :

parameter	required	default	choices	comments
content	no			When used instead of *src*, sets the contents of a file directly to the specified value. For anything advanced or with formatting also look at the template module.
decrypt (added in 2.4)	no	Yes	• yes • no	This option controls the autodecryption of source files using vault.
dest	yes			Remote absolute path where the file should be copied to. If *src* is a directory, this must be a directory too. If *dest* is a nonexistent path and if either *dest* ends with "/" or *src* is a directory, *dest* is created. If *src* and *dest* are files, the parent directory of *dest* isn't created: the task fails if it doesn't already exist.

Un exemple d'utilisation du module Copy :

```
- copy:
    src: /srv/myfiles/foo.conf
    dest: /etc/foo.conf
    owner: foo
    group: foo
    mode: u+rw,g-wx,o-rwx
```

On trouve dans la documentation Ansible, tous les modules et leur utilisations

7. Les playbooks

Jusqu'à présent nous avons utiliser Ansible en mode commande ou mode adhoc. Nous ne pouvons pas gérer efficacement une infrastructure avec le simple mode commande.
Nous avons besoin pour cela de recette a appliquer sur les cibles. Pour ce faire, Ansible nous fournit des playbooks

7.1 Introduction

Un playbook est un méthode qui va nous permettre d'enchaîner plusieurs actions les une a la suite des autres.

On spécifie dans un simple fichier yaml (Yet Another Markup Language), les actions a effectuer sur les différentes cibles.

La commande ansible-playbook exécutera le playbook

```
#ansible-playbook installation.yml
```

7.2 Mon premier playbook

Nous allons prendre un exemple pour comprendre le fonctionnement des playbooks

Playbook pour installer un paquet bash-completion sur une distributin Centos

```
#cat install.yml

 #les hôtes qui sont concernées par le playbook
 - hosts : webserver
 #les taches que l'on va effectuer sur les cibles webserver
   tasks :
 #Le nom de la tache a effectuer dans notre une installation
   - name : Installation du paquet bash-completion
 # le module invoquer pour installer YUM
     yum :
      name : bash-completion
 #l'état dans lequel doit se trouver le paquet sur les cibles
      state : present
```

Dans ce playbooks, on demande que le paquet **bash-completion** soit **present** sur la ou les machines cibles et ce avec le module **yum**.

On peut voir qu'Ansible fonctionne par état. Le paquet est'il présent. Non, alors Ansible demande l'installation du paquet.

Un autre exemple:

```
#cat copie.yml

 - hosts: serverweb
   tasks:
   -name: Copie du fichiers de conf de nginx
    copy:
      src: /tmp/nginx.conf
     dest: /etc/nginx/
    owner: nginx
```

```
#cat install.yml

- hosts : webserver
  tasks :
  - name : Installation du paquet bash-completion
    yum :
     name : bash-completion
     state : present
  - name : Installation du paquet vim
    yum :
     name : vim
     state : present
  - name : Installation du paquet nano
    yum :
     name : nano
     state : present
  - name :  . . .
```

Si l'on regarde ce playbook et que l'on doit installer plusieurs paquet, peut etre peut on trouver une solution plus élégante pour écrire notre playbook.

7.3 Les boucles

Nous avons souvent le besoin de répéter des taches. Comme dans l'exemple ci dessus. Ansible nous offre des boucles qui vont nous permettent de réaliser des itérations sur des actions.

With_items

Prenons le fichier install.yml pour comprendre la boucle with_items

```
#cat install.yml

- hosts: webserver
  tasks:
  - name: Installation du paquet bash-completion
    yum:
     name: «{{ item }}»
     state: present
  with_items:
     - bash-completion
     - nano
     - vim
```

La boucle With_items est bien plus simple a utiliser que la version précédente.

IL existe maintenant la boucle loop.

```
#cat install.yml

- hosts: webserver
  tasks:
  - name: Installation du paquet bash-completion
    yum:
     name: «{{ item }}»
     state: present
  loop:
     - bash-completion
     - nano
     - vim
```

Des sous items

```
#cat install.yml

- hosts: webserver
  tasks:
  - name: gestion des packages
    package: name={{ item.name }} state={{ item.state }}
    with_items:
     - { name: tree, state: present }
     - { name: nano, state: present }
OU
    loop:
      - { name: tree, state: present }
     - { name: nano, state: present }
```

7.4 Les variables

La façon la plus simple de définir des variables est de les insérer directement dans votre playbook à l'intérieur d'une section **vars**

```
 vars:
  nginx_ssl : /etc/nginx/ssl
  nginx_conf_file : /etc/nginx/nginx.conf
```

Nous avons définis deux variables, **nginx_ssl** et **nginx_conf_file** avec leurs valeurs respectives.A partir un exemple très simple où l'on définit une variable, lui attribue une valeur puis l'affiche au sein de notre playbook. Voici le contenu de notre playbook :

```
 vars:
  nginx_ssl: /etc/nginx/ssl
  nginx_conf_file: /etc/nginx/nginx.conf

 - name: creation du directory ssl et copie du fichier de conf
ansible
   file: state=directory path= {{ nginx_ssl }}
 - name: copie du ficher nginx.conf
   copy: src=nginx.conf dest= {{  nginx_conf_file }}
```

Allons un peu plus loin avec les variables.

