

Introduction au cours complet sur Ansible

Introduction au cours complet pour débutants afin d'apprendre pas à pas les différents concepts de l'outil d'automatisation Ansible.

Introduction

Bonjour les copains, prêt pour découvrir une nouvelle technologie ? Allons-y, let's go, c'est parti les amis !



C'est quoi déjà Ansible ?

Ansible est un **moteur d'automatisation** informatique open source, qui peut éliminer les corvées de votre vie professionnelle, et améliorer également considérablement l'évolutivité, la cohérence et la fiabilité de votre environnement informatique.

Vous pouvez utiliser Ansible pour automatiser trois types de tâches :

- **Provisioning** : configurer les différents serveurs dont vous avez besoin dans votre infrastructure (On a déjà eu l'occasion par exemple de l'utiliser avec Vagrant pour créer un cluster Kubernetes, article).
- **Configuration** : modifier la configuration d'une application, d'un système d'exploitation ou d'un périphérique, démarrer et arrêter les services, installer ou mettre à jour des

applications, mettre en œuvre une politique de sécurité, ou effectuer une grande variété d'autres tâches de configuration.

- **Déploiement d'applications** : adopter une démarche DevOps en automatisant le déploiement d'applications développées en interne sur vos environnements de production.

Bref, un vrai couteau suisse ☐ !

Objectif du cours

Dans ce cours nous allons **apprendre à automatiser les tâches d'administration système répétitives** sur un parc informatique On premise (local) ou Cloud à l'aide d'Ansible.

Histoire d'Ansible

Lorsque nous examinons l'histoire d'Ansible, il est facile de comprendre la motivation qui a poussé les inventeurs de l'outil, à créer ce système d'automatisation, atteignant aujourd'hui un seuil de maturité assez exceptionnel.

L'outil Ansible a été développé par **Michael DeHaan** (il n'est pas le seul bien sûr, il faut tout une armée d'informaticiens pour construire à tel outil, mais l'idée de base vient de lui), c'est aussi l'auteur de l'application de serveur de provisionnement **Cobbler** (l'un de ses premiers projets qu'il a commencé à Redhat). **Red Hat** était très intéressé par un outil de gestion de configurations, plus particulièrement après l'arrivée de Puppet et Chef dans le marché des outils d'automatisation.

Pendant ce temps, le mot DevOps a commencé à être beaucoup utilisé, et diverses discussions ont lieu sur les chaînes d'outils devops. Les discussions sont entièrement axées sur la culture mais surtout sur le fait de rendre les outils (souvent Open Source) plus faciles à utiliser, plutôt que d'acheter un outil universel à un grand fournisseur.

Auparavant Michael DeHaan travaillait pour Puppet en tant que chef de produit pendant un certain temps, ou il y a eu quelques divergences.. Pour lui la simplicité de l'outillage, aurait dû être plus prioritaire et la chose la plus importante dans un outillage d'automatisation.

Après avoir quitté Puppet et travaillé dans de nouveaux projets, il n'a pas réussi à convaincre les gens et ses nouveaux collègues à utiliser Puppet en tant qu'outillage d'automatisation principal. Il trouvait aussi que les utilisateurs n'étaient pas non plus en mesure d'adopter des

pratiques d'automatisation de style DevOps en raison de la complexité des outils alternatifs à Puppet. Les scripts internes étaient considérés comme plus faciles, mais ce n'était pas plus fiable.

L'idée d'un outil d'automatisation simple a commencé à émerger. Ansible n'a pas vraiment commencé non plus, même si les idées se formaient rapidement. Finalement, c'était inévitable, En février 2012 Ansible a donc commencé en tant que projet. Il a décollé assez rapidement grâce à de nombreux administrateurs systèmes et développeurs travaillant d'arrache-pied sur le projet. Le but principal consistait à montrer au monde qu'il y avait un moyen plus **simple d'automatisation**.

Pari réussi, puisque l'entreprise Red Hat rachète Ansible en octobre 2015, Ansible tourne actuellement à plein régime, et le projet est pris en charge par une entreprise de bonne taille (Red Hat). Ansible est actuellement l'outil de gestion de configuration le plus étoilé et le plus forké sur [GitHub](#).

Public visé

Ce tutoriel est conçu pour les débutants pour les **aider à comprendre les bases Ansible à partir de zéro**. Ce tutoriel vous donnera une compréhension suffisante de la technologie, qui vous permettra plus tard d'atteindre des niveaux d'expertise beaucoup plus élevés.

Prérequis

Avant de poursuivre ce cours, vous devez au minimum avoir une compréhension de base sur les commandes Linux, cela vous aidera mieux dans les tâches Ansible. et d'avancer rapidement sur la piste d'apprentissage, mais ce n'est pas non plus indispensable.

Êtes-vous prêt pour automatiser vos tâches ingrates ? C'est parti !

Avantages et fonctionnement d'Ansible

Dans ce chapitre, nous verrons les problèmes rencontrés avant l'utilisation Ansible, son utilisation et son fonctionnement

Pourquoi avons-nous besoin d'Ansible ?

Bien avant de vous expliquer ce qu'est Ansible, il est de la plus haute importance de comprendre les **problèmes rencontrés avant l'utilisation Ansible**. Prenons un petit retour en arrière sur le début de l'informatique lorsque le déploiement et la gestion des serveurs de manière fiable et efficace ont été un défi. Auparavant, les administrateurs système géraient les serveurs manuellement, installaient les logiciels, changeaient les configurations et administraient les services sur des serveurs individuels.

À mesure que les Datacenters se développaient et que les applications hébergées devenaient plus complexes, les administrateurs ont réalisé qu'ils ne pouvaient pas faire évoluer leur gestion manuelle des systèmes aussi rapidement que les applications qu'ils activaient. Cela a également entravé la vitesse du travail des développeurs car l'équipe de développement était agile et publiait fréquemment des logiciels, mais les opérations informatiques passaient plus de temps à configurer les systèmes. C'est pourquoi que des outils de provisionnement de serveur et de gestion de la configuration automatique ont prospéré. Une solution des solutions au problème mentionné ci-dessus est Ansible.

L'outil Ansible vise à fournir des gains de productivité importants à une grande variété de défis d'automatisation. Comme vu sur le chapitre précédent, c'est un outil qui se veut simple à l'utilisation mais suffisamment puissant pour automatiser des environnements d'applications complexes à plusieurs niveaux.

Avantages d'Ansible

Voici une liste des avantages d'Ansible :

- **Gratuit:** Ansible est un outil open source.
- **Simple :** Ansible utilise une syntaxe simple écrite en YAML. Aucune compétence en programmation particulière n'est nécessaire pour créer les playbooks d'Ansible. Il est également simple à installer.
- **Puissant:** Ansible vous permet de modéliser des workflows très complexes.

- **Flexible:** Ansible vous fournit des centaines de modules prêts à l'emploi pour gérer vos tâches, quel que soit l'endroit où ils sont déployés. Vous pouvez réutiliser le même playbook sur un parc de machines Red Hat, Ubuntu ou autres.
- **Agentless :** vous n'avez pas besoin d'installer d'autres logiciels ou d'ouvrir des ports de pare-feu supplémentaires sur les systèmes clients que vous souhaitez automatiser. Ansible réduit encore l'effort requis pour que votre équipe commence à automatiser immédiatement.
- **Efficace:** Parce que vous n'avez pas besoin d'installer de logiciel supplémentaire, il y a plus de place pour les ressources d'application sur votre serveur.

Que peut faire Ansible?

Ansible peut être utilisé de différentes manières. J'en ai mentionné quelques-unes ci-dessous:

Déploiement d'applications

Ansible vous permet de déployer rapidement et facilement des applications à plusieurs niveaux. Vous n'aurez pas besoin d'écrire du code personnalisé pour automatiser vos systèmes; vous listez les tâches à effectuer en écrivant un playbook, et Ansible trouvera comment amener vos systèmes à l'état dans lequel vous voulez qu'ils soient. En d'autres termes, vous n'aurez pas à configurer manuellement les applications sur chaque machine . Lorsque vous exécutez un playbook à partir de votre machine de contrôle, Ansible utilisera le protocole SSH pour communiquer avec les hôtes distants et exécuter toutes les tâches (Tasks).

Orchestration

Comme son nom l'indique, l'orchestration consiste à amener différentes éléments à interagir ensemble sans incohérence. Par exemple, avec le déploiement d'applications, vous devez gérer non seulement les services frontend, mais également les services backend comme les bases de données, le réseau, le stockage, etc... Vous devez également vous assurer que toutes les tâches sont gérées dans le bon ordre. Grâce à Ansible vous orchestrez les éléments de votre infrastructure à l'aide des playbooks Ansible, et vous pouvez les réutiliser sur différents types de machines, grâce à la portabilité des modules Ansible.

Conformité et sécurité

Comme pour le déploiement d'applications, des politiques de sécurité de votre entreprise (telles que des règles de pare-feu ou le verrouillage des utilisateurs) peuvent être mises en œuvre avec d'autres processus automatisés. Si vous configurez les détails de sécurité sur la machine de contrôle et exécutez le playbook associé, tous les hôtes distants seront automatiquement mis à jour avec ces détails. Cela signifie que vous n'aurez pas besoin de surveiller chaque machine pour vérifier la conformité de la sécurité en continu manuellement. De plus, tous les identifiants (identifiants et mots de passe des utilisateurs admin) qui sont stockés dans vos playbooks ne sont récupérables en brut par aucun utilisateur.

Provisionnement du cloud

La première étape de l'automatisation du cycle de vie de vos applications consiste à automatiser l'approvisionnement de votre infrastructure. Avec Ansible, vous pouvez provisionner des plateformes cloud, des hôtes virtualisés, des périphériques réseau et des serveurs physiques.

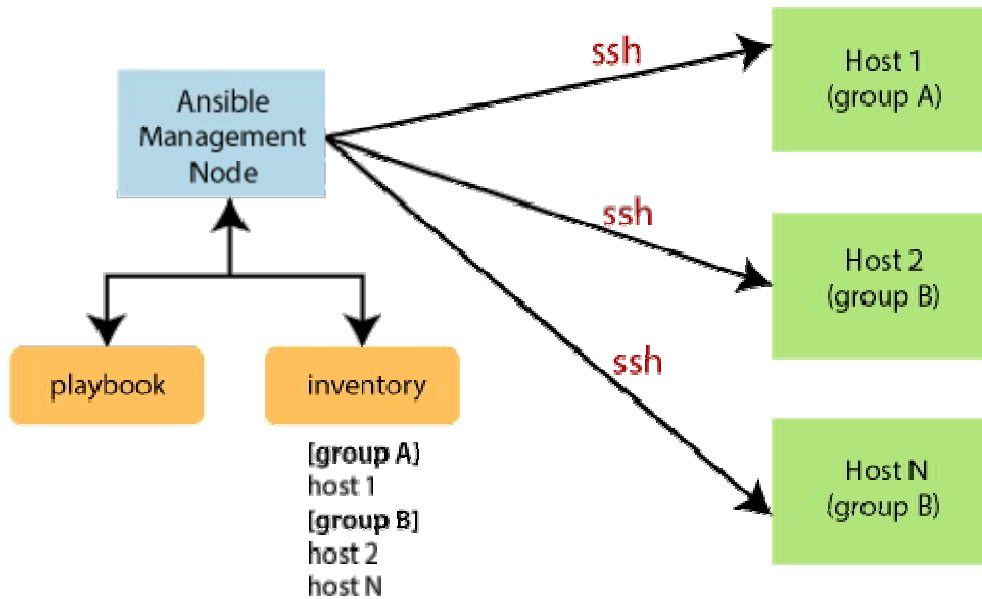
Comment fonctionne Ansible

Dans Ansible, il existe deux catégories d'ordinateurs: le **nœud maître** (master) et les **nœuds esclaves** (slaves). Le nœud maître est une machine sur laquelle est installé l'outil Ansible. Il doit y avoir au moins un nœud maître, bien qu'un nœud maître de sauvegarde puisse également exister.

Ansible fonctionne en se connectant à vos nœuds en **SSH** et en y poussant de petits programmes, appelés **modules**. Ces modules sont définis dans un fichier nommé le **Playbook**. Le nœud maître, se base sur un fichier d'inventaire qui fournit la liste des hôtes sur lesquels les modules Ansible doivent être exécutés.

Ansible exécute ces modules en SSH et les supprime une fois terminé. La seule condition requise pour cette interaction est que votre nœud maître Ansible dispose d'un accès de connexion aux nœuds esclaves. Les clés SSH sont le moyen le plus courant de fournir un accès, mais d'autres formes d'authentification sont également prises en charge.

Voici un schéma qui reprend notre explication :



Dans la prochaine partie, nous découvrirons comment configurer notre environnement d'Ansible.

Configurer votre environnement Ansible

Dans ce chapitre nous verrons comment configurer notre environnement Ansible et nous utiliserons l'outil Vagrant pour provisionner nos nœuds distants.

Introduction

Dans ce chapitre, nous découvrirons **comment configurer notre environnement Ansible**. Pour ce faire il nous faut principalement deux types de machines :

- **Machine de contrôle** (ou machine maître) : Machine à partir de laquelle nous pouvons gérer d'autres machines. et pour gérer ces nœuds distants, nous devons installer Ansible sur la machine maître.
- **Machine esclave** : Machines manipulées / contrôlées par une machine de contrôle.

Machine de contrôle

Prérequis et conseils

Actuellement, Ansible peut être exécuté à partir de n'importe quelle machine sur laquelle Python 2 (version 2.7) ou Python 3 (version 3.5 et supérieures) est installé (l'interpréteur python est installé par défaut sur les machines Linux). Cela inclut Red Hat, Debian, CentOS, macOS, n'importe lequel des BSD, etc. Cependant, **Windows n'est pas pris en charge pour le nœud de contrôle.**

Conseil

Lors du choix de votre nœud de contrôle, gardez à l'esprit qu'il vaut mieux que le nœud de contrôle exécute vos playbooks à proximité des machines gérées, car les modules sont lancés à partir du protocole de communication SSH. Par exemple, si vous exécutez Ansible dans un cloud, envisagez de l'exécuter à partir d'une machine à l'intérieur de ce cloud.

Installation d'Ansible

Il existe **différentes façons d'installer le moteur Ansible**. Vous avez le choix entre les méthodes suivantes :

- Installation de la dernière version proposée par votre **gestionnaire de package** de votre OS.
- Installation avec le **gestionnaire de packages python pip**.
- Installation depuis les **sources Ansible** afin d'utiliser et tester les dernières fonctionnalités.

Installer Ansible sur Ubuntu

Commencez d'abord par mettre à jour la liste de vos dépôts APT :

```
sudo apt update
```

Installez ensuite le logiciel software-properties-common qui vous permettra de gérer plus facilement les dépôts indépendants (ppa) de votre distribution.

```
sudo apt install software-properties-common
```

Ajoutez le repository ansible en version stable :

```
sudo apt-add-repository --yes --update ppa:ansible/ansible
```

Enfin, installez le moteur ansible :


```
sudo apt install ansibleCopier
```

Installation d'Ansible sur RHEL, CentOS ou Fedora

L'installation se tient sur une ligne de code ☐ :

```
sudo yum install ansibleCopier
```

Installer depuis pip

Comme dit tout à l'heure, Ansible peut être installé avec le gestionnaire de paquets Python pip. Cette méthode peut être intéressante, quand la dernière version stable n'est pas proposée par le gestionnaire de package de votre OS.

Si pip n'est pas déjà disponible sur votre système Python, exécutez les commandes suivantes pour l'installer:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py  
python get-pip.py --userCopier
```

Installez ensuite ansible :

```
pip install --user ansibleCopier
```

Les nœuds esclaves

Sur les nœuds esclaves, vous avez besoin d'un moyen de communication SSH, le port SSH doit donc être ouvert. Vous avez également besoin de Python 2 (version 2.6 ou ultérieure) ou Python 3 (version 3.5 ou ultérieure).

Je ne cesserai de vous le dire chaque jour : "Un bon devopsien, est un informaticien qui automatise ses tâches !". Et comme nous sommes malins ☐, nous allons automatiser le provisionnement de nos nœuds esclaves depuis l'outil Vagrant dans l'hyperviseur Virtualbox.

Rendez-vous sur la [page d'installation officielle de vagrant](#) et téléchargez le package correspondant à votre système d'exploitation et à votre architecture.

Par exemple sur Fedora 30, il faut choisir le package 64 bits sous Centos. L'installation se passera ainsi comme ça :

```
sudo rpm -Uvh
https://releases.hashicorp.com/vagrant/2.2.6/vagrant_2.2.6_x86_64.rpm
mCopier
```

Pour ubuntu, il faut choisir le package 64 bits sous Debian, soit :

```
curl -O
https://releases.hashicorp.com/vagrant/2.2.6/vagrant_2.2.6_x86_64.deb
b

sudo apt install ./vagrant_2.2.6_x86_64.deb
Copier
```

Testez ensuite le bon déroulement de votre installation, en vérifiant la version de vagrant :

```
vagrant --version
Copier
```

Résultat :

```
Vagrant 2.2.6
```

Vagrantfile

C'est à partir du fichier Vagrantfile que toute la procédure débute. Ce fichier décrit comment nos nouvelles machines esclaves seront provisionnées. J'ai mis un maximum de commentaires, histoire de comprendre la finalité de chaque instruction. Voici déjà à quoi ressemble donc notre fichier Vagrantfile :

```
#
#####

# ##### CONFIGURATION VARIABLES
#####

#
#####

IMAGE_NAME = "bento/ubuntu-18.04" # Image to use

MEM = 2048 # Amount of RAM
```

```

CPU = 1                                # Number of processors

SLAVE_NBR = 2                          # Number of slaves node

NETWORK_ADAPTER="wlp1s0"              # Bridge network adapter


Vagrant.configure("2") do |config|

  # RAM and CPU config

  config.vm.provider "virtualbox" do |v|

    v.memory = MEM

    v.cpus = CPU

  end

  # Slave node config

  (1..SLAVE_NBR).each do |i|

    config.ssh.insert_key = false

    config.vm.define "slave-#{i}" do |slave|

      # OS and Hostname

      slave.vm.box = IMAGE_NAME

      slave.vm.hostname = "slave-#{i}"

      slave.vm.network "public_network", bridge:
NETWORK_ADAPTER

    end
  end
end

```

```
end  
endCopier
```

Vous pouvez personnaliser vos machines esclaves depuis le fichier Vagrantfile à partir des variables de configuration. Par exemple vous pouvez agrandir le nombre de nœuds en changeant la variable **SLAVE_NBR**, voire attribuer davantage de ressources à vos nœuds en revalorisant les variables **CPU** et/ou **RAM**.

Pour ce cours, j'ai choisi le mode réseau accès par pont (bridge) afin de faire communiquer mes machines virtuelles totalement entre elles vers l'extérieur via la machine hôte s'approchant. Dans mon cas j'utilise la carte réseau de mon ordinateur **wlp1s0** (lancez la commande `ip a` afin de connaître la votre).

Information

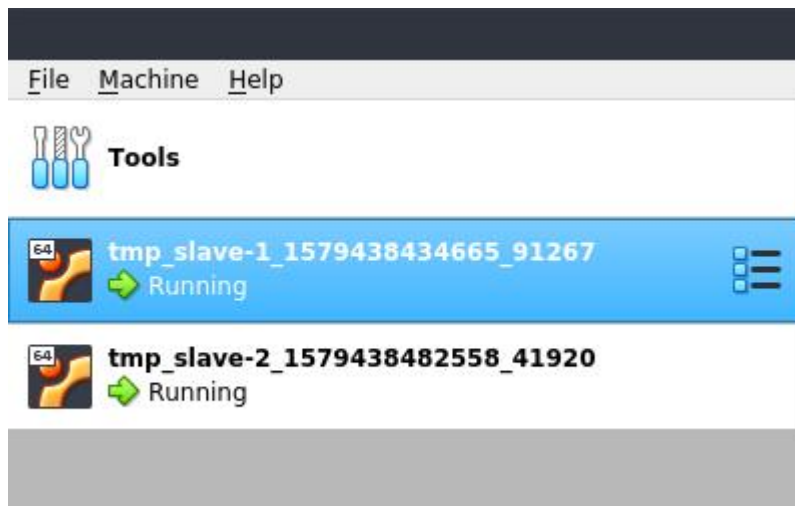
Vous pouvez aussi définir des IPs privées en ajoutant l'option **ip: "MON IP"** après le paramètre **bridge**. Par exemple si vous souhaitez avoir l'ip 192.169.0.21 sur votre nœud distant 1 et l'IP 192.168.0.22 sur votre nœud distant 2, alors vous écririez alors la ligne suivante : **slave.vm.network "public_network", bridge: NETWORK_ADAPTER, ip: "192.168.0.2#{i}"**

Enfin nous travaillerons sur des machines virtuelles Ubuntu sous sa version 18.04.

Pour provisionner vos deux nouvelles VMs. Ouvrez un terminal et placez vous d'abord au même niveau que le fichier Vagrantfile et lancez ensuite la commande suivante :

```
vagrant upCopier
```

Après la fin de l'exécution, si on retourne sur Virtualbox, on peut visualiser nos deux nouvelles machines :



Conclusion

Après avoir exécuté les instructions ci-dessus, vous êtes prêt à gérer les machines distantes via Ansible. Dans le prochain chapitre suivant nous verrons comment exécuter nos premiers modules sur nos machines esclaves.

Exécutez vos premières commandes Ansible

Dans ce chapitre nous apprendrons à utiliser la cli Ansible et configurer notre inventaire Ansible.

Introduction

Maintenant que nous avons étudié comment installer et configurer notre environnement Ansible, et que nous avons vu le fonctionnement global d'Ansible. Il est temps maintenant de **lancer nos premières commandes Ansible**.

Comme vu sur les chapitres précédents Ansible peut servir pour différentes tâches, et nous allons alors tout au long de ce cours **comprendre les cas d'utilisation les plus courants** avant d'explorer toutes les puissantes fonctionnalités de configuration et d'orchestration qu'offre Ansible.

Concernant mon environnement de travail pour ce cours, j'aurai comme IP statique **192.169.0.21** pour mon nœud distant numéro 1 et l'IP **192.168.0.22** pour mon nœud distant numéro 2. J'en ai aussi profité pour rajouter en outre un alias dans le fichier `hosts` afin de mieux distinguer mes différentes machines, voici à quoi ressemble le contenu de ce fichier :

```
192.168.0.21 slave-1
192.168.0.22 slave-2
```

Lancement de notre première commande Playbook

Prérequis

Ansible communique avec des machines distantes via le protocole SSH. Il faut donc vous assurez que vous pouvez vous connecter à vos différents nœuds distants de votre inventaire en utilisant le protocole SSH. Dans notre cas nous utiliserons la commande `ssh-copy-id` afin d'ajouter la clé SSH publique du serveur de contrôle au fichier `authorized_keys` sur nos systèmes gérés.

Par défaut, Vagrant vous donne l'accès en ssh depuis l'utilisateur **vagrant** avec le mot de passe **vagrant**. Débutons alors par la **configuration des accès de notre machine de contrôle** :

```
ssh-copy-id vagrant@192.168.0.21

# mdp : vagrant

ssh-copy-id vagrant@192.168.0.22

# mdp : vagrantCopier
```

L'inventaire

Ansible lit les informations sur les machines que vous souhaitez gérer à partir d'un fichier nommé **l'inventaire**. Bien que vous puissiez transmettre une adresse IP en paramètre avec la commande `ansible`, vous aurez besoin d'un inventaire pour profiter de la flexibilité et de la répétabilité complètes d'Ansible.

Vous pouvez définir vos machines esclaves dans le fichier inventaire qui est situé par défaut dans le fichier `inventory`. Votre inventaire peut stocker les adresses IP ou les noms de domaines complets de vos machines distantes (mais aussi des groupes, des alias ou des variables).

Dans cet exemple, dans mon fichier inventaire je vais insérer le nom DNS complet ma machine esclave numéro 1 et l'IP de ma machine esclave numéro 2 :

```
slave-1
192.168.0.22
```

Nous verrons plus tard dans ce chapitre **comment intégrer la notion de groupes dans notre inventaire**.

Exécutez vos premières commandes Ansible

D'abord qu'est-ce qu'un module ? Les modules sont comme de petits programmes qu'Ansible pousse depuis une machine de contrôle vers tous les hôtes distants. Les modules sont exécutés à l'aide de playbooks ou depuis la cli Ansible, et ils contrôlent des éléments tels que les services, les packages, les fichiers et bien plus.

Dans notre exemple, on utilisera le module `ping` qui envoie une requête ping à tous les nœuds de votre inventaire. Ouvrez donc un terminal depuis votre machine de contrôle et lancez-y la commande suivante :

```
ansible all -m ping -u vagrantCopier
```

Voici une liste d'explication des différentes options de notre commande :

- `all` : ici on demande à Ansible d'exécuter la commande sur tous les hôtes de notre inventaire
- `-m ping` : ici on lui demande d'utiliser le module `ping`
- `-u vagrant` : ici on lui demande de lancer notre module depuis l'utilisateur vagrant

Résultat :

```
192.168.0.22 | SUCCESS => {  
  
    "ansible_facts": {  
  
        "discovered_interpreter_python": "/usr/bin/python"  
  
    },  
  
    "changed": false,  
  
    "ping": "pong"  
  
}  
  
slave-1 | SUCCESS => {  
  
    "ansible_facts": {  
  
        "discovered_interpreter_python": "/usr/bin/python"  
  
    },  
  
    "changed": false,  
  
    "ping": "pong"  
  
}
```

Vous pouvez aussi **exécuter directement des commandes Linux sur Ansible**, en utilisant tout simplement l'option **-a**. Par exemple :

```
ansible all -a "whereis python" Copier
```

Résultat :

```
192.168.0.22 | CHANGED | rc=0 >>
```



```
python: /usr/bin/python3.6 /usr/bin/python3.6m /usr/bin/python
/usr/bin/python2.7 ...

slave-1 | CHANGED | rc=0 >>

python: /usr/bin/python3.6 /usr/bin/python3.6m /usr/bin/python
/usr/bin/python2.7 ...
```

Certaines commandes ou modules peuvent nécessiter une **élévation de privilèges**. Par exemple si vous tentez d'exécuter la commande ci-dessous depuis le compte **vagrant** vous obtiendrez une jolie erreur **Permission denied** :

```
ansible all -a "grep vagrant /etc/shadow" -u vagrantCopier
```

Erreur :

```
slave-1 | FAILED | rc=2 >>

grep: /etc/shadow: Permission denied non-zero return code

192.168.0.22 | FAILED | rc=2 >>

grep: /etc/shadow: Permission denied non-zero return code
```

Dans le cas où vous auriez besoin d'une élévation de privilèges pour l'exécution d'une commande, utilisez alors l'option **--become** (cette option exploite les outils existants d'élévation de privilèges comme `sudo`), comme suit :

```
ansible all -a "grep vagrant /etc/shadow" -u vagrant --becomeCopier
```

Résultat :

```
slave-1 | CHANGED | rc=0 >>

vagrant:$6$lrFYJT5b$Zm1DfFD2P8FWgB....

192.168.0.22 | CHANGED | rc=0 >>
```

```
vagrant:$6$lrFYJT5b$Zm1DfFD2P8FWgB...
```

Les groupes dans l'inventaire

Vous pouvez (et le ferez probablement) mettre chaque hôte distant dans un ou plusieurs groupes. Par exemple, vous pouvez intégrer vos machines de production dans un groupe nommé **[prod]** et vos machines de test dans un groupe nommé **[dev]**.

Mettons en pratique notre exemple. Voici donc à quoi ressemblera notre fichier inventaire :

```
[prod]

slave-1

[dev]

192.168.0.22

192.168.0.23
```

Pour lancer nos modules ou commandes sur un groupe d'hôtes, il suffit de remplacer l'option **all** par le nom de notre groupe :

```
ansible dev -m ping -u vagrantCopier
```

Résultat :

```
192.168.0.22 | SUCCESS => {

  "ansible_facts": {

    "discovered_interpreter_python": "/usr/bin/python"

  },

  "changed": false,

  "ping": "pong"

}
```

```
192.168.0.23 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python"
  },
  "changed": false,
  "ping": "pong"
}
```

Voici la commande pour **lancer vos modules sur différents groupes de votre inventaire Ansible**. Exemple sur le groupe **dev** et **prod**:

```
ansible dev,prod -m ping -u vagrantCopier
```

Utilisateur par défaut

Je ne sais pas vous, mais de mon côté j'en ai marre de taper à chaque fois l'option `-u vagrant`, j'aimerais bien que cette option soit prise en charge par défaut lors de l'exécution des modules/commandes Ansible. Ça tombe bien, car il est possible de définir un utilisateur par défaut sur votre inventaire avec l'option **ansible_user** :

```
slave-1 ansible_user=vagrant
192.168.0.22 ansible_user=vagrant
```

Dorénavant, il n'y a plus besoin d'utiliser l'option **-u** :

```
ansible all -m pingCopier
```

Résultat :

```
192.168.0.21 | SUCCESS => {
  "ansible_facts": {
```

```

        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}

192.168.0.22 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}

```

Les modules avec les options

Certains modules ansible nécessitent des paramètres supplémentaires pour une utilisation efficiente du module en question. Par exemple si on fait un tour sur la [documentation du module file](#) (module permettant de gérer les fichiers et les propriétés des fichiers.) on peut remarquer qu'il peut être utilisé avec différents paramètres additionnels :

Dans notre exemple on souhaite créer un fichier vide nommé `test.txt` dans le dossier `/home/vagrant/` et ne lui fournir que des droits de lecture (0644).

- **path** (obligatoire) : pour l'emplacement et le nom de notre fichier (chemin absolu + le nom de notre fichier)
- **state** : nous utiliserons la valeur **touch** pour obliger la création de notre fichier. message personnalisé et le paramètre
- **mode** : pour les autorisations que le fichier doit avoir dans notre cas ça sera le droit 0644

Pour cela, nous utiliserons l'option **-a** qui permet également de fournir des arguments complémentaires à l'option **-m** (module). Nous aurons ainsi la commande suivante :

```
ansible all -m file -a "path=/home/vagrant/test.txt state=touch mode=0644"Copier
```

Résultat :

```
192.168.0.21 | CHANGED => {
    "changed": true,
    "dest": "/home/vagrant/test.txt",
    "gid": 0,
    "group": "vagrant",
    "mode": "0644",
    "owner": "vagrant",
    "secontext": "unconfined_u:object_r:user_home_t:s0",
    "size": 0,
    "state": "file",
    "uid": 0
}

192.168.0.22 | CHANGED => {
    "changed": true,
    "dest": "/home/vagrant/test.txt",
    "gid": 0,
```

```
"group": "vagrant",

"mode": "0644",

"owner": "vagrant",

"secontext": "unconfined_u:object_r:user_home_t:s0",

"size": 0,

"state": "file",

"uid": 0

}
```

Parallélisme

Par défaut, Ansible utilise seulement 5 processus simultanés. Cependant, si vous avez plus de hôtes que la valeur définie par défaut pour le nombre de processus, Ansible prendra ainsi un peu plus de temps pour traiter tous vos hôtes. Vous pouvez augmenter le nombre de processus avec l'option **-f**. Dans cet exemple je vais utiliser 9 processus simultanés pour lancer le module ping :

```
ansible all -m ping -f 8Copier
```

Recueillir des informations (Facts) sur vos hôtes

Dans cette partie, nous allons **passer en revue les Facts sur Ansible** et comment vous pouvez les utiliser avec la commande `ansible`.

Les Facts sont des propriétés système qui sont collectées par Ansible lors de son exécution sur un système distant. Les Facts contiennent des détails utiles tels que le stockage, la version (mineur et majeur) de l'OS, configuration du réseau sur un système cible. Ils peuvent être exportés vers un fichier en tant que type de rapport système, ou ils peuvent être utilisés pendant l'exécution d'un playbook Ansible pour conditionner l'exécution de vos tâches.

Voici la commande pour voir tous les Facts :

```
ansible all -m setupCopier
```

Vous pouvez également filtrer la sortie pour afficher uniquement certains Facts en utilisant le paramètre **filter**. Dans cet exemple je vais récupérer des informations réseau de mes hosts :

```
ansible all -m setup -a 'filter=*ip*'Copier
```

Résultat :

```
192.168.0.21 | SUCCESS => {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "192.168.0.21"
    ],
    "ansible_all_ipv6_addresses": [],
    "ansible_default_ipv4": {
      "address": "192.168.0.21",
      ...
      "gateway": "192.168.0.1",
      "interface": "wlp1s0",
      "macaddress": "00:40:56:94:5f:8e",
      "netmask": "255.255.255.0",
      ...
    },
    ...
  }
}
```

Pour plus d'informations, consultez la documentation du [module setup](#).

Conclusion

Nous nous sommes penchés dans ce chapitre sur l'exécution des modules et des commandes depuis la cli d'ansible. Vous êtes désormais capable de vous amuser avec les **différents modules** qu'offre Ansible. Dans notre futur chapitre nous apprendrons à créer et à exécuter notre premier playbook Ansible, ce qui nous facilitera ainsi la gestion de nos modules.

[Chapitre précédent](#) [Chapitre suivant](#)

[Mon jeu](#)

Création de notre playbook Ansible (stack LAMP)

Dans ce chapitre nous apprendrons à créer notre premier playbook Ansible en créant une stack LAMP.

Introduction

Nous avons étudié dans le chapitre précédent comment lancer des modules en utilisant seulement la cli ansible. Dans ce chapitre, nous découvrirons un autre moyen d'**exploiter les modules ansible à travers un fichier qu'on nomme le Playbook**.

Pourquoi les Playbooks

Par rapport aux modules utilisés précédemment exclusivement depuis la cli Ansible, les Playbooks sont utilisés dans des scénarios complexes et offrent une flexibilité accrue très bien adaptée au déploiement d'applications complexes. De plus les Playbooks sont plus susceptibles d'être gardés sous une source de contrôle (git) et d'assurer des configurations conformes aux spécifications de votre entreprise.

Les Playbooks sont **exprimés au format YAML** et ont un minimum de syntaxe, qui essaie intentionnellement de ne pas être un langage de programmation ou un script, mais plutôt un modèle de configuration (même s'il reste possible d'y intégrer des boucles et des conditions).

Comme ça utilise la syntaxe yaml, il faut faire attention à bien respecter l'indentation (**c'est 2 espaces et non une tabulation pour faire une indentation**).

Les Playbooks contiennent des tâches qui sont exécutées séquentiellement par l'utilisateur sur une machine particulière (ou un groupe de machine). Une tâche (Task) n'est rien de plus qu'un appel à un module ansible.

Dans ce chapitre nous allons **créer une stack LAMP à partir d'un Playbook Ansible**. Ce mini-projet va nous permettre d'examiner un exemple d'arborescence d'un projet Ansible et de découvrir quelques modules intéressants d'Ansible.

Récupérez d'abord le projet complet [ici](#) et ensuite sans plus attendre, commençons par les explications !

Structure du projet

Quels sont les objectifs de notre Playbook ?

Ce Playbook Ansible nous fournira une alternative à l'exécution manuelle de la procédure d'installation générale d'un serveur LAMP (Linux, Apache, MySQL et PHP). L'exécution de ce Playbook automatisera donc les actions suivantes sur nos hôtes distants :

- Côté serveur web :
 - Installer les packages apache2, php et php-mysql
 - Déployer les sources de notre application dans notre serveur web distant
 - S'assurer que le service apache est bien démarré
- Côté serveur base de données :
 - Installer les packages mysql
 - Modifier le mot de passe root
 - Autoriser notre serveur web à communiquer avec la base de données
 - Configurer notre table mysql avec les bonnes colonnes et autorisations

Arborescence du projet

Voici à quoi ressemble l'arborescence de notre projet une fois téléchargé :

```
|— files
|
|   └─ app
```

```
|   |— index.php
|   |— validation.php
|
|— templates
|   |— db-config.php.j2
|   |— table.sql.j2
|
|— vars
|   |— main.yml
|
|— ansible.cfg
|— hosts
|— Playbook.yml
```

- `Playbook.yml` : fichier Playbook contenant les tâches à exécuter sur le ou les serveurs distants.
- `vars/main.yml` : fichier pour nos variables afin de personnaliser les paramètres du Playbook (on peut aussi déclarer des variables dans le fichier Playbook).
- `hosts` : Fichier inventaire de notre Playbook.
- `ansible.cfg` : par défaut ansible utilise le fichier de configuration `/etc/ansible/ansible.cfg` mais on peut surcharger la config en rajoutant un fichier nommé `ansible.cfg` à la racine du projet.
- `files/` : contient les sources de notre stack LAMP qui seront par la suite destinés à être traités par le module `copy`.
- `templates/` : contient des modèles de configurations dynamiques au format `jinja` qui sont destinés à être traités par le module `template`.

Le fichier inventaire

Pour ce projet, j'ai décidé de me séparer du fichier inventaire situé par défaut dans `/etc/ansible/hosts` et de créer mon propre fichier `hosts` à la racine du projet, où je me suis

permis de séparer le serveur de base de données par rapport au serveur web, voici donc à quoi ressemble notre nouveau fichier inventaire :

```
[web]

slave-1 ansible_user=vagrant

[db]

slave-2 ansible_user=vagrantCopier
```

Pour que notre nouveau fichier inventaire personnalisé soit pris en compte par votre Playbook, il faut au préalable modifier la valeur de la variable `inventory` située dans notre fichier de configuration ansible.

Par défaut ce fichier se situe dans le fichier `/etc/ansible/ansible.cfg`. Mais pour faire les choses dans les règles de l'art, nous allons laisser la configuration par défaut choisie par Ansible et créer notre propre fichier de configuration à la racine du projet. Dans notre nouveau fichier de config nous surchargerons uniquement la valeur de la variable `inventory`, ce qui nous donne le fichier `ansible.cfg` suivant :

```
[defaults]

inventory = ./hosts
```

Explication du Playbook

Pour commencer, voici déjà le contenu de notre Playbook :

```
---

# WEB SERVER

- hosts: web

  become: true

  vars_files: vars/main.yml
```

```
tasks:

- name: install apache and php last version

  apt:

    name:

      - apache2

      - php

      - php-mysql

    state: present

    update_cache: yes

- name: Give writable mode to http folder

  file:

    path: /var/www/html

    state: directory

    mode: '0755'

- name: remove default index.html

  file:

    path: /var/www/html/index.html

    state: absent
```

- name: upload web app source

 - copy:

 - src: app/

 - dest: /var/www/html/

- name: deploy php database config

 - template:

 - src: "db-config.php.j2"

 - dest: "/var/www/html/db-config.php"

- name: ensure apache service is start

 - service:

 - name: apache2

 - state: started

 - enabled: yes

DATABASE SERVER

- hosts: db

```
become: true

vars_files: vars/main.yml

vars:

    root_password: "my_secret_password"

tasks:

- name: install mysql

    apt:

        name:

            - mysql-server

            - python-mysqldb # for mysql_db and mysql_user modules

        state: present

        update_cache: yes

- name: Create MySQL client config

    copy:

        dest: "/root/.my.cnf"

        content: |

            [client]

            user=root

            password={{ root_password }}
```

mode: 0400

- name: Allow external MySQL connexions (1/2)

lineinfile:

path: /etc/mysql/mysql.conf.d/mysqld.cnf

regexp: '^skip-external-locking'

line: "# skip-external-locking"

notify: Restart mysql

- name: Allow external MySQL connexions (2/2)

lineinfile:

path: /etc/mysql/mysql.conf.d/mysqld.cnf

regexp: '^bind-address'

line: "# bind-address"

notify: Restart mysql

- name: upload sql table config

template:

src: "table.sql.j2"

dest: "/tmp/table.sql"

```
- name: add sql table to database

mysql_db:

    name: "{{ mysql_dbname }}"

    state: present

    login_user: root

    login_password: '{{ root_password }}'

    state: import

    target: /tmp/table.sql


- name: "Create {{ mysql_user }} with all {{ mysql_dbname }}
privileges"

mysql_user:

    name: "{{ mysql_user }}"

    password: "{{ mysql_password }}"

    priv: "{{ mysql_dbname }}.*:ALL"

    host: "{{ webserver_host }}"

    state: present

    login_user: root

    login_password: '{{ root_password }}'

    login_unix_socket: /var/run/mysqld/mysqld.sock


handlers:
```



```
- name: Restart mysql

  service:

    name: mysql

    state: restartedCopier
```

Comme dit précédemment, nous avons choisi de séparer dans notre nouveau fichier inventaire le serveur de base de données par rapport à notre serveur web. Notre Playbook doit continuer dans cette voie en ciblant d'abord le serveur Web, puis le serveur de base de données (ou inversement).

Serveur web

Dans cette partie, nous nous intéresserons particulièrement à la partie Web de notre playbook.

Partie hosts

Pour chaque jeu dans un Playbook, vous pouvez choisir les machines à cibler pour effectuer vos tâches. Dans notre cas on commence par cibler notre serveur web :

```
---

- hosts: webCopier
```

Information

Les 3 tirets au début d'un fichier yaml ne sont pas obligatoires.

élévation de privilèges

On demande au moteur Ansible d'exécuter toutes nos tâches en tant qu'utilisateur root grâce au mot-clé **become** :

```
become: trueCopier
```

Vous pouvez également utiliser le mot-clé **become** sur une tâche particulière au lieu de l'ensemble de vos tâches :

```
tasks:

- service:

    name: nginx

    state: started

    become: yesCopier
```

Variables

Concernant les variables, vous avez le choix entre les placer directement depuis le mot-clé **vars**, ou vous pouvez les charger depuis un fichier en utilisant le mot-clé **vars_files** comme ceci :

```
vars_files: vars/main.ymlCopier
```

Voici le contenu du fichier de variables :