On peut également placer les variables dans un fichier externe.

```
 - hosts: server00

  vars_files:
   - vars.yml

  tasks:
  - name: installation des dépots epel
    yum:
      name: "{{ pkg }}"
      state: absent
  - name: creation de user
    user:
      name: "{{ user }}"
      state: present
```

Le contenu du fichiers vars.yml

```
pkg: epel-release
user: root
password: pass
```

Toujours dans l'utilisation des variables

```
---
- hosts: server00
  vars:
   dir:
   - celine
   - ludo
   - bill
  tasks:
  - name: création de repertoire avec une variable DIR
    file: path=/home/{{ item }} state=directory
    with_items: "{{ dir }}"
```

On créer des repertoires **/home/celine, /home/ludo**, ... avec le contenu de la variable **dir**.

Avec l’instruction loop au lieu de with_items

```
---
- hosts: server00
  vars:
   dir:
   - celine
   - ludo
   - bill
  tasks:
  - name: création de repertoire avec une variable DIR
    file: path=/home/{{ item }} state=directory
    loop: "{{ dir }}"
```

Il y a des variables dont nous n’avons pas parlé. Facts : recueillir des « faits » En anglais, on utilise le terme « gather facts », on dit qu’Ansible recueille des faits.

Il se connecte à la machine cible puis récupère un certain nombre d’informations : le système d’exploitation, les adresses IP, la mémoire, le disque, etc. Il stocke ensuite ces informations dans des variables qu’on nomme **facts**, qu’on pourrait traduire en français par faits.

Pour vous montrer un exemple, on pourrait afficher le nom du système d’exploitation installé sur votre machine cible. Le playbook de déploiement deploy.yml ressemblera à :

```
- name: show os name
  hosts: server00
  gather_facts: true

  tasks:
    - name: Affiche le système d’exploitation
      debug:
        msg: "Operating system name is :{{ ansible_distribution }}"
```

Recueillir les facts en ligne de commande, nous avons pu voircomment recueillir les facts dans un playbook en demandans d’afficher un message avec la variable de la variable *ansible_distribution .*

Lors de la connexion a la machine cible, stocke ces facts dans des variables. Pour afficher tous les facts on utlise le module **setup** qui s’utilise de la manière suivante :

```
#ansible -m setup server00
```

7.5 Les conditions

7.6 Les includes

8. La puissance des rôles

Les rôles représentent une manière d'abstraire les directives *includes*. C'est en quelque sorte une couche d'abstraction. Grâce aux rôles, il n'est plus utile de préciser les divers *includes* dans le playbook, ni les paths des fichiers de variables etc. Le playbook n'a qu'à lister les différents rôles à appliquer et le tour est joué !

En outre, depuis les *tasks* du rôle, l'ensemble des chemins sont relatifs. Inutile donc de préciser l'intégralité du path lors d'un copy, template ou d'une tâche. Le nom du fichier suffit, Ansible s'occupe du reste.

On peut faire beaucoup avec les playbooks et les *includes*, cependant, lorsque nous commençons à gérer des infrastructures complexes, on a beaucoup de tâches et les rôles s'avèrent salvateurs dans l'organisation et l'abstraction qu'ils apportent.

Par ailleurs, Ansible met à disposition une plate-forme permettant de télécharger et de partager des rôles utilisateurs : Ansible galaxy. Un bon moyen de ne pas réinventer la roue. Doit on faire confiance a des plateformes externes ?

8.1 Création d'un rôle simple

En CLI, on utilise la commande afin de préparer l'arborescence d'un modele de rôle. Nos playbooks seront à la racine de notre dossier tandis que les rôles seront dans rôles.

Deploy.yml
roles/

- user
- nginx
- mysql

Ainsi, on invoque le rôle depuis un playbook, sans avoir besoin de préciser autre chose que son nom, Ansible connaît le chemin.

on initialise un role vide depuis le dossier rôles

```
ansible-galaxy init common

cd common && ls -R
README.md    handlers    tasks        vars
defaults    meta        tests

./defaults:
main.yml

./handlers:
main.yml

./meta:
main.yml

./tasks:
main.yml

./tests:
inventory    test.yml

./vars:
main.yml
```

On voit ici qu'Ansible nous a créé plusieurs dossiers avec un fichier main.yml dans chacun d'eux. Par défaut, Ansible ira chercher les informations dans les main.yml de chaque répertoire. Cependant, vous pouvez créer d'autres fichiers et les référencer dans vos instructions. Par exemple, il est tout à fait possible de créer une autre tâche dans tasks et de l'inclure depuis tasks/main.yml. Passons rapidement en revue la fonction de chaque répertoire.

La commande ansible-galaxy est normalement destinée à être utilisée avec Galaxy, nous en tirons cependant profit pour facilement obtenir un boilerplate de rôle vide. Par défaut, elle ne créée pas les répertoires files et templates dont nous allons tout de même parler.

defaults
Ce sont ici les variables par défaut qui seront à disposition du rôle.

vars
De la même manière que defaults, il s'agit ici de variables qui seront à disposition du rôle, cependant, celles-ci ont en général vocation à être modifiées par l'utilisateur et elles prennent le dessus sur celles de defaults si elles sont renseignées.

tasks
Sans grande surprise, c'est ici que vous référencerez vos tâches.
files

Tous les fichiers étant destinés à être traités par le module copy seront placés ici.

templates
Idem que copy, mais cela concerne les fichiers du module template.

meta
Il y a ici plusieurs usages, notamment dans le cas de rôles publiés sur Galaxy. Dans notre cas, on référencera ici les dépendances à d'autres rôles.

tests
Pas utile pour nous, c'est seulement pour Galaxy et la doc n'est pas très loquace à ce propos…

En plus des répertoires sus-mentionnés, il y a le README qu'il est de bon ton de renseigner afin d'expliquer comment utiliser le rôle, quelles sont les variables à définir etc.

Aucun des répertoires n'est impératif – quoique sans tasks, notre rôle ne sert pas à grand chose. On effacera donc les répertoires dont on n'a pas l'utilité.