```
---

mysql_user: "admin"

mysql_password: "secret"

mysql_dbname: "blog"

db_host: "192.168.0.22"

webserver_host: "192.168.0.21"Copier
```

- **mysql_user** : l'utilisateur de notre base de données mysql qui exécutera nos requêtes SQL depuis notre application web.
- **mysql_password** : le mot de passe de l'utilisateur de notre base de données mysql.
- **mysql_dbname** : le nom de notre base de données.
- **db_host** : l'ip de notre machine mysql (utile pour la partie configuration mysql de notre application web).
- **webserver_host** : l'ip de la machine web (utile pour autoriser uniquement l'ip du serveur web à communiquer avec notre base de données).

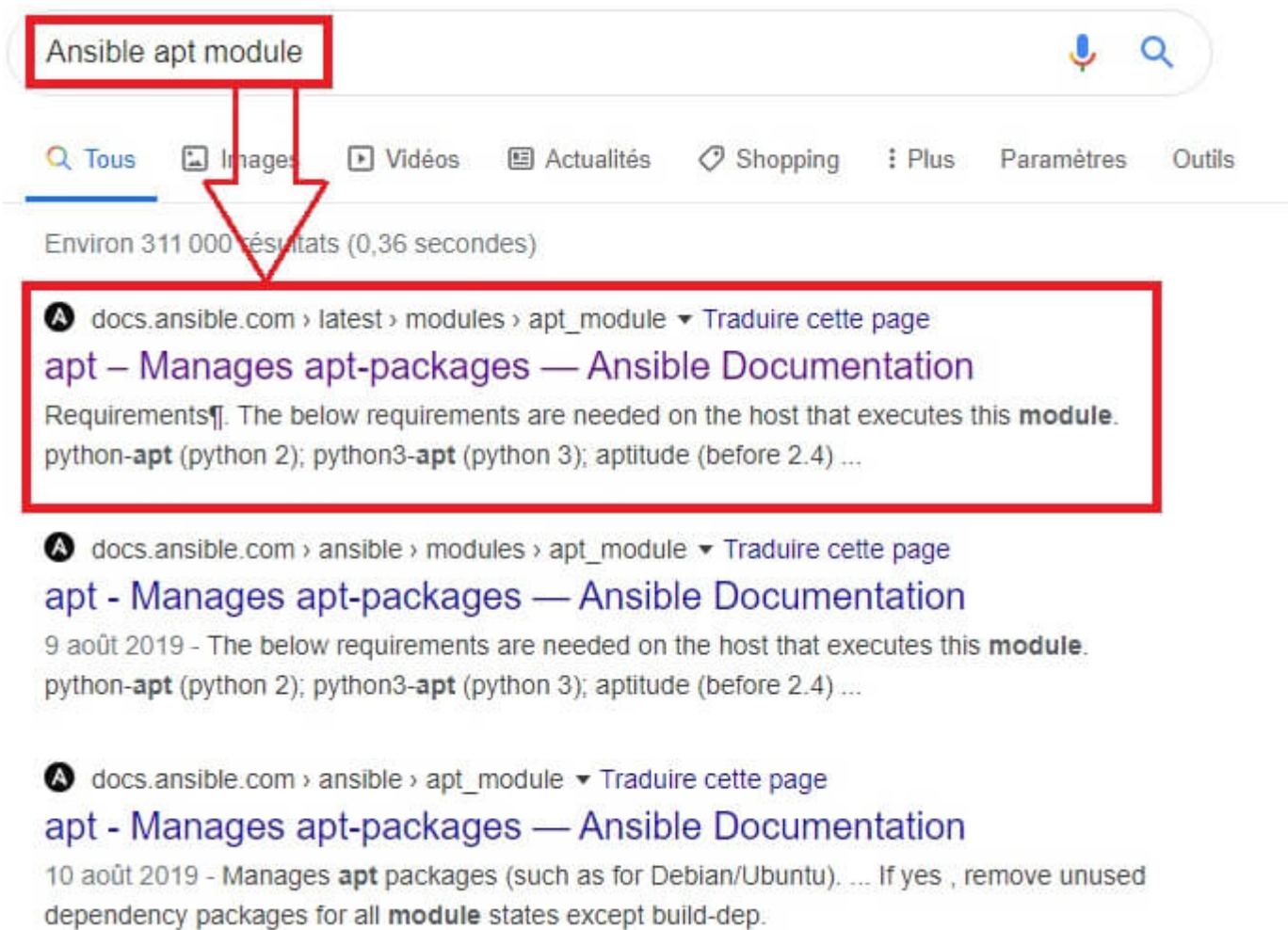
Les tâches

Chaque hôte contient une liste de tâches au-dessous du mot-clé **tasks**. Les tâches sont exécutées dans l'ordre, une à la fois, sur toutes les machines correspondant au modèle d'hôte avant de passer à la tâche suivante.

Le but de chaque tâche est d'exécuter un module Ansible avec des arguments très spécifiques. Les variables peuvent également être utilisées dans les arguments des modules.

Chaque tâche peut débuter avec le mot-clé **name**, qui est simplement une brève description de votre tâche. Cette information s'affichera à la sortie de l'exécution du Playbook, son but principal est de pouvoir distinguer et décrire vos différentes tâches. Il est donc utile de fournir de bonnes petites descriptions pour chaque tâche. Si le champ n'est pas saisi alors le nom du module sera utilisée comme sorties. Au-dessous du mot-clé **name**, vous insérez le nom du module avec ses différents paramètres.

Dans notre projet, notre premier besoin consiste à installer les packages apache2, php et php-mysql avec le gestionnaire de paquets apt. Et peut-être que vous vous demandez comment j'ai faits pour trouver le module adéquat ? La réponse est "Google !", en effet Google est votre meilleur ami (ou Bing, Ecosia, Qwant, DuckDuckGo, etc ...) ! J'ai tapé sur le moteur de recherche les mots-clés suivants "Ansible apt module" et j'ai cliqué sur le premier lien fourni par Google ([celui-ci](#)).



Sur cette page vous avez le Synopsis qui vous fournit une description courte du module :

Synopsis

- Manages *apt* packages (such as for Debian/Ubuntu).

Si on traduit mot par mot le Synopsis, nous aurons la phrase suivante : "Gère les paquets apt (comme pour Debian/Ubuntu)".

Ça correspond parfaitement à notre besoin ! Maintenant l'étape suivante consiste à rechercher les différents paramètres que propose le module apt. Dans notre cas on cherche à installer la dernière version des packages apache2, php et php-mysql. En lisant la documentation on peut vite s'apercevoir qu'il existe les options suivantes :

- **name** (type: liste) : liste de noms de packages (on peut aussi spécifier la version du package ex curl=1.6 ou curl=1.0*).
- **state** (type: string) : indique l'état du package, voici un exemple des valeurs possibles :
 - **latests** : assure que c'est toujours la dernière version qui est installée.

- **present** : vérifie si le package est déjà installé, si c'est le cas il ne fait rien, sinon il l'installe.
 - **absent** : supprime le package s'il est déjà installé.
- **update_cache** (type: booléen) : exécute l'équivalent de la commande `apt-get update` avant l'installation des paquets.

Si on combine toutes ces informations on se retrouve avec la tâche suivante :

```
- name: install apache and php last version

apt:

  name:

    - apache2

    - php

    - php-mysql

  state: present

  update_cache: yesCopier
```

J'ai utilisé la même méthodologie de recherches pour retrouver le reste des tâches de ce Playbook.

Les types en Yaml :

J'aimerais simplement prendre quelques instants pour vous expliquer l'**utilisation de quelques types de variables dans le langage Yaml**. En effet, vous avez différentes façons pour valoriser vos variables selon leurs types.

Par exemple, pour le paramètre **name** du module apt qui est de type **list**, on peut aussi l'écrire comme une liste sur python, soit :

```
- name: install apache and php last version

apt:

  name: ['apache2', 'php', 'php-mysql']
```

```
state: present

update_cache: yesCopier
```

Concernant les types booléens, comme pour le paramètre `update_cache`, vous pouvez spécifier une valeur sous plusieurs formes:

```
update_cache: yes

update_cache: no

update_cache: True

update_cache: TRUE

update_cache: falseCopier
```

Vous avez aussi la possibilité de raccourcir la tâche d'un module. Prenons par exemple la tâche suivante :

```
tasks:

- name: deploy test.cfg file

  copy:

    src: /tmp/test.cfg

    dest: /tmp/test.cfg

    owner: root

    group: root

    mode: 0644Copier
```

Pour la raccourcir, il suffit de mettre tous vos paramètres sur une seule ligne (possibilité de faire un saut à la ligne) et de remplacer les `:` par des `=`. Ce qui nous donne :

```
tasks:

- name=deploy test.cfg file
```

```
copy: src=/tmp/test.cfg dest=/tmp/test.cfg
```

```
owner=root group=root mode=0644Copier
```

Idempotence

Les modules doivent être idempotents, c'est-à-dire que l'exécution d'un module plusieurs fois dans une séquence doit avoir le même effet que son exécution unique.

Les modules fournis par Ansible sont en général idempotents, mais il se peut que vous ne trouveriez pas des modules répondant parfaitement à votre besoin, dans ce cas vous passerez probablement par le module [command](#) ou [shell](#) qui vont vous permettre ainsi d'exécuter vos propres commandes shell.

Si vous êtes amené à travailler avec ces modules dans votre Playbook, il faut faire attention à ce que vos tâches soient idempotentes, la réexécution du Playbook doit être sûre.

Cette parenthèse étant fermée, on peut continuer par l'explication de notre Playbook

Suite des tâches

- ~~Installer les packages apache2, php et php-mysql~~
- Déployer les sources de notre application dans notre serveur web distant
- S'assurer que le service apache est bien démarré

Pour déployer les sources de notre application, il faut au préalable donner les droits d'écriture sur le dossier `/var/www/html`, pour cela rien de mieux que d'utiliser le module `file` ([documentation ici](#)) qui permet entre autres de gérer les propriétés des fichiers/dossiers.

```
- name: Give writable mode to http folder
```

```
file:
```

```
path: /var/www/html
```

```
state: directory
```

```
mode: '0755'Copier
```

J'enchaîne ensuite par la suppression de la page d'accueil du serveur apache, en éliminant le fichier `index.html`.

```
- name: remove default index.html

  file:

    path: /var/www/html/index.html

    state: absent
```

Une fois que nous avons les droits d'écriture dans ce dossier, la prochaine étape comprend l'upload des sources de notre application dans le dossier `/var/www/html` de notre serveur web distant.

Un des modules qui peut répondre à une partie de notre besoin, est le module copy ([Documentation ici](#)) qui permet de copier des fichiers ou des dossiers de notre serveur de contrôle vers des emplacements distants.

```
- name: upload web app source

  copy:

    src: app/

    dest: /var/www/html/Copier
```

Peut-être que vous l'avez remarqué, mais je n'ai pas eu besoin de fournir le dossier `files` dans le chemin du paramètre `src`, car ce dossier est spécialement conçu pour que le module copy recherche dedans automatiquement nos différents fichiers ou dossiers à envoyer (si vous déposez vos fichiers dans un autre emplacement, il faut dans ce cas que vous insériez le chemin relatif ou absolu complet)

Fichier de configuration dynamique (Jinja2)

Cependant, nous allons être confrontés à un problème. En effet, nous avons déclaré des variables dans le fichier `vars/main.yml`, dont quelques-unes pour se connecter à notre base de données. Comme par exemple l'utilisateur et le mot de passe mysql.

Il nous faut donc un moyen pour que notre fichier php, qui permet la connexion à la base données, soit automatiquement en accord avec ce que l'utilisateur a décidé de valoriser dans le fichier `vars/main.yml`.

La solution à ce problème est l'utilisation du module template ([Documentation ici](#)). Il permet de faire la même chose que le module copy. Cependant, ce module permet de **modifier dynamiquement un fichier** avant de l'envoyer sur le serveur cible. Pour ce faire les fichiers sont écrits et traités par le [langage Jinja2](#).

Je ne rentrerai pas trop dans les détails de ce langage, mais concernant notre besoin, où il s'agit de remplacer certaines valeurs de notre fichier php, on exploitera les variables dans le langage Jinja2.

Vous pouvez effectivement, jouer avec les variables dans les modèles jinja qui seront au préalable valorisées par le module template. Il suffit donc dans notre fichier jinja de reprendre le même nom que notre variable Ansible et de la mettre entre deux accolades, voici par exemple le contenu de notre template `db-config.php.j2` :

```
<?php

const DB_DSN = 'mysql:host={{ db_host }};dbname={{ mysql_dbname }}';

const DB_USER = "{{ mysql_user }}";

const DB_PASS = "{{ mysql_password }}";


$options = array(

    PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8", // encodage
    utf-8

    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION, // gérer les
    erreurs en tant qu'exception
```

```
PDO::ATTR_EMULATE_PREPARES => false // faire des vrais requêtes
préparées et non une émulation

);Copier
```

Par exemple, pour ce même fichier, le module template remplacera `{{ mysql_user }}` par la valeur de la variable `mysql_user` située dans le fichier `vars/main.yml` avant de l'envoyer sur notre serveur web.

Ce qui nous donne la tâche suivante :

```
- name: deploy php database config

  template:

    src: "db-config.php.j2"

    dest: "/var/www/html/db-config.php"Copier
```

Information

Comme pour le module copy, ici nul besoin de fournir le dossier `templates/` dans le chemin du paramètre `src`, car le module template recherche automatiquement nos différents fichiers jinja dans ce dossier (si vous déposez vos fichiers dans un autre emplacement, il faut dans ce cas que vous insériez le chemin relatif ou absolu complet).

Le module service

- Installer les packages `apache2`, `php` et `php-mysql`
- Déployer les sources de notre application dans notre serveur web distant
- S'assurer que le service apache est bien démarré

Quand il s'agit de gérer des services Linux, il faut penser directement au module `service` ([Documentation ici](#)). Il reste très simple à utiliser, il suffit simplement de lui fournir le nom du service à gérer dans le paramètre `name`, ainsi que l'état souhaité du service dans le paramètre `state`, qui peut contenir les valeurs suivantes :

- `reloaded` : recharger le service sans le stopper
- `restarted` : redémarrage du service (arrêt + démarrage)
- `started` : si nécessaire le service sera démarré
- `stopped` : si nécessaire le service sera arrêté

Voici à quoi ressemble notre dernière tâche de notre serveur web :

```
- name: ensure apache service is start

  service:

    name: apache2

    state: started

    enabled: yesCopier
```

Voilà, dorénavant les tâches de notre serveur web sont finalisées. On s'attaquera maintenant à l'hôte de base de données.

Serveur de base de données

Comme pour notre serveur web, on commence d'abord par préparer le terrain pour les différentes tâches de notre hôte de base de données.

Préparation des tâches

Comme pour notre serveur web, nous utilisons le mot-clé **become** pour l'élévation de privilèges et le mot-clé **vars_files** pour inclure les variables situées dans le fichier **vars/main.yml**. Cependant, j'ai choisi de placer une variable uniquement utilisable par les tâches de notre serveur de base données, soit la variable **root_password**. Ce qui nous donne la configuration suivante :

```
- hosts: db

  become: true

  vars_files: vars/main.yml

  vars:

    root_password: "my_secret_password"Copier
```

Installation des paquets

Pour rappel, voici les étapes à effectuer sur notre serveur de base de données :

- Installer les packages mysql
- Modifier le mot de passe root
- Autoriser notre serveur web à communiquer avec la base de données
- Configurer notre table mysql avec les bonnes colonnes et autorisations

```
- name: install mysql

apt:

  name:

    - mysql-server

    - python-mysqldb # for mysql_db and mysql_user modules

  state: present

  update_cache: yes Copier
```

On utilise une nouvelle fois le module apt afin d'installer nos différents packages. Le package mysql-server nous permet d'installer notre base de données relationnelle. Ensuite on installe le package python-mysqldb qui est nécessaire pour utiliser plus tard le module [mysql_user](#) and [mysql_db](#).

Modification du mot de passe root

Il existe différentes manières pour modifier le mot de passe mysql du compte root. Pour ma part, j'ai choisi de surcharger le fichier de configuration mysql par défaut. Pour cela j'ai créé sur le serveur distant un fichier `.my.cnf` à l'emplacement `/root/`. Pour cela, j'ai utilisé le module copy, mais cette fois-ci avec le paramètre `content` à la place du paramètre `src`. Lorsque ce paramètre est utilisé à la place de `src`, on peut comme son nom l'indique définir le contenu d'un fichier directement sur la valeur spécifiée. Ce qui nous donne :

```
- name: Create MySQL client config

copy:
```

```
dest: "/root/.my.cnf"

content: |

    [client]

    user=root

    password={{ root_password }}Copier
```

La valeur `{{ root_password }}` sera bien sûr remplacée par la valeur de variable `root_password` soit dans cet exemple la valeur `"my_secret_password"`.

Information

Pour créer un contenu multiligne il faut utiliser le caractère `|` après le nom du module, comme j'ai pu le faire pour cet exemple.

Autorisation des connexions externes

Pour autoriser les communications externes sur notre serveur mysql, On peut commenter la ligne commençant par `bind-address` et `skip-external-locking` dans le fichier de configuration `/etc/mysql/mysql.conf.d/mysqld.cnf` du serveur mysql distant.

Quand il s'agit de faire des modifications sur des fichiers distants, le module le plus adapté reste le module `lineinfile` ([Documentation ici](#)).

C'est un module spécialement conçu pour gérer les lignes dans les fichiers texte. Dans notre cas il nous est demandé de commenter des lignes commençant par un mot bien particulier. Pour cela, nous aurons besoin des expressions régulières, soit le paramètre `regexp` du module `lineinfile` et le paramètre `line` pour la ligne de remplacement. Ce qui nous donne le résultat suivant :

```
- name: Allow external MySQL connexions (1/2)

lineinfile:

    path: /etc/mysql/mysql.conf.d/mysqld.cnf

    regexp: '^skip-external-locking'
```

```
    line: "# skip-external-locking"

    notify: Restart mysql

- name: Allow external MySQL connexions (2/2)

  lineinfile:

    path: /etc/mysql/mysql.conf.d/mysqld.cnf

    regexp: '^bind-address'

    line: "# bind-address"

  notify: Restart mysqlCopier
```

notify et handlers

Vous remarquerez que j'utilise le mot-clé **notify** (notification en français). Ce sont tout simplement des actions (tâches) qui sont déclenchées à la fin de chaque bloc de tâches.

Ces actions sont répertoriées dans la partie **handlers**. Les handlers sont des listes de tâches, qui ne diffèrent pas vraiment des tâches normales, qui sont référencées par un nom globalement unique et qui sont déclenchées par le mot-clé **notify**.

Dans notre cas c'est le handler suivant qui est déclenché à la fin de notre tâche :

```
handlers:

- name: Restart mysql

  service:

    name: mysql

    state: restartedCopier
```

Création et configuration de notre base de données

Notre serveur mysql est dorénavant démarré et configuré pour accepter des connexions externes. La prochaine étape est de créer notre table et notre utilisateur mysql avec les privilèges appropriés. Pour ce faire, nous avons besoin de deux modules : le module template pour adapter notre fichier sql (fichier qui contient la structure de notre base de données) avant de l'envoyer au serveur distant, qui sera par la suite exécuté par le module mysql_db ([Documentation ici](#)) :

```
- name: upload sql table config

  template:

    src: "table.sql.j2"

    dest: "/tmp/table.sql"


- name: add sql table to database

  mysql_db:

    name: "{{ mysql_dbname }}"

    state: present

    login_user: root

    login_password: '{{ root_password }}'

    state: import

    target: /tmp/table.sqlCopier
```

Information

Bien sûr notre base de données sera créée grâce au paramètre **name** avant d'exécuter notre fichier sql défini sur le paramètre **target** (ce qui est assez logique sinon on se retrouvera avec des erreurs ☐)

La dernière étape de configuration est de créer notre utilisateur mysql défini dans le fichier `vars/main.yml`, et de lui fournir les autorisations uniquement sur notre base de données fraîchement créée précédemment. Il ne faut pas oublier aussi d'autoriser uniquement notre serveur web à communiquer avec notre base de données. Toutes ces exigences peuvent être résolues grâce au module `mysql_user` ([Documentation ici](#)). Ce qui nous donne la tâche suivante :

```
- name: "Create {{ mysql_user }} with all {{ mysql_dbname }}
  privileges"

mysql_user:

  name: "{{ mysql_user }}"

  password: "{{ mysql_password }}"

  priv: "{{ mysql_dbname }}.*:ALL"

  host: "{{ webserver_host }}"

  state: present

  login_user: root

  login_password: '{{ root_password }}'

  login_unix_socket: /var/run/mysqld/mysqld.sockCopier
```

Test

Voici la commande pour **lancer votre playbook** :

```
ansible-playbook playbook.ymlCopier
```

Si tout c'est bien déroulé, alors visitez la page suivante http://IP_SERVEUR_WEB, et vous obtiendrez la page d'accueil suivante :

Articles

Nouveau article

Titre *

Nom de l'auteur *

Contenu *

Le lorem ipsum est, en imprimerie, une
remplacer le faux-texte dès qu'il est prè

Envoyer

Liste d'articles

Pour tester la connexion à notre base de données, je vais appuyer sur le bouton "Envoyer" pour valider le formulaire et rajouter mon article à la base de données, ce qui nous donne le résultat suivant :

Liste d'articles

Test

24/01/20 19:26

Le lorem ipsum est, en imprimerie, une suite de mots sans signification utilisée à titre provisoire venant remplacer le faux-texte dès qu'il est prêt ou que la mise en page est achevée. Généralement on utilise du Lorem Ipsum.

— *hatim*

Conclusion

Je pense que vous l'aurez compris, le Playbook est un fichier permettant de faciliter la gestion de nos modules Ansible. Nous verrons dans le prochain chapitre comment améliorer notre playbook avec les conditions et nous aborderons également les boucles dans les playbooks.

Création d'un playbook multi distributions

Dans ce chapitre, nous verrons comment adapter notre playbook pour différentes distributions Linux grâce aux variables Facts, conditions, boucles et les variables enregistrées.

Introduction

Dans le chapitre précédent, nous avons conçu un Playbook Ansible permettant l'installation d'une stack LAMP. Je vous avais promis d'améliorer notre Playbook grâce à l'exploitation des conditions, des variables enregistrées et des boucles dans Ansible.

Le problème avec notre Playbook précédent, c'est qu'il ne peut fonctionner que sur des machines de la famille Debian. Puisque nous avons utilisé par exemple que le module apt afin d'installer les différents packages de notre stack LAMP. Ce module est particulièrement conçu pour les machines de la famille Debian.

Il serait préférable d'**adapter notre playbook pour d'autres familles de distribution**. Nous allons donc dans cet article adapter notre playbook pour les machines de famille RedHat : RHEL, CENTOS et Fedora. Nous allons aussi profiter de cet article pour **apporter plus de flexibilité à notre Playbook** grâce aux systèmes de boucles et des variables enregistrées.

Êtes-vous prêts pour toutes ces améliorations ? Commençons alors !

Préparation des nouvelles machines

Jusqu'ici, nous avons travaillé avec des machines Ubuntu et nous les avons créées avec l'outil [vagrant](#). Si vous le souhaitez, vous pouvez les supprimer pour en créer des nouvelles avec l'image Fedora dans le but de tester notre nouveau Playbook. Pour ce faire, il suffit d'utiliser une nouvelle image dans notre fichier Vagrantfile, voici à quoi ressemblera ce nouveau fichier :

```
#
#####

# ##### CONFIGURATION VARIABLES
#####

#
#####

IMAGE_NAME = "fedora/30-cloud-base" # Image to use

MEM = 2048 # Amount of RAM
```

```
CPU = 1                                # Number of processors

SLAVE_NBR = 2                          # Number of slaves node

NETWORK_ADAPTER="wlp1s0"              # Bridge network adapter


Vagrant.configure("2") do |config|

  config.ssh.insert_key = false

  # RAM and CPU config

  config.vm.provider "virtualbox" do |v|

    v.memory = MEM

    v.cpus = CPU

  end

  # Slave node config

  (1..SLAVE_NBR).each do |i|

    config.vm.define "slave-#{i}" do |slave|

      # OS and Hostname

      slave.vm.box = IMAGE_NAME

      slave.vm.hostname = "slave-#{i}"

      slave.vm.network "public_network", bridge:
NETWORK_ADAPTER, ip: "192.168.0.2#{i}"
```

```
end

end

endCopier
```

Placez-vous au même niveau que votre fichier VagrantFile. Provisionnez ensuite vos nouvelles machines virtuelles avec la commande suivante :

```
vagrant upCopier
```

N'oubliez pas de rajouter la clé publique de votre machine de contrôle dans vos machines cibles :

```
ssh-copy-id vagrant@[IP_MACHINE_1]

ssh-copy-id vagrant@[IP_MACHINE_2]Copier
```

Rappel

Mon environnement de travail n'a pas changé pour ce cours, j'aurai toujours comme IP statique **192.169.0.21** avec l'alias **slave-1** pour mon nœud distant numéro 1 et l'IP **192.168.0.22** avec l'alias **slave-2** pour mon nœud distant numéro 2.

Amélioration du Playbook

Les Facts

Avant d'utiliser le [module yum](#) pour l'installation de nos packages LAMP sur nos machines de la famille Redhat, nous avons besoin de connaître au préalable la distribution utilisée sur nos machines cibles.

Avant de modifier notre Playbook rappelez-vous que nous avons déjà eu l'occasion dans ce [chapitre](#) de récupérer des informations systèmes sur nos hôtes distants grâce aux Facts. Pour ce faire, nous avons utilisé le module Ansible [module setup](#). Réutilisons ce même module depuis une commande Ansible sur l'un de nos systèmes distants :

```
ansible slave-1 -m setupCopier
```

Résultat :

```
slave-1 | SUCCESS => {  
  
  "ansible_facts": {  
  
    "ansible_all_ipv4_addresses": [  
  
      "10.0.2.15",  
  
      "192.168.0.21"  
  
    ]  
  
    ...  
  
    ...  
  
    "ansible_virtualization_role": "guest",  
    "ansible_virtualization_type": "virtualbox",  
    "discovered_interpreter_python": "/usr/bin/python3",  
    "gather_subset": [  
  
      "all"  
  
    ],  
  
    "module_setup": true  
  
  },  
  
  "changed": false  
  
}
```

La liste de résultat est très longue, par conséquent nous filtrerons le résultat grâce au paramètre **filter** en se basant sur l'OS. Nous aurons ainsi la commande suivante :

```
ansible slave-1 -m setup -a "filter=*os*"Copier
```

Résultat :

```
slave-1 | SUCCESS => {  
  
  "ansible_facts": {  
  
    "ansible_bios_date": "12/01/2006",  
  
    "ansible_bios_version": "VirtualBox",  
  
    "ansible_hostname": "slave-1",  
  
    "ansible_hostnqn": "",  
  
    "ansible_os_family": "RedHat",  
  
    "ansible_ssh_host_key_ecdsa_public":  
"AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBBCI9gA8vQhwIRvv  
driZH9MFnRuZCVv5KnKGZo6RUEt4Ljk//e0HX1a16SZFljnxEANig7RibqDvL+3Ky2rp  
TGr4=",  
  
    "ansible_ssh_host_key_ed25519_public":  
"AAAAC3NzaC1lZDI1NTE5AAAAIIIs2pEEEGym6W+0VWZP3pFPfyUzxXXfkeI43PtsXky4  
W",  
  
    "ansible_ssh_host_key_rsa_public":  
"AAAAB3NzaC1yc2EAAAADAQABAAQACrtN15m8q3EkuKqL9F6V0EDex6PjNTVQphVnv  
pGAOLooPJpnwQj1pTMdMgueVEjZc4Q8TTWUbA7ike59or5CanEvwJ9gYoN/dTWzyKSe  
NlUi8Yx/ka7K0Ax7Z0jwU0jA6P59w7m1AfyoN/EKLgOmSX9k6FjNEU+D+wiwf+KVnDaN  
TNBlpGRWHhnTixnL4qtC9f12ufP2QEmIWo2KZN3mLhkI0IUtijTK/u0U+lwjXiGNIN+Z  
LWYBDtyLwWFbyIAci1fofyM7ntez5yPYsmyeP1rWFJobl4KhXchEjrIWH/SrSUq1QYDl  
FbvE8B04p0quLLef8+3byvuQFmWqdj1kF",  
  
    "ansible_user_gecos": "",  
  
    "discovered_interpreter_python": "/usr/bin/python3"  
  
  },  
  
  "changed": false  
  
}
```

Parfait! Nous obtenons exactement ce qu'on souhaite, à savoir la famille d'os à laquelle appartient notre machine distante depuis la variable `ansible_os_family`. L'étape suivante consiste à utiliser cette variable dans notre Playbook.

Les conditions

Dans cette partie nous allons appliquer une refonte de notre Playbook afin de s'adapter également à la famille RedHat au moyen des conditions. Commencez par récupérer notre ancien projet complet en [cliquant ici](#).

Ouvrez le fichier `playbook.yml` et dans notre première tâche testons la variable `ansible_os_family` en utilisant le `module debug`, comme suit :

```
---

# WEB SERVER

- hosts: web

  become: true

  vars_files: vars/main.yml

  tasks:

    - name: "Juste pour le debug"

      debug: var=ansible_os_family

    - fail:

      when: 1 == 1
```



```
...
...Copier
```

Vous remarquerez l'utilisation d'un nouveau module nommé fail ([Documentation ici](#)), ainsi qu'une nouvelle instruction nommée **when**. En effet, il s'agit d'une astuce que j'utilise afin de tester une tâche rapidement sans avoir à exécuter les tâches suivantes. Premièrement, nous avons le module fail qui nous permettra de quitter notre playbook en simulant une erreur. Deuxièmement il faut forcer la sortie de notre Playbook, pour cela nous devons créer une condition qui est toujours vraie. Pour ce faire, nous utilisons l'instruction **when** qui permet **d'ignorer ou de forcer l'exécution d'une tâche** particulière sur un hôte particulier, dans notre cas comme 1 est toujours égal à 1 la condition sera toujours vraie ☐.

Voici les différents opérateurs de comparaison qu'il est possible d'utiliser sur vos playbooks Ansible :

Opérateur	Description	Exemple	Résultat
==	Compare deux valeurs et vérifie leur égalité	x==4	La condition est bonne si x est égale à 4
<	Vérifie qu'une variable est strictement inférieure à une valeur	x<4	La condition est bonne si x est strictement inférieure à 4
<=	Vérifie qu'une variable est inférieure ou égale à une valeur	x<=4	La condition est bonne si x est inférieure ou égale à 4
>	Vérifie qu'une variable est strictement supérieure à une valeur	x>4	La condition est bonne si x est strictement supérieure à 4
>=	Vérifie qu'une variable est supérieure ou égal à une valeur	x>=4	La condition est bonne si x est supérieure ou égale à 4
!=	Vérifie qu'une variable est différente à une valeur	x != 4	La condition est bonne si x est différent de 4
is	Vérifie qu'une variable représente le même objet	x is True	La condition est bonne si x est True
is not	Vérifie qu'une variable ne représente pas le même objet	x is not True	La condition est bonne si x n'est pas True

Exécutons notre playbook :

```
ansible-playbook playbook.ymlCopier
```

Résultat :

```
PLAY [web] *****
```

```

TASK [Gathering Facts] ***

ok: [slave-1]


TASK [Juste pour le debug] *****

ok: [slave-1] => {
    "ansible_os_family": "RedHat"
}


TASK [fail] *****

fatal: [slave-1]: FAILED! => {"changed": false, "msg": "Failed as requested from task"}


PLAY RECAP *****

slave-1 : ok=2    changed=0    unreachable=0    failed=1

```

Vous avez aussi une autre manière d'utiliser vos Facts dans vos playbooks, comme suit :

```

- name: "Juste pour le debug"

  debug: var=ansible_facts['os_family']

```

Utilisation des conditions dans la partie Web

Sans plus attendre, utilisons une nouvelle fois l'instruction **when** avec le module yum si la distribution fait partie de la famille **RedHat** ou avec le module apt si elle fait partie de la famille **Debian**. Nous aurons ainsi les tâches suivantes :

```

---
```

```
# WEB SERVER

- hosts: web

  become: true

  vars_files: vars/main.yml

  tasks:

    - name: install apache and php last version for Debian os family

      apt:

        name: ['apache2', 'php', 'php-mysql']

        state: present

        update_cache: yes

        when: ansible_facts['os_family'] == "Debian"

    - name: install apache and php last version for RedHat os family

      yum:

        name: ['httpd', 'php', 'php-mysqlnd']

        state: present

        update_cache: yes

        when: ansible_facts['os_family'] == "RedHat"
```

```
- fail:

  when: 1 == 1Copier
```

Maintenant, il faut adapter les tâches suivantes de la partie web pour qu'elles soient compatibles avec les machines de la famille RedHat. Ce qui nous donnera les tâches suivantes :

```
- name: Give writable mode to http folder

  file:

    path: /var/www/html

    state: directory

    mode: '0755'

- name: remove default index.html

  file:

    path: /var/www/html/index.html

    state: absent

- name: upload web app source

  copy:

    src: app/

    dest: /var/www/html/

- name: deploy php database config
```

```
template:

  src: "db-config.php.j2"

  dest: "/var/www/html/db-config.php"

- name: ensure apache service is start (Debian os family)

  service:

    name: apache2

    state: started

    enabled: yes

  when: ansible_facts['os_family'] == "Debian"

- name: ensure apache service is start (RedHat os family)

  service:

    name: httpd

    state: started

    enabled: yes

  when: ansible_facts['os_family'] == "RedHat"

- name: enable connection with remote database (RedHat os family)

  shell: setsebool -P httpd_can_network_connect_db 1

  when: ansible_facts['os_family'] == "RedHat" Copier
```

Pour le moment, si vous exécutez votre Playbook et que vous visitez la page d'accueil http://IP_SERVEUR_WEB, vous obtiendrez alors l'erreur SQL suivante :

Mon serveur apache ansible ! [Accueil](#) [Articles](#)

Articles

Nouveau article

Titre *

Nom de l'auteur *

Contenu *

Liste d'articles

ERREUR :SQLSTATE[HY000] [2002] Per

Cette erreur SQL est tout à fait normale car nous n'avons pas encore configuré les tâches de notre partie base de données.

Utilisation des conditions dans la partie base de données

Lors de la refonte du playbook, la refonte de la partie base de données a été pour moi un peu plus complexe. Pour ne rien vous cacher, cette section m'a pris plus temps que prévu à debug ☐. Cependant, d'un autre côté elle nous permettra de voir un maximum de concepts Ansible.

Comme pour la partie Web, les packages diffèrent selon la famille de distribution utilisée. Néanmoins, pour la famille RedHat j'ai choisi d'ajouter d'abord le repository mysql et d'installer ensuite le package mysql sous sa version 5.7. Ce qui nous donne le résultat suivant :

```
- name: install mysql repo (Fedora)

  yum:

    name: "http://repo.mysql.com/mysql80-community-release-fc{{
ansible_facts['distribution_major_version'] }}.rpm"

    state: present

    update_cache: yes

    when: ansible_facts['distribution'] == "Fedora"

- name: install mysql repo (CENTOS or RedHat)

  yum:

    name: "http://repo.mysql.com/mysql80-community-release-el{{
ansible_facts['distribution_major_version'] }}.rpm"

    state: present

    update_cache: yes

    when: ansible_facts['os_family'] == "RedHat" and
ansible_facts['distribution'] != "Fedora"
```

```

- name: install mysql package (RedHat os family)

  yum:

    name: mysql-community-server

    state: present

    disablerepo: mysql80-community

    enablerepo: mysql57-community

    when: ansible_facts['os_family'] == "RedHat"

- name: install PyMySQL from pip (RedHat os family)

  pip:

    name: PyMySQL # for mysql_db and mysql_user modules

    when: ansible_facts['os_family'] == "RedHat"

```

Comme vous pouvez l'apercevoir, les repositories mysql se distinguent selon la distribution utilisée. Les packages CENTOS et RHEL utilisent le même repository ce qui n'est pas le cas pour Fedora. Pour gérer cette condition, on peut utiliser l'opérateur logique **and** comme dans la ligne suivante :

```

when: ansible_facts['os_family'] == "RedHat" and
ansible_facts['distribution'] != "Fedora"

```

Je vérifie donc si la machine appartient à la famille Redhat et si la distribution est différente à Fedora. Voici une liste des opérateurs logiques possibles sur Ansible :

Opérateur	Signification	Description
or	OU	Vrai si au moins une des comparaisons est vraie

Opérateur	Signification	Description
and	ET	Vrai si toutes les comparaisons sont vraies
not	NON	Retourne faux si la comparaison est vraie et vraie si la comparaison est fausse

Ensuite, on s'assure que les services mysql sont bien démarrés :

```
- name: ensure mysql service is start (Debian os family)

  service:

    name: mysql

    state: started

    enabled: yes

  when: ansible_facts['os_family'] == "Debian"

- name: ensure mysqld service is start (RedHat os family)

  service:

    name: mysqld

    state: started

    enabled: yes

  when: ansible_facts['os_family'] == "RedHat"
Copier
```

Enregistrer les sorties dans une variable

Une autre utilisation importante des variables consiste à exécuter un module et d'**enregistrer le résultat en tant que variable**. Par exemple on peut exécuter une tâche avec le module [shell](#) ou le module [command](#) et enregistrer la valeur de retour dans une variable pour l'utiliser dans des tâches ultérieures. Ce type de variable est nommée en anglais "registered variable" ou en français "variable enregistrée".

Pourquoi, je vous parle de ce type de variable ? Tout simplement car à un moment donné, il faudra récupérer le mot de passe temporaire généré à la fin de l'installation du package mysql. Cette action est possible avec l'utilisation de l'instruction **register** :

```
- name: Register temporary password (RedHat os family)

  shell: "grep 'temporary password' /var/log/mysqld.log | awk
'{{print $(NF)}}'"

  register: password_tmp

  when: ansible_facts['os_family'] == "RedHat" Copier
```

Dans notre exemple la sortie de la commande `grep 'temporary password' /var/log/mysqld.log | awk '{{print $(NF)}}'` sera enregistrée dans une variable temporaire nommée **password_tmp**.

Cette variable enregistrée est de type **List** et stock donc plusieurs données dont la sortie standard de la commande dans la clé **stdout** et le code retour de la commande dans la clé **rc**.

Voici le résultat lors du debug de cette variable enregistrée :

```
- name: Register temporary password (RedHat os family)

  shell: "grep 'temporary password' /var/log/mysqld.log | awk
'{{print $(NF)}}'"

  register: password_tmp

  when: ansible_facts['os_family'] == "RedHat"

- debug: var=password_tmp

- fail:

  when: 1 == 1 Copier
```

Résultat :

```
TASK [debug] *****

ok: [slave-2] => {

    "password_tmp": {

        "changed": true,

        "cmd": "grep 'temporary password' /var/log/mysqld.log | awk
'{print $(NF)}'",

        "delta": "0:00:00.007624",

        "end": "2020-02-05 18:35:21.521570",

        "failed": false,

        "rc": 0,

        "start": "2020-02-05 18:35:21.513946",

        "stderr": "",

        "stderr_lines": [],

        "stdout": "d.to29&Rque&",

        "stdout_lines": [

            "d.to29&Rque&"

        ]

    }

}
```

Surcharger la valeur d'une variable

Lors de mes tests, j'ai remarqué que le mot de passe du compte root mysql par défaut pouvait être utilisé sur des machines de la famille Debian, ce qui n'est pas le cas des machines de type RedHat où il faut utiliser le mot de passe temporaire récupérée précédemment. Le mot de passe peut donc varier d'une famille de distribution à l'autre.

Ici l'astuce consiste à stocker le mot de passe root (chaîne de caractères vide) par défaut dans une variable nommée `default_root_password`, et de la **surcharger avec la valeur de la variable `password_tmp`** quand il s'agit d'une machine la famille Redhat. Pour surcharger la valeur par défaut d'une variable, il faut utiliser l'instruction `set_fact` comme suit :

```
- name: Set default root user password (RedHat os family)

  set_fact:

    default_root_password: '{{ password_tmp.stdout }}'

  when: ansible_facts['os_family'] == "RedHat" Copier
```

Forcer l'ignorance d'une erreur

Généralement, les playbooks cessent d'exécuter d'autres étapes sur un hôte dont la tâche échoue. Cependant, parfois vous voulez continuer vos tâches même après erreur d'où l'utilisation de l'instruction `ignore_errors` avec la valeur `yes`.

Dans notre playbook, on utilisera le module shell avec la commande `grep` afin de vérifier si notre fichier de configuration mysql est configuré comme il se doit. Toutefois Ansible considère qu'il doit quitter un playbook à chaque code retour différent de 0 et le code retour de la commande `grep` est 0 si les lignes sélectionnées sont trouvées et 1 s'ils ne sont pas trouvés. Dans notre cas, on souhaite continuer l'exécution de notre playbook même s'il ne trouve pas le mot recherché dans le fichier de configuration mysql. Ce qui nous donne le résultat suivant :

```
- name: check if mysql config is correct (RedHat os only)

  shell: 'grep "^bind-address" /etc/my.cnf'

  register: test_grep

  when: ansible_facts['os_family'] == "RedHat"
```

```

ignore_errors: yes # dont exit if it doesn't found something

- name: change mysql config (RedHat os only)

  blockinfile:

    path: /etc/my.cnf

    insertafter: EOF

    block: |

        default_authentication_plugin=mysql_native_password

        bind-address=0.0.0.0

        default_password_lifetime=0

        validate_password_policy=LOW

        validate_password_length=6

        validate_password_number_count=0

    when: ansible_facts['os_family'] == "RedHat" and test_grep.rc != 0

    notify: Restart mysqldCopier

```

La suite de notre playbook consiste à changer le mot passe root mysql et d'y implémenter l'architecture de notre base de données, comme suit :

```

- name: Change root SQL password and GRANT root privileges (RedHat
os family)

    command: "mysql --user=root --password={{ default_root_password }}
--connect-expired-password --execute=\"ALTER USER
'root'@'localhost' IDENTIFIED BY '{{ root_password }}'; grant all
privileges on *.* to 'root'@'localhost' with grant option;\""

    ignore_errors: yes # ignore errors because we only change mysql
root password once

```

```
when: ansible_facts['os_family'] == "RedHat"

- name: Create MySQL client config (Debian os family)

  copy:

    dest: "/root/.my.cnf"

    content: |

      [client]

      user=root

      password={{ root_password }}

    mode: 0400

when: ansible_facts['os_family'] == "Debian"

- name: upload sql table config

  template:

    src: "table.sql.j2"

    dest: "/tmp/table.sql"

- name: add sql table to database

  mysql_db:

    name: "{{ mysql_dbname }}"
```

```
state: present

login_user: root

login_password: '{{ root_password }}'

state: import

target: /tmp/table.sql


- name: "Create {{ mysql_user }} with all {{ mysql_dbname }}
privileges (Debian os family)"

mysql_user:

    name: "{{ mysql_user }}"

    password: "{{ mysql_password }}"

    priv: "{{ mysql_dbname }}.*:ALL"

    host: "{{ webserver_host }}"

    state: present

    login_user: root

    login_password: '{{ root_password }}'

    login_unix_socket: /var/run/mysqld/mysqld.sock

when: ansible_facts['os_family'] == "Debian"

notify: Restart mysql
```

```
- name: "Create {{ mysql_user }} with all {{ mysql_dbname }}
privileges (RedHat os family)"

mysql_user:

    name: "{{ mysql_user }}"

    password: "{{ mysql_password }}"

    priv: "{{ mysql_dbname }}.*:ALL"

    host: "{{ webserver_host }}"

    state: present

    login_user: root

    login_password: '{{ root_password }}'

    login_unix_socket: /var/lib/mysql/mysql.sock

when: ansible_facts['os_family'] == "RedHat"

notify: Restart mysqldCopier
```

Les Boucles

Pour économiser des lignes d'instructions, **des tâches répétées peuvent être écrites en raccourci grâce aux boucles**. Dans notre exemple nous utiliserons les boucles afin d'autoriser l'utilisateur du playbook à télécharger des extras packages.

Commençons d'abord par créer deux variables de type **List**, une variable pour les extras packages de la famille Debian et une autre pour la famille RedHat. Ces variables seront déclarées dans notre fichier `vars/main.yml` comme suit :

```
---

# ...

# ...
```



```
extra_packages_debian: ['php-curl', 'php-gd', 'php-mbstring']
extra_packages_redhat: ['php-xml', 'php-gd', 'php-mbstring']Copier
```

Une fois déclarées, il suffit de les intégrer dans un nouveau module yum et apt, comme suit :

```
- name: install extra packages (Debian os family)

  apt:

    name: "{{ extra_packages_debian }}"

    state: present

    when: ansible_facts['os_family'] == "Debian"

- name: install extra packages (RedHat os family)

  yum:

    name: "{{ extra_packages_redhat }}"

    state: present

    when: ansible_facts['os_family'] == "RedHat"Copier
```

Cependant, certains paramètres de modules ne prennent en compte que des chaînes de caractères. C'est le cas par exemple du module `user` ([Documentation ici](#)) où le paramètre `name` est de type de `string`. Si vous souhaitez par exemple vérifier, créer, modifier ou supprimer plusieurs comptes, il ne sera pas judicieux d'écrire plusieurs fois la même tâche pour chaque compte différent, dans ce cas vous devez penser directement aux boucles avec l'instruction `with_items` qui s'utilisera comme suit :

```
- name: add several users

  user:

    name: "{{ item }}"
```

```
state: present

groups: "dev"

with_items:

  - user1

  - user2Copier
```

Notez que les types d'éléments que vous parcourez avec `with_items` ne doivent pas être que de type `List`, vous pouvez aussi référencer des clés-valeurs (type Map) l'utilisant comme suit :

```
- name: add several users

user:

  name: "{{ item.name }}"

  state: present

  groups: "{{ item.groups }}"

  with_items:

    - { name: 'test', groups: 'test' }

    - { name: 'admin', groups: 'prod' }Copier
```

Conclusion et Test

Notre playbook est désormais opérationnel sur les machines cibles de la famille RedHat et Debian. Ce changement nous a permis d'étudier un maximum de concepts sur Ansible. Vous pouvez télécharger le nouveau projet complet en [cliquant ici](#).

Une fois votre playbook lancé, visitez la page suivante http://IP_SERVEUR_WEB, et vous obtiendrez la page d'accueil suivante :

Articles

Nouveau article

Titre *

Nom de l'auteur *

Contenu *

Le lorem ipsum est, en imprimerie, une
remplacer le faux-texte dès qu'il est prè

Envoyer

Liste d'articles

Pour tester la connexion à votre base de données, appuyez sur le bouton "Envoyer" pour valider le formulaire et rajouter un article à la base de données, ce qui nous donnera le résultat suivant :

Liste d'articles

Test

24/01/20 19:26

Le lorem ipsum est, en imprimerie, une suite de mots sans signification utilisée à titre provisoire venant remplacer le faux-texte dès qu'il est prêt ou que la mise en page est achevée. Généralement on utilise du Lorem Ipsum.

— *hatim*

Dans le prochain chapitre, nous verrons comment mieux organiser notre playbook grâce à la notion de rôles.

Les rôles sur Ansible

Dans ce chapitre, nous verrons comment améliorer notre ancien Playbook grâce aux systèmes de rôle proposés par Ansible.

Introduction

Dans le chapitre précédent, nous avons réussi à adapter notre playbook pour les systèmes d'exploitation de type Redhat et Debian. Je vous avais aussi précisé que nous allions **améliorer la structure de notre Playbook grâce aux systèmes de rôle** proposés par Ansible.

C'est quoi un rôle et pourquoi ?

Les rôles sont une caractéristique robuste d'Ansible qui **facilitent la réutilisation**, favorisent davantage la modularisation de votre configuration et simplifient l'écriture de vos Playbooks complexes en le divisant logiquement en **composants réutilisables** et en profiter par la même occasion pour le rendre plus **facile à lire**.

Les rôles fournissent un cadre pour une réutilisation indépendante de variables, tâches, fichiers, modèles et modules. Pour faire simple, vous pouvez comparer les rôles Ansible aux bibliothèques dans les langages de programmation. En effet, en programmation les bibliothèques contiennent leurs propres variables et fonctions que vous pouvez réutiliser pour vos différents projets de programmation. C'est le cas aussi pour les rôles, qui possèdent leurs propres tâches, variables, handlers etc ... que vous pouvez importer et utiliser dans vos différents playbooks.

Chaque rôle est essentiellement limité à une fonctionnalité particulière (Ex: installation d'apache) qui peut être **utilisée indépendamment** et il n'existe aucun moyen d'exécuter directement les rôles car ils ne peuvent être utilisés que depuis vos playbooks, puisque les rôles n'ont pas de paramètre explicite pour l'hôte auquel le rôle s'appliquera, cette tâche est gérée au niveau supérieur par vos playbooks qui maintiennent les hôtes de votre fichier d'inventaire vers les rôles qui doivent être appliqués à ces hôtes.

L'anatomie d'un rôle Ansible

La **structure de répertoires des rôles** est essentielle pour créer un nouveau rôle, nous utiliserons la commande `ansible-galaxy` pour **créer le squelette de notre rôle automatiquement** :

```
ansible-galaxy init [ROLE NAME]
```

Soit dans notre exemple la création d'un rôle nommé **test-role** :

```
ansible-galaxy init test-role
```

Résultat :

```
- Role test-role was created successfully
```

Voici à quoi ressemble la structure du répertoire de notre nouveau rôle nommé **test-role** :

```
test-role
```

```
|— defaults
|   |_____| main.yml
|— files
|— handlers
|   |_____| main.yml
|— meta
|   |_____| main.yml
|— tasks
|   |_____| main.yml
|— templates
|— tests
|   |_____| inventory
|   |_____| test.yml
|— vars
|   |_____| main.yml
|— README.md
```

Les rôles s'attendent à ce que les fichiers se trouvent dans certains noms de répertoire. Les rôles doivent inclure au moins un de ces répertoires, mais il est parfaitement correct d'exclure ceux qui ne sont pas utilisés. Lorsqu'il est utilisé, chaque répertoire doit contenir un fichier (sauf pour le dossier *files* et *templates*).

Voici ci-dessous **les caractéristiques de l'arborescence d'un rôle** :

- *tasks*: contient la liste principale des tâches à exécuter par le rôle.
- *handlers*: contient les handlers, qui peuvent être utilisés par ce rôle ou même en dehors de ce rôle.

- *defaults*: variables par défaut pour le rôle.
- *vars*: d'autres variables pour le rôle.
- *files*: contient des fichiers qui peuvent être déployés via ce rôle.
- *templates*: contient des modèles (jinja2) qui peuvent être déployés via ce rôle.
- *meta*: définit certaines métadonnées pour ce rôle.
- : inclut une description générale du fonctionnement de votre rôle.
- *test*: contient notre playbook (on peut cependant déposer notre playbook à la racine du projet ou dans un dossier sous un nom différent).

Notre but dans cet article est de **répartir les tâches de notre ancien playbook LAMP sous forme de rôles** en utilisant l'arborescence ci-dessus. Nous aurons ainsi un rôle pour l'installation et la configuration de la partie web et un second pour notre base de données.

Création de nos rôles

Commencez par télécharger notre ancien projet complet en [cliquant ici](#).

La partie web

Dans la racine du projet créez un dossier nommé *roles* qui contiendra pour l'instant notre rôle **web**. Dans ce même sous dossier rajoutons l'arborescence d'un rôle avec la commande vue précédemment `ansible-galaxy` :

```
mkdir roles

cd roles

ansible-galaxy init web

cd ../../Copier
```

Nous supprimerons ensuite le dossier *tests*, *defaults*, *vars* et *handlers* qui nous ne serviront à rien pour le moment:

```
rm -rf roles/web/{tests,defaults,handlers,vars}Copier
```

La première étape repose sur le déplacement de nos fichiers jinja2 et de nos fichiers sources situés respectivement dans le dossier *templates* et *files* vers notre nouveau rôle **web**. Pour ce faire, placez-vous donc sur la racine de votre projet et lancez les commandes suivantes :

```
mv templates/db-config.php.j2 roles/web/templates

mv files/app roles/web/files

rm -rf filesCopier
```

L'étape suivante consiste à déplacer les tâches de la partie web directement dans le fichier de tâches de notre rôle **web** situé dans `roles/web/tasks` qui ressemblera après changement à ceci :

```
---

# tasks file for web

- name: install apache and php last version for (Debian os family)

  apt:

    name: ['apache2', 'php', 'php-mysql']

    state: present

    update_cache: yes

    when: ansible_facts['os_family'] == "Debian"

- name: install extra packages (Debian os family)

  apt:

    name: "{{ extra_packages_debian }}"

    state: present

    when: ansible_facts['os_family'] == "Debian" and
extra_packages_debian is defined

- name: install apache and php last version for (RedHat os family)
```



```
yum:
```

```
  name: ['httpd', 'php', 'php-mysqlnd']
```

```
  state: present
```

```
  update_cache: yes
```

```
when: ansible_facts['os_family'] == "RedHat"
```

```
- name: install extra packages (RedHat os family)
```

```
yum:
```

```
  name: "{{ extra_packages_redhat }}"
```

```
  state: present
```

```
  when: ansible_facts['os_family'] == "RedHat" and  
extra_packages_redhat is defined
```

```
- name: Give writable mode to http folder
```

```
file:
```

```
  path: /var/www/html
```

```
  state: directory
```

```
  mode: '0755'
```

```
- name: remove default index.html
```

```
file:
```

```
    path: /var/www/html/index.html

    state: absent

- name: upload web app source

  copy:

    src: app/

    dest: /var/www/html/

- name: deploy php database config

  template:

    src: "db-config.php.j2"

    dest: "/var/www/html/db-config.php"

- name: ensure apache service is start (Debian os family)

  service:

    name: apache2

    state: started

    enabled: yes

    when: ansible_facts['os_family'] == "Debian"

- name: ensure apache service is start (RedHat os family)
```

```
service:

  name: httpd

  state: started

  enabled: yes

  when: ansible_facts['os_family'] == "RedHat"

- name: enable connection with remote database (RedHat os family)

  shell: setsebool -P httpd_can_network_connect_db 1

  when: ansible_facts['os_family'] == "RedHat" Copier
```

Pour finir, il ne faut pas oublier d'importer notre rôle dans notre playbook comme suit :

```
---

# WEB SERVER

- hosts: web

  become: true

  vars_files: vars/main.yml

  roles:

    - web
```

```
# DATABASE SERVER

# suite tâches databaseCopier
```

Partie base de données.

Nous referons la même manipulation pour notre nouveau rôle **database**, mais cette fois-ci nous garderons le dossier *vars* et le dossier *handlers*, car nous les réutiliserons plus tard :

```
mkdir roles

cd roles

ansible-galaxy init database

rm -rf database/{tests,defaults}

cd ../Copier
```

Commençons par déplacer le dernier fichier jinja2 de notre dossier *templates* vers notre nouveau rôle **database** :

```
mv templates/table.sql.j2 roles/database/templates

rm -rf templatesCopier
```

La prochaine étape comprend le déplacement des handlers utilisés dans les tâches de la partie base de données dans le fichier `handlers.yml` de notre base de données, qui ressemblera à ceci :

```
---

# handlers file for database

- name: Restart mysql # Debian os family

  service:

    name: mysql

    state: restarted
```

```
- name: Restart mysqld # RedHat os family

service:

    name: mysqld

    state: restartedCopier
```

Si vous ouvrez votre fichier playbook , vous pouvez apercevoir la variable suivante :

```
vars:

    default_root_password: ""Copier
```

La variable **default_root_password** n'est utilisable que par nos tâches de la partie de la base de données, elle permet en effet de récupérer automatiquement le mot de passe root mysql. L'utilisateur du playbook n'a pas à modifier cette variable, on peut donc la déplacer vers le fichier , comme suit :

```
---

# defaults file for database

default_root_password: ""Copier
```

Maintenant, il ne nous reste qu'à déplacer les tâches de la partie de notre base de données vers les tâches du rôle **database** dans le fichier :

```
---

# tasks file for database

- name: install mysql (Debian os family)

  apt:

    name:
```

```
- mysql-server

- python-mysqldb # for mysql_db and mysql_user modules

state: present

update_cache: yes

when: ansible_facts['os_family'] == "Debian"

- name: install mysql repo (Fedora)

  yum:

    name: "http://repo.mysql.com/mysql80-community-release-fc{{
ansible_facts['distribution_major_version'] }}.rpm"

    state: present

    update_cache: yes

    when: ansible_facts['distribution'] == "Fedora"

- name: install mysql repo (CENTOS or RedHat)

  yum:

    name: "http://repo.mysql.com/mysql80-community-release-el{{
ansible_facts['distribution_major_version'] }}.rpm"

    state: present

    update_cache: yes

    when: ansible_facts['os_family'] == "RedHat" and
ansible_facts['distribution'] != "Fedora"
```

```
- name: install mysql package (RedHat os family)

yum:

    name: mysql-community-server

    state: present

    disablerepo: mysql80-community

    enablerepo: mysql57-community

when: ansible_facts['os_family'] == "RedHat"


- name: install PyMySQL from pip (RedHat os family)

pip:

    name: PyMySQL # for mysql_db and mysql_user modules

when: ansible_facts['os_family'] == "RedHat"


- name: ensure mysql service is start (Debian os family)

service:

    name: mysql

    state: started

    enabled: yes

when: ansible_facts['os_family'] == "Debian"
```

```
- name: ensure mysqld service is start (RedHat os family)

service:

    name: mysqld

    state: started

    enabled: yes

when: ansible_facts['os_family'] == "RedHat"


- name: Allow external MySQL connections (1/2) (Debian os family)

lineinfile:

    path: /etc/mysql/mysql.conf.d/mysqld.cnf

    regexp: '^skip-external-locking'

    line: "# skip-external-locking"

notify: Restart mysql

when: ansible_facts['os_family'] == "Debian"


- name: Allow external MySQL connections (2/2) (Debian os family)

lineinfile:

    path: /etc/mysql/mysql.conf.d/mysqld.cnf

    regexp: '^bind-address'
```



```
    line: "# bind-address"

    notify: Restart mysql

    when: ansible_facts['os_family'] == "Debian"

- name: check if mysql config is correct (RedHat os only)

    shell: 'grep "^bind-address" /etc/my.cnf'

    register: test_grep

    when: ansible_facts['os_family'] == "RedHat"

    ignore_errors: yes # dont exit if it doesn't found something

- name: change mysql config (RedHat os only)

    blockinfile:

        path: /etc/my.cnf

        insertafter: EOF

        block: |

            default_authentication_plugin=mysql_native_password

            bind-address=0.0.0.0

            default_password_lifetime=0

            validate_password_policy=LOW

            validate_password_length=6
```

```

    validate_password_number_count=0

when: ansible_facts['os_family'] == "RedHat" and test_grep.rc != 0

notify: Restart mysqld

- name: Register temporary password (RedHat os family)

  shell: "grep 'temporary password' /var/log/mysqld.log | awk
'{{print $(NF)}}'"

  register: password_tmp

  when: ansible_facts['os_family'] == "RedHat"

- name: Set default root user password (RedHat os family)

  set_fact:

    default_root_password: '{{ password_tmp.stdout }}'

  when: ansible_facts['os_family'] == "RedHat"

- name: Change root SQL password and GRANT root privileges (RedHat
os family)

  command: "mysql --user=root --password={{ default_root_password }}
--connect-expired-password --execute=\"ALTER USER
'root'@'localhost' IDENTIFIED BY '{{ root_password }}'; grant all
privileges on *.* to 'root'@'localhost' with grant option;\""

  ignore_errors: yes # ignore errors because we only change mysql
root password once

  when: ansible_facts['os_family'] == "RedHat"

```

```
- name: Create MySQL client config (Debian os family)
```

```
  copy:
```

```
    dest: "/root/.my.cnf"
```

```
    content: |
```

```
      [client]
```

```
      user=root
```

```
      password={{ root_password }}
```

```
    mode: 0400
```

```
  when: ansible_facts['os_family'] == "Debian"
```

```
- name: upload sql table config
```

```
  template:
```

```
    src: "table.sql.j2"
```

```
    dest: "/tmp/table.sql"
```

```
- name: add sql table to database
```

```
  mysql_db:
```

```
    name: "{{ mysql_dbname }}"
```

```
    state: present
```

```
login_user: root

login_password: '{{ root_password }}'

state: import

target: /tmp/table.sql


- name: "Create {{ mysql_user }} with all {{ mysql_dbname }}
privileges (Debian os family)"

mysql_user:

    name: "{{ mysql_user }}"

    password: "{{ mysql_password }}"

    priv: "{{ mysql_dbname }}.*:ALL"

    host: "{{ webserver_host }}"

    state: present

    login_user: root

    login_password: '{{ root_password }}'

    login_unix_socket: /var/run/mysqld/mysqld.sock

when: ansible_facts['os_family'] == "Debian"

notify: Restart mysql


- name: "Create {{ mysql_user }} with all {{ mysql_dbname }}
privileges (RedHat os family)"
```

```
mysql_user:

  name: "{{ mysql_user }}"

  password: "{{ mysql_password }}"

  priv: "{{ mysql_dbname }}.*:ALL"

  host: "{{ webserver_host }}"

  state: present

  login_user: root

  login_password: '{{ root_password }}'

  login_unix_socket: /var/lib/mysql/mysql.sock

  when: ansible_facts['os_family'] == "RedHat"

  notify: Restart mysqldCopier
```

Enfin la dernière étape consiste à importer notre rôle **database** dans notre playbook, voici donc à quoi ressemblera la version finale de notre playbook :

```
---

# WEB SERVER

- hosts: web

  become: true

  vars_files: vars/main.yml

  roles:

    - web
```

```
# DATABASE SERVER

- hosts: db

  become: true

  vars_files: vars/main.yml


roles:

  - databaseCopier
```

Ansible Galaxy

C'est quoi ?

Ansible Galaxy est une plateforme Web où les utilisateurs peuvent **partager leurs rôles Ansible** et c'est également un outil en ligne de commande pour installer, créer et gérer des rôles.

Information

Nous avons déjà eu l'occasion d'utiliser précédemment la commande `ansible-galaxy init`.

Ansible Galaxy est un site gratuit qui vous permet de rechercher, télécharger et partager des rôles développés par la communauté. Le téléchargement de rôles depuis Galaxy est un excellent moyen de relancer vos projets d'automatisation.

Vous pouvez également utiliser le site pour partager les rôles que vous créez. En vous authentifiant auprès du site à l'aide de votre compte [GitHub](#). Vous pouvez importer des rôles et les rendre disponibles à la communauté Ansible. Les rôles importés deviennent disponibles dans l'index de recherche Galaxy et visibles sur le site, permettant aux utilisateurs de les découvrir et de les télécharger. Nous allons de ce fait, **partager un de nos deux rôles dans la plateforme Ansible Galaxy**.

Documentez vos rôles !

Avant toute chose, il faut commencer par documenter vos rôles. La commande `ansible-galaxy init` lancée antérieurement, crée également à la racine de vos rôles un fichier nommé `README.md` où nous devons décrire dedans l'utilisation de notre rôle. Par exemple dans mon cas, voici à quoi ressemble ma documentation pour le rôle **web** :

```
web

=====

install apache and php with a test web app.


Requirements
-----

Debian os family or RedHat OS family


Role Variables
-----

| name                | type   | description |
| mandatory           |        |              |
| -----|-----|-----|
| `mysql_user`        | string | Mysql user that will be used in |
| the php app         | yes   |              |
```

<code>`mysql_password`</code> used in the php app	string	Mysql password that will be yes
<code>`mysql_dbname`</code> in the php app	string	Database name that will be used yes
<code>`db_host`</code> be used in the php app	string	Database ip/host that that will yes
<code>`extra_packages_debian`</code> be downloaded	list	extra Debian packages that will no
<code>`extra_packages_redhat`</code> be downloaded	list	extra RedHat packages that will no

Dependencies

n/a

Example Playbook

```
```yaml
```

```
- hosts: web
```

```
 become: yes
```

```
 vars: vars/main.yml
```

```
 - mysql_user: "admin"
```



```

- mysql_password: "Test_34535$"

- mysql_dbname: "blog"

- db_host: "192.168.0.22"

- extra_packages_debian: ['php-curl', 'php-gd', 'php-mbstring']

- extra_packages_redhat: ['php-xml', 'php-gd', 'php-mbstring']

roles:

 - web
...

License

BSD

Author Information

https://devopssec.fr/Copier

```

Lorsque vous recherchez ou importez un rôle depuis Ansible Galaxy, le processus de recherche ou d'importation se base sur les métadonnées trouvées dans le fichier `meta/main.yml` du rôle. Ce fichier peut contenir les informations suivantes :

```
galaxy_info:

 role_name:

 author:

 description:

 company:

 license:

 min_ansible_version:

 platforms:

 galaxy_tags:

dependencies: []
```

Ci-dessous une explication de certains paramètres :

- **role\_name** : (Optionnel) nom du rôle.
- **galaxy\_tags** : (Optionnel) fournir des balises qui sont unique qui permettent de classer votre rôle (une par ligne).
- **plateformes** : (Obligatoire) liste de plates-formes valides avec une liste de versions valides. (La liste des plate-formes est disponibles [ici](#)).
- **dependencies** : (Optionnel) liste de vos dépendances de rôle ici, ces dépendances seront automatiquement installées (une par ligne).

Voici à quoi ressemble par exemple le fichier `galaxy.yml` de notre rôle **web** :

```
galaxy_info:

 author: AJDAINI Hatim

 description: install apache and php with a test web app

 company: https://devopssec.fr

 license: MIT
```

```
min_ansible_version: 1.6
```

```
platforms:
```

```
- name: Fedora
```

```
 versions:
```

```
 - 28
```

```
 - 29
```

```
 - 30
```

```
 - 31
```

```
- name: Ubuntu
```

```
 versions:
```

```
 - all
```

```
- name: Debian
```

```
 versions:
```

```
 - all
```

```
- name: EL
```

```
 versions:
```

```
 - 7
```

```
galaxy_tags:
```

- web
- apache
- php

dependencies: []Copier

## Publier votre rôle sur Ansible Galaxy

### **Attention**

J'ai créé temporairement ce repository Github qui sera par la suite supprimé à la publication de cet article.

Vous devez au préalable posséder un compte Github pour publier vos rôles dans la plateforme Ansible Galaxy. La première étape consiste à créer un repository Github où vous déposerez directement dedans l'arborescence de votre rôle :

hajdaini / web

<> Code

Issues 0

Pull requests 0

Actions

Projects 0

Wiki

Security

my web (apache + php) ansible role

Manage topics

2 commits

1 branch

0 packages

0 releases

Branch: master ▾ New pull request Create new file

hajdaini Add files via upload

files/app	Add files via upload
meta	Add files via upload
tasks	Add files via upload
templates	Add files via upload
vars	Add files via upload
README.md	Add files via upload

README.md

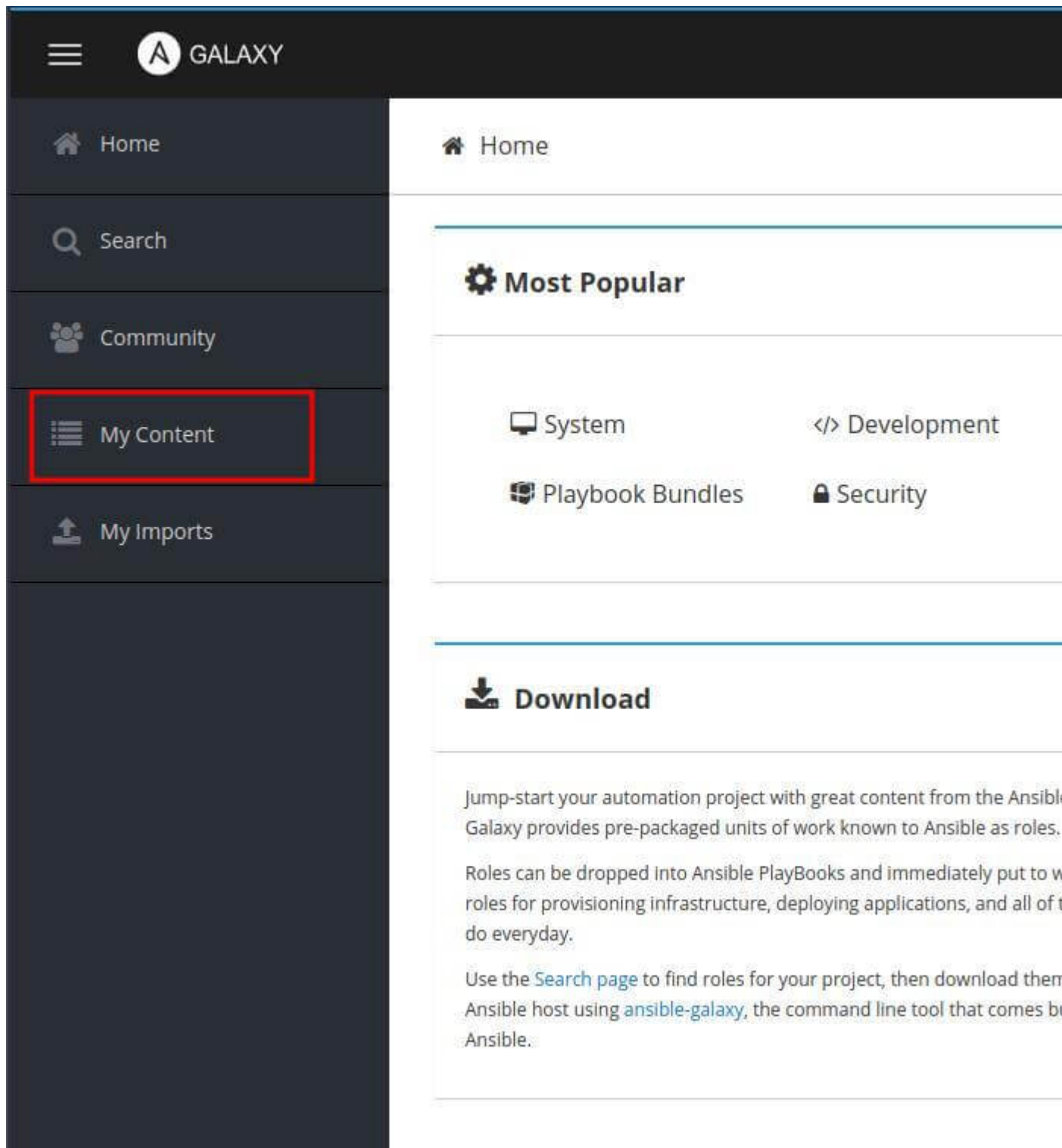
# web

install apache and php with a test web app.

## Requirements

Debian os family or RedHat OS family

Connectez-vous ensuite avec votre compte Github sur le site [Ansible Galaxy](#) et cliquez sur le bouton "My content" sur le menu à gauche :



Par la suite, cliquez sur le bouton "+ Add Content" :

## My Content

Name ▾  | Name ▾ 

  **hajdaini** Hatim


Name ▾  | Name ▾ 


10 ^


per page

Add content

Après cela, cliquez sur le bouton "Import Role from GitHub" :

Add Content 

 Import Role from GitHub

 Upload New Collection

Cancel

Choisissez ensuite le rôle adéquat :

Add repositories to hajdaini

hajdaini

Filter by name...

☐

hajdaini/Monitoring

☐

hajdaini/pythonchallenge.com

☐

hajdaini/Reminder\_aide\_memoire

☐

hajdaini/Security

☐

hajdaini/System

☒

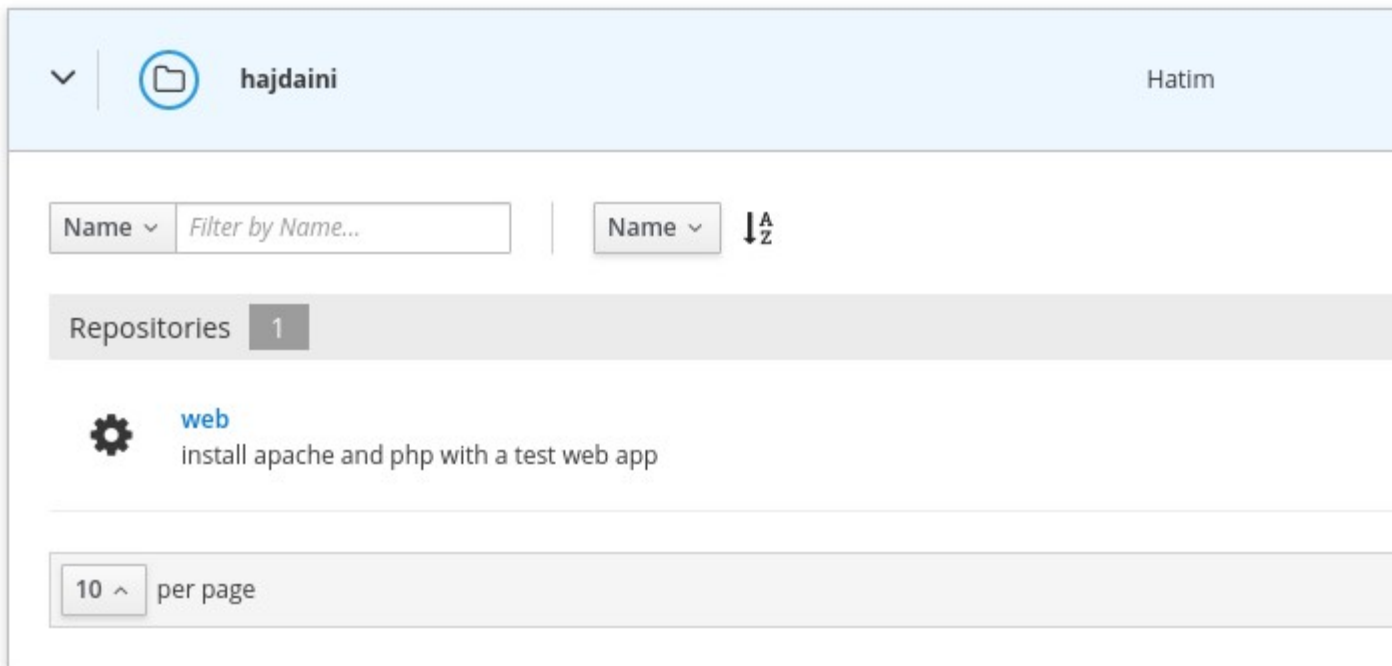
hajdaini/web

Back

OK

Cancel





## Récupérer votre rôle depuis Ansible Galaxy

N'importe qui peut dorénavant télécharger votre rôle ! Vous pouvez soit lancer une recherche directement depuis la plateforme Ansible Galaxy, soit directement depuis la ligne de commande `ansible-galaxy`. Pour notre exemple, nous utiliserons la commande avec l'option `-author` afin de mieux filtrer nos recherches :

```
ansible-galaxy search web --author hajdainiCopier
```

### Résultat :

```
Found 1 roles matching your search:

Name Description

hajdaini.web install apache and php with a test web app
```

Vous pouvez également **récupérer des informations supplémentaires sur votre rôle** en tapant la commande suivante :

```
ansible-galaxy info hajdaini.webCopier
```

## Résultat :

```
Role: hajdaini.web

description: install apache and php with a test web app

active: True

commit: 8a1302a30b4657f3a290287762c0dd1a7fff4b17

commit_message: Update main.yml

commit_url:
https://api.github.com/repos/hajdaini/web/git/commits/8a1302a30b4657f3a290287762c0dd1a7fff4b17

company: https://devopssec.fr

created: 2020-02-11T12:30:22.362817Z

download_count: 1

forks_count: 0

github_branch: master

github_repo: web

github_server: https://github.com

github_user: hajdaini

id: 46451

imported: 2020-02-11T07:43:39.834522-05:00

is_valid: True

issue_tracker_url: https://github.com/hajdaini/web/issues

license: MIT

min_ansible_version: 1.6
```

```
modified: 2020-02-11T12:43:39.849377Z
```

Vous pouvez ensuite **installer le rôle** depuis la commande suivante :

```
ansible-galaxy install hajdaini.webCopier
```

Vous pouvez également définir un fichier un fichier nommé , où vous spécifiez les différents rôles à télécharger :

```

roles:

 - src: https://github.com/hajdaini/web

 version: master

 name: web

 - src: https://github.com/hajdaini/database

 version: master

 name: databaseCopier

ansible-galaxy install -r requirements.yml -p
[CHEMIN_DE_DESTINATION]Copier
```

Vous trouverez plus d'informations sur la [documentation officielle](#).

## Conclusion

---

Grâce aux systèmes de rôles, nous nous approchons de plus en plus vers un playbook de plus en plus modulable. Vous pouvez télécharger le projet complet en [cliquant ici](#).

Le prochain chapitre se concentrera plus sur l'aspect sécurité de notre playbook LAMP. Je vous donne donc rendez-vous au prochain chapitre, où on étudiera Ansible Vault.

# Les Vaults sur Ansible

Dans ce chapitre, nous étudierons les Vaults (coffre-fort) sur Ansible afin de sécuriser le stockage des données sensibles.

## Introduction

---

Dans le chapitre précédent, nous avons abouti à un Playbook modulable en utilisant le système de rôle proposé par Ansible. Néanmoins, on peut davantage améliorer notre playbook en utilisant le système de Vaults.

### C'est quoi Ansible Vault ?

Ansible Vault (coffre-fort en français) est une fonctionnalité d'Ansible qui vous permet de **conserver des données sensibles** telles que des mots de passe, des clés SSH, de certificats SSL, des jetons d'API et tout ce que vous ne voulez pas que le public voie, plutôt que de les stocker sous un format brut dans des playbooks ou des rôles.

Comme il est courant de stocker des configurations Ansible dans un contrôleur de version tel que git, nous avons besoin d'un moyen de stocker ces données secrètes en toute sécurité. Le Vault est la réponse à cela, puisqu'il permet de chiffrer n'importe quoi à l'intérieur de votre fichier YAML, ces données de Vault peuvent ensuite être distribuées ou placées dans le contrôle de code source.

Nous utiliserons le système de Vault dans notre ancien projet afin de chiffrer le nouveau mot de passe root et le nom et le mot de passe du nouvel utilisateur. Commencez par télécharger notre ancien projet complet en [cliquant ici](#).

## Manipulation du Vault

---

Pour activer les fonctionnalités Vaults, il suffit d'utiliser l'outil de ligne de commande `ansible-vault` afin d'utiliser ou modifier les fichiers de données secrètes.

### Création notre fichiers chiffrés

Dans notre cas nous allons déplacer les variables `mysql_user`, `mysql_password` et `root_password` de notre

fichier vers notre nouveau fichier de variables secrètes, que nous créons tout de suite.

Placez-vous d'abord à la racine du projet et **créons un nouveau notre fichier de données chiffré**, en exécutant la commande suivante :

```
ansible-vault create vars/mysql-users.ymlCopier
```

Vous serez d'abord invité à saisir et confirmer un mot de passe (**retenez ce mot de passe, il est très important !**)

```
New Vault password:
```

```
Confirm New Vault password: Copier
```

Après avoir fourni votre mot de passe, l'outil lancera l'éditeur que vous avez défini dans la variable d'environnement **\$EDITOR** (par défaut vi). Voici les données remplîtes dans notre nouveau fichier secret:

```

mysql_user: "admin"

mysql_password: "Test_34535$"

root_password: "Test_34049$"Copier
```

Voici à quoi ressemble le contenu de mon fichier chiffré quand je tente de l'afficher :

```
cat vars/mysql-users.ymlCopier
```

**Résultat :**

```
$ANSIBLE_VAULT;1.1;AES256

39316537353463636533303430643963306535386665326361363934613566

626332616337663362

3661646139356433643736616537656266656137313632320a623032383533
```

353334373662623766

36333435303762353434366335373230613966636363643634323465303365

366534313638636364

6664366330626632360a323563366664386332313233643462326436323163

373963363636303439

35633437623332383066353064646363383162363737313939643330396130323664  
313865383764

37623136663433616132643537363536633630333165323562643065666636323030  
613531616537

33613262616438633639656536343061386365396438323762303338613062363738  
633334383662

34363038393966656530386237333464346634326238346339316131326537643963  
623737333234

3437

## Information

Le chiffrement par défaut est AES.

## Édition de fichiers chiffrés

Pour **modifier un fichier chiffré** sur place, utilisez la commande suivante :

```
ansible-vault edit vars/mysql-users.ymlCopier
```

## Modifier le mot de passe des fichiers chiffrés

Si vous souhaitez **modifier votre mot de passe** sur un ou plusieurs fichiers chiffrés par le Vault, vous pouvez le faire avec la commande `rekey` :

```
ansible-vault rekey vars/mysql-users.ymlCopier
```

Vous pouvez aussi spécifier une liste de fichiers comme suit :

```
ansible-vault rekey fichier1.yml fichier2.yml fichier3.ymlCopier
```

Dans ce cas le Vault Ansible vous demandera le mot de passe d'origine ainsi que le nouveau mot de passe pour chaque fichier.

## Chiffage de fichiers non chiffrés

Si vous souhaitez **chiffrer des fichiers bruts existants**, alors utilisez l'option `encrypt`. Cette commande peut également fonctionner sur plusieurs fichiers à la fois:

```
ansible-vault encrypt mon-fichier-non-chiffre.ymlCopier
```

## Déchiffrement des fichiers déjà chiffrés

Si vous avez des fichiers existants que vous ne souhaitez plus conserver chiffrés, vous pouvez les **déchiffrer de façon permanente** en exécutant l'option `decrypt` :

```
ansible-vault decrypt mon-fichier-deja-chiffre.ymlCopier
```

## Affichage du contenu des fichiers chiffrés

Si vous souhaitez **afficher le contenu d'un fichier chiffré sans le modifier**, vous pouvez utiliser l'option `view` comme suit :

```
ansible-vault view vars/mysql-users.ymlCopier
```

Utilisez la touche "q" pour quitter le mot d'affichage.

## Utilisation des fichiers chiffrés dans un Playbook

Il existe différents moyens pour lire vos variables chiffrées directement depuis vos playbooks. On peut utiliser la méthode classique en important notre fichier de variables chiffrées grâce à l'instruction **`vars_files`**. L'utilisation de cette méthode nous donnera le fichier playbook suivant :

```

WEB SERVER
```

```
- hosts: web

 become: yes

 vars_files:

 - vars/main.yml

 - vars/mysql-users.yml

 roles:

 - web

DATABASE SERVER

- hosts: db

 become: yes

 vars_files:

 - vars/main.yml

 - vars/mysql-users.yml

 roles:

 - databaseCopier
```

Lors du lancement de votre playbook utilisez l'option `--ask-vault-pass` afin d'avoir une saisie utilisateur pour fournir votre mot de passe Vault, comme suit :

```
ansible-playbook playbook.yml --ask-vault-passCopier
```



Si vous ne souhaitez **pas** saisir le mot de passe Vault à chaque fois que vous exécutez votre **playbook**, vous pouvez ajouter votre mot de passe Vault à un fichier et référencer le fichier pendant l'exécution. Voici un exemple :

```
echo 'votre_mdp' > .vault_passCopier
```

Si vous utilisez un contrôleur de version (git), assurez-vous d'ajouter le fichier de mot de passe au fichier `.gitignore` afin de l'ignorer pendant votre commit :

```
echo '.vault_pass' >> .gitignoreCopier
```

Lors du lancement de votre playbook remplacez l'option `--ask-vault-pass` par l'option `--vault-password-file` comme suit :

```
ansible-playbook playbook.yml --vault-password-file=.vault_passCopier
```

Vous pouvez aussi ignorer l'option `--ask-vault-pass` en rajoutant le fichier contenant votre mot de passe vault directement sur votre fichier `ansible.cfg` :

```
ansible.cfg

[defaults]

. . .

vault_password_file = ../.vault_pass
```

Lancez ensuite votre playbook comme vous avez l'habitude de le faire :

```
ansible-playbook playbook.ymlCopier
```

## Utilisation d'une variable chiffrée dans un Playbook

Une autre méthode d'utilisation d'une donnée secrète consiste à alimenter simplement une variable chiffrée dans l'instruction **vars**. Pour ce faire vous devez d'abord **créer votre variable secrète** avec l'option `encrypt_string`, comme suit :

```
ansible-vault encrypt_string 'secret-text' --name 'secret'Copier
```

## Résultat :

```
secret: !vault |

$ANSIBLE_VAULT;1.1;AES256

33616163316438306663303330376334363862343430363432343536313835323132
353939376438

3439333730353532346630626133666434363932303133650a343137353735363138
646536633432

39303765306633643830353238646433356230623537363466373438323931353763
666537386163

3062303334343830370a333939346632346134313063363136363038356266303835
363331376631

3664
```

Ansible vous demandera ensuite de saisir le mot de passe Vault pour chiffrer votre variable :

```
New Vault password:

Confirm New Vault password:
```

### Information

Si vous avez déjà spécifié l'option **vault\_password\_file** dans le fichier *ansible.cfg*, alors ansible nous vous demandera pas de saisir un mot de passe et utilisera le mot de passe de votre fichier Vault.

Ensuite il suffit de copier le résultat dans votre instruction **vars** comme suit :

```

- hosts: localhost

 vars:
```

```

- secret: !vault |

 $ANSIBLE_VAULT;1.1;AES256

33616163316438306663303330376334363862343430363432343536313835323132
353939376438

3439333730353532346630626133666434363932303133650a343137353735363138
646536633432

39303765306633643830353238646433356230623537363466373438323931353763
666537386163

3062303334343830370a333939346632346134313063363136363038356266303835
363331376631

 3664

tasks:

- debug: var=secret

- fail:

 when: 1 == 1Copier

```

Exécutez ensuite votre playbook avec la commande habituelle :

```
ansible-playbook playbook.yml --ask-vault-passCopier
```

## Conclusion

---

Nous avons enfin réussi à rendre notre playbook modulable grâce à l'utilisation des rôles et sécurisé grâce à l'utilisation des Vaults Ansible. Vous retrouverez l'intégralité du projet depuis mon repository Github [ici](#).

# Créez vos propres modules ansible

Dans ce chapitre, nous apprendrons à créer nos propres Ansible afin de mieux correspondre à nos besoins.

## Introduction

---

Chaque jour, le nombre de modules Ansible continue de croître avec un support supplémentaire ajouté à mesure que l'utilisation d'Ansible continue. On peut donc se poser la question suivante ? **"Pourquoi aurions-nous besoin de créer nos propres modules Ansible ?"** . Eh bien, il y a un certain nombre de raisons, mais je pense que la plus valable reste de personnaliser un module qui n'existe pas encore dans la bibliothèque Ansible, mais aussi de mieux comprendre le fonctionnement des modules Ansible.

### Prérequis

Tout ce dont vous avez besoin est un peu de connaissance de Python, combinez-le avec vos compétences Ansible et vous pourrez commencer à **créer vos propres modules Ansible**. Dans ce chapitre tout sera expliqué au fur et à mesure que nous le parcourons ensemble.

## Création d'un module personnalisé

---

### Hello world

Vous allez vous rendre compte qu'il est très facile de créer des modules Ansible. Notre but pour le moment est d'**afficher le fameux message "Hello world"**.

Commencez déjà par créer un fichier sous un dossier *library* et rajoutez-y le contenu suivant :

```
#!/usr/bin/python
-*- coding: utf-8 -*-
```

```

from ansible.module_utils.basic import *

def main():

 module = AnsibleModule(argument_spec={})

 response = {"result" : "hello world"}

 module.exit_json(changed=False, meta=response)

if __name__ == '__main__':

 main()

```

Voici ci-dessous une liste d'explication du code ci-dessus :

- **#!/usr/bin/python**: l'interpréteur python qui sera utilisé par Ansible.
- **# -\*- coding: utf-8 -\*-**: l'encodage qui sera utilisé par Ansible.
- **from ansible.module\_utils.basic import \***: importation de la librairie permettant de créer des modules Ansible.
- **main()**: le point d'entrée dans votre module.
- **AnsibleModule()**: c'est la classe qui nous permet de créer et manipuler notre module Ansible, comme par exemple la gestion des paramètres de notre module.
- **response = {"result" : "hello world"}**: ce sont les métadonnées de notre module sous forme d'un dictionnaire.
- **module.exit\_json()**: cette partie désigne la fin d'exécution de notre module, elle permet l'affichage des métadonnées et l'état de votre module.

Sur la racine de votre projet créez votre playbook et appelez-y votre module comme suit :

```

- hosts: localhost

tasks:

```

```
- name: Test de notre module

 test:

 register: result

- debug: var=resultCopier
```

Si vous avez suivi à la lettre mes instructions, vous devriez avoir l'arborescence suivante :

```
|— library
| └─ test.py
└─ main.yml
```

Exécutez ensuite votre playbook et vous obtiendrez le résultat suivant :

```
ansible-playbook main.ymlCopier
```

### Résultat :

```
TASK [Test de notre module] *****

ok: [localhost]

TASK [debug] *****

ok: [localhost] => {

 "result": {

 "changed": false,

 "failed": false,

 "meta": {
```

```
 "result": "hello world"

 }

}
```

## Les paramètres

Pour que notre module soit plus utile, nous aurons besoin de **rajouter quelques entrées**. Dans notre exemple nous allons imaginer un module qui permet de décrire un personnage de jeux vidéo avec les spécifications et les exigences suivantes :

Nom	Type	Valeur par défaut
name	string	unknown
description	string	empty
attack	string avec choix limité	melee
inventory	list	[] (liste vide)
monster	bool	False

Transformons ce tableau sous forme de module Ansible, ce qui nous donnera le code suivant :

```
#!/usr/bin/python

-*- coding: utf-8 -*-

from ansible.module_utils.basic import *

def main():

 fields = {

 "name": {"default" : "unknown", "type": "str"},
```

```

 "description": {"default": "empty", "required": False,
"type": "str"},

 "attack": {

 "default": "melee",

 "choices": ['melee', 'distance'],

 "type": 'str'

 },

 "inventory": {"default": [], "required": False,
"type": "list"},

 "monster": {"default": False, "required": False,
"type": "bool"},

 }

 module = AnsibleModule(argument_spec=fields)

 module.exit_json(changed=False, meta=module.params)

if __name__ == '__main__':

 main()

```

Vous remarquerez les options suivantes :

- **default**: la valeur par défaut de votre paramètre dans le cas où l'utilisateur ne spécifie rien.
- **choices**: la liste des valeurs possibles à proposer à l'utilisateur final.
- **type**: le type de votre paramètre (str, bool, int dict, list, etc ...).
- **required**: un booléen afin de savoir si le paramètre est obligatoire ou non (True ou False).

## Information



Les paramètres sont par défaut obligatoires.

Faisons appel à notre beau module depuis notre playbook, comme suit :

```
- hosts: localhost

 tasks:

 - name: Test de notre module

 test:

 name: "MageDarkX"

 attack: distance

 inventory:

 - powder

 - stick

 - potion

 register: result

- debug: var=resultCopier
```

Puis exécutons notre playbook :

```
ansible-playbook main.ymlCopier
```

**Résultat :**

```
TASK [debug] *****

ok: [localhost] => {

 "result": {
```

```
 "changed": false,

 "failed": false,

 "meta": {

 "attack": "distance",

 "description": "empty",

 "inventory": [

 "powder",

 "stick",

 "potion"

],

 "monster": false,

 "name": "MageDarkX"

 }

}
```

Vous pouvez aussi **récupérer chaque entrée dans une variable unique**. Pour ce faire, créons une fonction nommée **presentation()** afin de mieux présenter notre personnage selon les entrées récupérées de notre utilisateur :

```
#!/usr/bin/python

-*- coding: utf-8 -*-

from ansible.module_utils.basic import *
```

```
def presentation(module):

 name = module.params['name']

 attack = module.params['attack']

 inventory = module.params['inventory']

 return {"Presentation" : "My name is {} and my type of attack
is {}, here is what you will find in my inventory : {}".format(name,
attack, inventory)}

def main():

 fields = {

 "name": {"default" : "unknown", "type": "str"},

 "attack": {

 "default": "melee",

 "choices": ['melee', 'distance'],

 "type": 'str'

 },

 "inventory": {"default": [], "required": False,
"type": "list"},

 }

 module = AnsibleModule(argument_spec=fields)

 module.exit_json(changed=False, meta=presentation(module))
```

```
if __name__ == '__main__':
 main()Copier
```

### Résultat :

```
TASK [debug] *****

ok: [localhost] => {
 "result": {
 "changed": false,
 "failed": false,
 "meta": {
 "Presentation": "My name is MageDarkX and my type of
attack is distance,
 here is what you will find in my inventory : ['powder',
'stick', 'potion']"
 }
 }
}
```

## Attributs et méthodes proposées par la librairie Ansible

Si jamais vous souhaitez **afficher tous les attributs et méthodes de la class `AnsibleModule`**, vous pouvez utiliser la fonction native de python nommé **`def()`**, comme suit :

```
#!/usr/bin/python
```

```
-*- coding: utf-8 -*-

from ansible.module_utils.basic import *

def main():

 module = AnsibleModule(argument_spec={})

 module.exit_json(changed=False, meta=dir(module))

if __name__ == '__main__':

 main()Copier
```

Pour tester plus rapidement notre module, utilisons l'option **-M** de la commande `ansible` qui permet de préciser le chemin de la librairie d'un module :

```
ansible all -M library -m testCopier
```

### Résultat :

```
TASK [debug]

ok: [localhost] => {

 "result": {

 "changed": false,

 "failed": false,

 "meta": [
```

```

 "fail_json"

 "__setattr__",

 "__sizeof__",

 ...

 ...

 "do_cleanup_files",

 ...

 "warn"

]

}

}

```

Dans la prochaine partie, nous nous intéresserons à deux méthodes bien utiles, à savoir la méthode `fail_json()` et `warn()`.

## Afficher un message d'avertissement et déclencher une erreur

La fonction pour **afficher un message d'avertissement** est la méthode `warn()`, elle reste très simple à utiliser et permet par exemple d'indiquer un message de recommandation sans interrompre l'exécution du module, ci-dessous un exemple d'utilisation :

```

#!/usr/bin/python

-*- coding: utf-8 -*-

from ansible.module_utils.basic import *

```

```

def verifyAge(module):

 age = module.params['age']

 if age < 18:

 module.warn("Attention vous êtes mineur, un accord
parental est requis.")

def main():

 fields = {

 "age": {"type": "int"}

 }

 module = AnsibleModule(argument_spec=fields)

 verifyAge(module)

 module.exit_json(changed=False, meta=module.params['age'])

if __name__ == '__main__':

 main()

```

Copier

Exécutons notre script avec un âge inférieur à 18 :

```
ansible all -M library -m test -a "age=17" Copier
```

**Résultat :**

```
[WARNING]: Attention vous êtes mineur, un accord parental est requis.
```

```
localhost | SUCCESS => {

 "changed": false,

 "meta": 17

}
```

Si vous souhaitez à la place **déclencher une erreur**, vous utiliserez alors la méthode **fail\_json()** à la place de la méthode **warn()**, voici un exemple d'utilisation :

```
...

def verifyAge(module):

 age = module.params['age']

 if age < 18:

 module.fail_json(msg="Attention vous êtes mineur, un accord parental est requis.")

...Copier
```

Exécutons une nouvelle fois notre ancienne commande :

```
ansible all -M library -m test -a "age=17" Copier
```

**Résultat :**

```
localhost | FAILED! => {

 "changed": false,
```



```
"msg": "Attention vous êtes mineur, un accord parental est requis."
}
```

## Exercice

---

### But

Les notions vues précédemment répondront à la plupart de vos besoins. Vous êtes donc désormais capable de rédiger vos propres modules, et si vous le permettez, je vais vous demander de vous exercer en créant un module qui permet de vérifier si un dossier est capable de stocker des données d'une taille entrée par utilisateur.

L'utilisateur final qui utilisera votre playbook devra spécifier deux paramètres :

- **path**: chemin du dossier qui sera analysé (exemple: '/home/hatim/')
- **size**: la taille de comparaison (exemple: '200g' pour 200 Gigaoctets, '200m' pour 200 Mégaoctets, etc ..)

Un exemple sera plus parlant, voici par exemple à quoi doit ressembler notre playbook :

```
- hosts: localhost

 tasks:

 - name: 'check folder size of /home/hatim'

 folder_space:

 path: '/home/hatim'

 size: '20g'
```

Si nous ne pouvons pas stocker 20Go de données dans le dossier */home/hatim*, nous devrions avoir un résultat similaire à celui-ci :

```
TASK [check folder size of /home/hatim] *****
```

```
fatal: [localhost]: FAILED! => {"changed": false, "msg": "Not enough space available on /home/hatim (found = 15.0g, need = 20g)"}
```

Dans le cas où le dossier `/home/hatim` possède les 20Go d'espace de stockage nécessaires, alors nous obtiendrons un résultat similaire à celui-ci :

```
TASK [check folder size of /home/hatim] *****
ok: [localhost] => {
 "result": {
 "changed": false,
 "failed": false,
 "meta": {
 "result": "You have enough space :)"
 }
 }
}
```

## Aide (sans solution)

Voici quelques fonctions qui vous permettront d'accomplir certaines tâches. La première astuce est une fonction qui vous permettra de récupérer la taille en octet que peut stocker votre dossier :

```
import os

def getAvailableSpace(path):

 try:

 statvfs = os.statvfs(path)
```

```

 return int(statvfs.f_bavail * statvfs.f_frsize)

 except OSError as e:

 print(e)

path='/home/hatim'

print("byte of {} : {}".format(path,
getAvailableSpace(path)))Copier

```

Maintenant je vais vous montrer comment convertir la taille d'un octet (byte) en Mégaoctets, Gigaoctets, etc .. et inversement :

```

size_conversion = { "k": 1024, "m": pow(1024,2), "g": pow(1024,3),
"t": pow(1024,4), "p": pow(1024,5) }

def convertToByte(space_unit, space):

 global size_conversion

 return int(space * size_conversion[space_unit])

def convertByteToOriginal(space_unit, space):

 global size_conversion

 return round(space / float(size_conversion[space_unit]), 2)

mb_to_byte = convertToByte('m', 1000)

```

```

gb_to_byte = convertToByte('g', 1)

print("1000 Mb to byte : {}".format(mb_to_byte))

print("1 Gb to byte : {}".format(gb_to_byte))

b_to_gb = convertByteToOriginal('g', 10000000)

print("10000000 byte to Gb : {}".format(b_to_gb))Copier

```

Voilà, maintenant c'est à votre tour !

## Solution

Je vais vous montrer ma solution, sachez juste qu'il existe différentes façons de faire ce type de module, donc n'hésitez pas à le modifier selon votre guise et de partager votre code dans l'espace commentaire ;).

Voici donc à quoi ressemble le code de mon module :

```

#!/usr/bin/python

-*- coding: utf-8 -*-

from ansible.module_utils.basic import *

import os

size_conversion = { "k": 1024, "m": pow(1024,2), "g": pow(1024,3),
"t": pow(1024,4), "p": pow(1024,5) }

```

```
def checkSizeAndUnit(module, space_unit, space):

 if space_unit not in 'kmgtp':

 module.fail_json(msg="Bad size specification for unit
{0}".format(space_unit))

 if not space.isdigit():

 module.fail_json(msg="Bad value for {0}, must be an
integer".format(space))

def getSizeAndUnit(module, size):

 space_unit, space = size[-1].lower(), size[0:-1]

 checkSizeAndUnit(module, space_unit, space)

 return space_unit, int(space)

def convertToByte(space_unit, space):

 global size_conversion

 return int(space * size_conversion[space_unit])

def convertByteToOriginal(space_unit, space):

 global size_conversion

 return round(space / float(size_conversion[space_unit]), 2)
```

```
def getWantedSpace(module, size):

 space_unit, space_wanted = getSizeAndUnit(module, size)

 return convertToByte(space_unit, space_wanted)

def getAvailableSpace(module, path):

 try:

 statvfs = os.statvfs(path)

 return int(statvfs.f_bavail * statvfs.f_frsize)

 except OSError as e:

 module.fail_json(msg="{}".format(e))

def CheckSizeAvailability(module, path, size):

 space_available = getAvailableSpace(module, path)

 space_wanted = getWantedSpace(module, size)

 space_unit, _ = getSizeAndUnit(module, size)

 space_available_converted = convertByteToOriginal(space_unit,
space_available)
```

```

 if space_available < space_wanted:

 module.fail_json(msg="Not enough space available on {}
(found = {}}, need = {})".format(path, space_available_converted,
space_unit, size))

def main():

 fields = {

 "path": {"required": True, "type": "str" },

 "size": {"required": True, "type": "str" }

 }

 module = AnsibleModule(argument_spec=fields)

 path = module.params['path']

 size = module.params['size']

 CheckSizeAvailability(module, path, size)

 response = {"result" : "You have enough space :)}"

 module.exit_json(changed=False, meta=response)

if __name__ == '__main__':

 main()Copier

```

# Conclusion

---

Vous pouvez dorénavant personnaliser vos modules pour qu'ils répondent exactement à vos besoins. Tentez tout de même de privilégier les modules préconçus par Ansible avant de songer à créer vos propres modules.

Je tiens à créditer cet [article](#) qui m'a aidé à la rédaction de cet exercice. J'en ai d'ailleurs profité pour améliorer la structure de son code comme vous pouvez le remarquer :).

# Déboguer vos playbooks Ansible

Dans ce chapitre, nous allons apprendre différentes façons de déboguer nos Playbooks Ansible.

## Introduction

---

Lors du dépannage des problèmes Ansible, il est utile de savoir comment **tester votre playbook en toute sécurité, activer le mode de débogage et augmenter le niveau de verbosité**. Nous allons apprendre à faire tout cela dans ce chapitre.

## Les différentes manières de débogage

---

### Mode vérification

Des fois, vous aurez besoin de vérifier si une tâche est fonctionnelle sur vos serveurs distants sans avoir à effectuer de modifications sur vos systèmes distants. Pour ce cas d'utilisation, il faut penser à l'option **--check** de la commande `ansible-playbook`.

```
ansible-playbook main.yml --checkCopier
```

Parfois, vous souhaitez peut-être **enter en mode de vérification sur des tâches individuelles**. Cela peut se faire via l'instruction **check\_mode**, qui peut être ajoutée à vos tâches, comme suit :



```
tasks:

- name: Add test text in /home/hatim/hatim.txt

 lineinfile:

 line: "test"

 dest: /home/hatim/hatim.txt

 state: present

 check_mode: yesCopier
```

Si vous souhaitez utiliser l'option **--check** et **ignorer les erreurs sur certaines tâches en mode vérification**, vous pouvez utiliser la variable booléenne **ansible\_check\_mode** qui aura comme valeur **True** pendant le mode vérification :

```
tasks:

- name: Add test text in /home/hatim/notfound.txt

 lineinfile:

 line: "test"

 dest: /home/hatim/notfound.txt

 state: present

 ignore_errors: '{{ ansible_check_mode }}'

 # Vous pouvez aussi l'instruction suivante: when: not
 ansible_check_mode

- debug: msg="Si vous voyez ce message c'est que la tâche
précédente a été ignorée"Copier
```

Exécutons le playbook pour voir le résultat :

```
ansible-playbook main.yml --checkCopier
```

### Résultat :

```
TASK [Add test text in /home/hatim/notfound.txt] *****

fatal: [localhost]: FAILED! => {"changed": false, "msg":
"Destination /home/hatim/notfound.txt does not exist !", "rc": 257}

...ignoring

TASK [debug] *****

ok: [localhost] => {

 "msg": "Si vous voyez ce message c'est que la tâche précédente a
été ignorée"

}
```

## Mode différenciation

C'est bien beau tout ça, mais ça serait mieux si Ansible nous **avertit des modifications qu'il est supposé apporter** sur la machine cible. Pour cela il existe l'option **--diff** qui est principalement utilisée dans les modules qui manipulent des fichiers, mais d'autres modules peuvent également afficher des informations «avant et après» (par exemple, le module **user**). D'ailleurs, elle fonctionne très bien avec l'option **--check** :

### Information

Étant donné que l'option diff produit une grande quantité de sortie, il est préférable de l'utiliser lors de la vérification d'un seul hôte à la fois

Reprenons l'exécution de notre exemple précédent :

```
tasks:

 - name: Add test text in /home/hatim/hatim.txt
```

```

lineinfile:

 line: "test"

 dest: /home/hatim/hatim.txt

 state: presentCopier

ansible-playbook main.yml --check --diffCopier

```

### Résultat :

```

TASK [Add test text in /home/hatim/hatim.txt]

--- before: /home/hatim/hatim.txt (content)

+++ after: /home/hatim/hatim.txt (content)

@@ -0,0 +1 @@

+test

changed: [localhost]

```

Si vous souhaitez **activer la différence sur une tâche particulière**, vous pouvez utiliser l'instruction booléenne **diff**. Selon comment vous lancez votre playbook, vous pouvez déclencher les comportements suivants :

- Si vous n'êtes pas en mode différenciation et que l'instruction **diff** est à **True** alors seule la tâche exécutant cette instruction entrera en mode différenciation.
- Si êtes en mode différenciation et que l'instruction **diff** est à **False** alors seule la tâche exécutant cette instruction ignorera le mode différenciation.

Voici comment elle s'intègre dans un playbook :

```

tasks:

- name: Add test text in /home/hatim/hatim.txt

 lineinfile:

```

```
 line: "test"

 dest: /home/hatim/hatim.txt

 state: present

diff: noCopier
```

## Le debugger des playbooks

Ansible inclut un débogueur, ce débogueur vous permet de **déboguer vos tâches**, c'est-à-dire que vous avez accès à toutes les fonctionnalités du débogueur dans le contexte de la tâche. Vous pouvez par exemple, vérifier ou modifier la valeur des variables de votre playbook, mettre à jour les arguments de votre module/tâche et relancer votre tâche avec les nouvelles variables et les nouveaux arguments pour vous aider à résoudre la cause de l'échec.

Pour utiliser le débogueur d'Ansible vous devez utiliser l'instruction **debugger** qui peut prendre les valeurs suivantes :

- **always**: appeler toujours le débogueur, quel que soit le résultat.
- **never**: n'invoquer jamais le débogueur, quel que soit le résultat.
- **on\_failed**: n'appeler le débogueur qu'en cas d'échec d'une tâche.
- **on\_unreachable**: n'appeler le débogueur que si l'hôte a est inaccessible.
- **on\_skipped**: n'appeler le débogueur uniquement si la tâche est ignorée.

L'instruction **debugger** peut être utilisée sur une tâche particulière ou pour toutes vos tâches.

Pour une tâche particulière :

```
- name: Execute a command

 command: ls -l /home/hatim

 debugger: on_failedCopier
```

Pour toutes vos tâches :

```
- hosts: all

 debugger: on_failed
```

```
tasks:

 - name: Execute a command

 command: ls -l /home/hatimCopier
```

Quand vous entrez en mode debug vous pouvez utiliser différentes commandes :

- `p task/task_vars/host/result`: affiche des informations utilisées pendant l'exécution par votre tâche module (le `p` équivaut au `print`).
- `task.args[key] = value`: met à jour l'argument du module.
- `task_vars[key] = value`: met à jour les variables de votre playbook.
- `update_task`: si vous modifiez les `task_vars` alors utilisez cette commande pour recréer la tâche à partir de votre nouvelle structure de données.
- `redo`: exécutez à nouveau la tâche.
- `continue`: passe à la tâche suivante.
- `quit`: quitte le débogueur. L'exécution du playbook est abandonnée.

Voici ci-dessous un exemple d'utilisation, je vais intentionnellement utiliser le module `lineinfile` avec un fichier qui n'existe pas, et par la suite je vais déboguer cette tâche pour remettre le bon chemin de mon fichier. Voici déjà à quoi ressemble mon playbook avec notre mauvaise valorisation :

```
tasks:

 - name: Add test text in /home/hatim/hatim.txt

 lineinfile:

 line: "test"

 dest: /home/hatim/hatim_fault.txt

 state: present

 debugger: on_failedCopier
```

En exécutant notre playbook, nous nous retrouverons en mode debug :

```
ansible-playbook main.ymlCopier
```

**Résultat :**

```
TASK [Add test text in /home/hatim/hatim.txt] *****

fatal: [localhost]: FAILED! => {"changed": false, "msg":
"Destination /home/hatim/hatim_fault.txt does not exist !", "rc":
257}

[localhost] TASK: Add test text in /home/hatim/hatim.txt (debug)>
```

Nous savons par avance que l'erreur vient de la mauvaise valorisation du paramètre **dest**, nous allons donc modifier sa valeur dans notre débogueur. Voici la démarche que je vais suivre pour arriver à notre but :

- Afficher plus d'informations sur notre erreur.
- Afficher les valeurs des différents arguments de notre tâche.
- Modifier la valeur de notre paramètre **dest**.
- Afficher la nouvelle valeur de notre paramètre **dest**.
- Exécuter de nouveau notre tâche.

Si nous transformons notre démarche sous forme de commandes dans notre outil de débogage, nous taperons les commandes suivantes :

```
(debug)> p result._result

{'_ansible_no_log': False,

 '_ansible_parsed': True,

 'changed': False,

 u'failed': True,

 u'invocation': {u'module_args': {u'attributes': None,

 ...

 u'delimiter': None,

 u'dest':

u'/home/hatim/hatim_fault.txt',

 u'directory_mode': None,

 ...
```

```
u'msg': u'Destination /home/hatim/hatim_fault.txt does not exist
!'
```

```
u'rc': 257}
```

```
(debug)> p task.args
```

```
{'_ansible_check_mode': False,
```

```
...
```

```
u'dest': u'/home/hatim/hatim_fault.txt',
```

```
u'line': u'test',
```

```
u'state': u'present'}
```

```
(debug)> task.args['dest'] = '/home/hatim/hatim.txt'
```

```
(debug)> p task.args['dest']
```

```
'/home/hatim/hatim.txt'
```

```
(debug)> redo
```

```
changed: [localhost]
```

```
PLAY RECAP
```

```


```

```
localhost : ok=1 changed=1 unreachable=0 failed=0
```

Enfin vous pouvez activer le mode débogage pour chaque exécution de votre playbook en mettant la variable **enable\_task\_debugger** à **True** dans le fichier :

```
[defaults]

enable_task_debugger = True
```

## La verbosité dans Ansible

Pour comprendre ce qui se passe lorsque vous exécutez le playbook, vous pouvez exécuter la commande `ansible-playbook` avec l'option verbose **-v**. Chaque "v" supplémentaire fournira à l'utilisateur final plus de sortie de débogage (vous pouvez aller jusqu'à 4 niveaux de verbosité) :

```
verbosité niveau 0

ansible-playbook main.yml

verbosité niveau 1

ansible-playbook -v main.yml

verbosité niveau 2

ansible-playbook -vv main.yml

verbosité niveau 3

ansible-playbook -vvv main.yml

verbosité niveau 4

ansible-playbook -vvvv main.ymlCopier
```



Vous pouvez également modifier la fichier  
pour **chaque exécution de vos playbooks** :

pour **appliquer un niveau de verbosité**

```
[defaults]
verbosity = 3
```

## Conclusion

---

Nous avons étudié de nombreuses manières de procéder à des tests sécurisés de vos playbooks , et vous avez désormais tous les outils à votre disposition pour identifier et corriger vos bogues à partir de votre playbook. Rendez-vous au prochain chapitre où nous aborderons la notion de performance pour nos playbooks.

# Augmenter les performances de votre Playbook

Dans ce chapitre, nous étudierons comment réduire le temps d'exécution de vos Playbooks.

## Introduction

---

Ansible est un outil pratique qui ne nécessite pas de configuration complexe en raison de sa nature agentless (vous n'avez pas besoin de préinstaller de logiciel ou d'agent sur des hôtes gérés, plus d'informations [ici](#)). Dans la plupart des cas, vous utiliseriez une connexion avec le protocole ssh pour configurer vos serveurs distants. L'un des inconvénients de cette simplicité est la vitesse.

En effet, en fonction de votre environnement et du flux de travail de votre playbook, Ansible peut fonctionner lentement, puisqu'il fait toute la logique localement, génère l'exécution de tâches, l'envoie à l'hôte distant, l'exécute, attend les résultats, lit les résultats, les analyses et continue vers la tâche suivante. Dans cet article, nous décrivons **plusieurs façons d'augmenter la vitesse de notre playbook**.

# Améliorer les performances de notre Playbook

---

Tout au long de cet article, nous prendrons comme exemple de test le playbook ci-dessous, avec le contenu suivant:

```
- hosts: test

vars:

 - test_path: "/home/hatim/hatim.txt"

tasks:

 - name: "Send {{ test_path }}"

 copy:

 content: "Ceci est une ligne de test"

 dest: '{{ test_path }}'

 - name: "Add text in {{ test_path }}"

 lineinfile:

 line: "Ceci est une autre ligne de test"

 dest: '{{ test_path }}'

 state: present
```

## Mesurer le temps d'exécution de nos tâches Ansible

Avant d'optimiser quoi que ce soit, nous devrions au préalable être capable de **mesurer les performances des tâches de notre playbook**. Pour, ce faire nous modifierons la variable `callback_whitelist` dans le fichier `ansible.cfg` avec les valeurs suivantes :

```
[defaults]

callback_whitelist = timer, profile_tasks
```

Après avoir ajouté cette ligne, vous commencerez à **voir le temps d'exécution de chaque tâche** ainsi que le temps d'exécution de votre playbook :

```
ansible-playbook main.ymlCopier
```

### Résultat :

```
TASK [Gathering Facts] *****

Tuesday 25 February 2020 15:50:02 +0100 (0:00:00.068)
0:00:00.068 *****

ok: [localhost]

TASK [Send /home/hatim/hatim.txt]

Tuesday 25 February 2020 15:50:05 +0100 (0:00:03.158)
0:00:03.226 *****

changed: [localhost]

TASK [Add text in /home/hatim/hatim.txt]

Tuesday 25 February 2020 15:50:07 +0100 (0:00:01.595)
0:00:04.822 *****

changed: [localhost]

PLAY RECAP

```

```
localhost : ok=3 changed=2 unreachable=0 failed=0
```

```
Tuesday 25 February 2020 15:50:07 +0100 (0:00:00.840)
0:00:05.663 *****
```

```
=====
=====
```

```
Gathering Facts -----
----- 3.16s
```

```
Send /home/hatim/hatim.txt -----
----- 1.60s
```

```
Add text in /home/hatim/hatim.txt -----
----- 0.84s
```

```
Playbook run took 0 days, 0 hours, 0 minutes, 5 seconds
```

## La collecte de Facts

### Désactiver la collecte des Facts

Au début de l'exécution de votre playbook, Ansible collecte des informations sur chaque système distant (il s'agit du comportement par défaut de la commande `ansible-playbook`). Vous remarquerez que cette étape nous a pris 3.16s de temps d'exécution. Si vous n'avez pas besoin de récupérer les facts, alors **cette étape peut être ignorée depuis votre playbook**, voici comment ça se passe :

```
- hosts: test

 gather_facts: no

 # suite de votre playbookCopier
```

De ce fait, nous gagnerons quelques secondes de performances :

```
ansible-playbook main.ymlCopier
```

## Résultat :

```
=====
Send /home/hatim/hatim.txt ----- 1.62s
Add text in /home/hatim/hatim.txt ----- 0.79s
Playbook run took 0 days, 0 hours, 0 minutes, 2 seconds
```

### Filtrer la collecte des Facts

Dans le cas où vous souhaitez **récupérer uniquement certains facts**, et donc d'intercepter exactement juste ce dont vous avez besoin, il suffit alors d'affiner la collecte de facts à l'intérieur de votre playbook.

En effet, le module `setup` propose un paramètre intéressant qui se nomme : **`gather_subset`** qui permet de collecter et ignorer un sous-ensemble de facts. Dans l'exemple ci-dessous, nous l'utiliserons pour ne récupérer que des informations sur la partie réseau de nos systèmes distants :

```
- hosts: test

gather_facts: no

vars:
 - test_path: "/home/hatim/hatim.txt"

pre_tasks:
 - name: Collect only facts returned by network

 setup:

 gather_subset:
 - '!all'

 - '!any'
```

```

- 'network'

tasks:

- debug:

 msg: 'Mon ip {{ ansible_default_ipv4.address }}'

suite du playbookCopier

```

Au lancement de notre playbook, on arrive à économiser le temps d'exécution des facts :

```

Send /home/hatim/hatim.txt ----- 1.46s

Collect only facts returned by network -----
----- 1.16s

Add text in /home/hatim/hatim.txt -----
----- 0.75s

debug -----
----- 0.05s

```

Voici les options disponible dans **gather\_subset** :

- **all**: rassembler tous les sous-ensembles (par défaut).
- **network**: recueillir des facts sur le réseau.
- **hardware**: recueillir des facts sur le matériel (facts les plus longs à récupérer).
- **virtual**: recueillir des informations sur les machines virtuelles hébergées sur la machine.
- **ohai**: recueillir des facts d'Ohai (facts de chef's).
- **facter**: recueillir des facts de facter (facter de puppet's).

Vous pouvez également **contrôler le processus de collecte des facts par configuration globale** dans votre fichier `ansible.cfg`, comme suit :

```

[defaults]

gather_subset = !hardware,!ohai,!facter

```

## Mise en cache des Facts

Vous pouvez aussi essayer d'utiliser la mise en cache des facts. Dans ce cas, nous demanderons à Ansible de conserver les facts pour un hôte donné qu'il recueille dans un fichier local. Vous pouvez également définir d'autre type de cache (memcache, redis etc ... plus d'informations [ici](#)). Pour ce faire, il suffit de rajouter l'option `fact_caching_connection` et `fact_caching` dans le fichier `ansible.cfg`, comme suit :

```
[defaults]

fact_caching_connection = /tmp/.ansible_fact_cache

fact_caching = jsonfile

fact_caching_timeout = 7200
```

### Information

`fact_caching_timeout = 7200` correspond à 2 heures de mise en cache.

## Le protocole ssh

Laissez-moi d'abord vous expliquer de façon très simple, **le workflow par défaut utilisé par Ansible pour exécuter les tâches de notre playbook** :

1. Génération d'un package d'exécution avec le module et ses paramètres pour une exécution à distance.
2. Connexion via SSH pour envoyer notre fichier via SFTP.
3. Vérification du code retour de votre module: si déclenchement d'erreur alors arrêt du playbook sinon il continue son exécution.
4. Connexion via SSH pour détecter l'existence de notre fichier distant et rajouter la ligne adéquate.
5. Vérification du code retour de votre module: si déclenchement d'erreur alors arrêt du playbook sinon il continue son exécution.
6. Résumé de l'exécution.

Vous remarquerez alors qu'ansible utilise à chaque fois le protocole ssh pour exécuter ses tâches. Il est donc important de s'intéresser de plus près à ce protocole pour améliorer nos performances.

Dans cette section, je vais vous dévoiler quelques astuces qui vous permettront d'optimiser le temps d'exécution de vos tâches. Comment ? Le but est d'**améliorer la façon dont le serveur maître utilise le protocole ssh** pour communiquer avec vos serveurs distants. Ces méthodes pourront vous apporter des avantages considérables en matière de performances, surtout si vous exécutez un grand nombre de tâches ou que vous exécutez sur un grand nombre d'hôtes, ou les deux.

### Le multiplexage SSH

L'idée derrière le multiplexage SSH est qu'une fois qu'une connexion ssh est établie avec un hôte distant, la **connexion restera persistante en arrière-plan** pendant une période de temps donnée. Donc, chaque fois que vous devez réexécuter vos tâches sur le même hôte, la même connexion sera réutilisée.

Nous allons dans un premier lieu, vérifier comment le protocole ssh est utilisé pour une simple exécution du module **ping**, pour cela exécuter la commande suivante avec un maximum de verbosité :

```
ansible test -m ping -vvvv | grep EXEC
```

#### Résultat filtré :

```
<localhost> SSH: EXEC ssh -vvv -C -o
ControlMaster=auto -o ControlPersist=60s
-o PreferredAuthentications=gssapi-with-mic,gssapi-
keyex,hostbased,publickey
ControlPath=/home/hatim/.ansible/cp/9ab9540323
```

Les options affichées sont des options de la commande **ssh**, ci-dessous leur explication :

- **ControlMaster auto** : ici on laisse SSH se charger de la détection de l'existence d'un socket (connexion) pour la connexion à l'host demandée.
- **ControlPersist=60s** : chaque connexion restera active (en arrière-plan) pendant 60 secondes au maximum.
- **PreferredAuthentications** : spécifie l'ordre dans lequel le serveur de contrôle doit essayer les méthodes d'authentification.
- **ControlPath** : chemin où sera créé le fichier socket.



Une fois que nous avons fait le tour sur l'explication des différentes options utilisées par défaut par ansible, on peut alors s'imaginer les modifications suivantes pour un gain de performances :

- Si nous avons plusieurs tâches à exécuter ou que nous exécutons les mêmes tâches sur le même serveur distant, alors il est préférable d'augmenter la valeur de **ControlPersist**.
- Si vous utilisez uniquement des clés publiques pour la connexion ssh avec les hôtes souhaités, alors vous pouvez supprimer les autres méthodes d'authentification dans l'option **PreferredAuthentications**.

Toutes ces modifications combinées nous donnerons le fichier `ansible.cfg` suivant :

```
[ssh_connection]

ssh_args = -o ControlMaster=auto -o ControlPersist=2h -o
PreferredAuthentications=publickey

control_path = %(directory)s/ansible-ssh-%%h-%%p-%%r
```

L'option **control\_path** n'est pas obligatoire mais je l'ai rajoutée pour sauvegarder de manière plus agréable notre fichier socket (%%h => host, %%p => port, %%r => utilisateur). Si nous relançons notre commande de debug alors nous obtiendrons le résultat avec nos nouvelles options :

```
<localhost> SSH: EXEC ssh -vvv -C -o
ControlMaster=auto -o ControlPersist=2h
-o PreferredAuthentications=publickey
ControlPath=/home/hatim/.ansible/cp/ansible-ssh-%%h-%%p-%%r
```

Vérifions si notre fichier socket existe bel et bien :

```
ls -l /home/hatim/.ansible/cp/Copier
```

**Résultat :**

```
srw----- 1 hatim hatim 0 Feb 26 10:44 ansible-ssh-localhost-22-
root
```

**MISE EN GARDE**

Lorsque vous utilisez le multiplexage SSH avec un temps **ControlPersist** plus long, il peut y avoir un problème potentiel de non connectivité. Voici par exemple un scénario qui m'est déjà arrivé : Mon pc s'est mis en veille et au moment de l'allumer, les connexions se sont interrompues, mais sont restées toujours persistantes, ce qui a interrompu la connectivité ssh aux hôtes multiplexés. Si jamais vous êtes dans ce cas de problème alors tuez les processus contenant le mot 'ssh' et '[mux]', cela pourrait être fait avec la commande suivante :

```
ps faux | grep ssh | grep "\[mux\]" | awk '{print $2}' | xargs
kill Copier
```

### Le Pipelining ssh

La pipeline ssh est la méthode Ansible moderne pour accélérer vos connexions ssh sur le réseau vers les hôtes gérés. C'est une fonctionnalité qui permet de **réduire le nombre de connexions ssh à un hôte**. Lorsqu'elle est activée, l'exécution d'un module se fait en passant les instructions à l'hôte via SSH, les instructions sont alors écrites directement sur le canal STDIN depuis l'interpréteur python, cela conduirait alors à de meilleures performances.

Pour mieux comprendre le but de cette fonctionnalité, le mieux reste de vérifier la pipeline utilisée sans et avec cette option. Pour ce faire, nous lancerons le module ping avec un niveau de verbosité élevé :

```
ansible-playbook main.ymlCopier
```

#### Résultat résumé :

```
SSH : EXEC ssh ...
SSH : EXEC ssh ...
SSH : EXEC sftp ...
SSH : EXEC ssh ... python ... ping.py
```

Avec l'option pipeline ssh activé voici le nouveau résultat obtenu :

```
SSH : EXEC ssh ... python && sleep 0
```

Voici ci-dessous le workflow avec le mode pipeline activé afin de mieux ce qui se passe en arrière-plan :

1. Génération d'un fichier python avec le module et ses paramètres pour une exécution à distance.
2. Connexion via SSH pour exécuter l'interpréteur Python.
3. Envoie du contenu de fichier Python à l'entrée standard de l'interpréteur.
4. Vérification du code retour de votre exécution: si déclenchement d'erreur alors arrêt du playbook sinon il continue l'exécution du fichier python.
5. Résumé de l'exécution.

Nous passons alors d'une seule connexion SSH au lieu de quatre ! Multipliez maintenant cela en un certain nombre de tâches et imaginez le nombre de connexions ssh économisé. L'amélioration de la vitesse est importante, en particulier sur les connexions WAN.

L'activation du pipeline se produit en ajoutant l'option suivante dans la section [ssh\_connection] du fichier `ansible.cfg`:

```
[ssh_connection]
pipelining = True
```

## MISE EN GARDE

Cette option ne peut fonctionner seulement si :

- Désactiver le `requiretty` dans le fichier `/etc/sudoers` sur tous les hôtes gérés.
- Aucun transfert de fichiers dans vos tâches.

## Forks

Les forks sont le **nombre de processus parallèles** à générer lors de la communication avec des hôtes distants. Par défaut, Ansible règle la valeur fork sur 5, ce qui signifie qu'à tout moment donné, Ansible peut exécuter jusqu'à 5 exécutions parallèles.

Si vous exécutez un playbook sur plus de 5 hôtes, il est préférable d'augmenter cette valeur. Vous pouvez augmenter ce nombre dans le fichier `ansible.cfg`:

```
[defaults]
forks = 10
```

Ou vous pouvez également utiliser l'option `-f` de la commande `ansible-playbook`.

## Attention

Notez que ce paramètre dépend de vos capacités matérielles et logicielles, telles que la bande passante réseau et la mémoire disponible sur la machine de contrôle. Limitez vos forks en fonction de vos ressources disponibles.

## Free Strategy

Il existe deux types de stratégies dans ansible. Celle qui est utilisée par défaut est la stratégie **serial** (linéaire) dans laquelle chaque tâche attend la fin de chaque hôte avant de passer à la tâche suivante. Cependant, pour de nombreuses situations, nous voulons simplement que chaque hôte termine son jeu de tâches aussi rapidement que possible sans attendre les autres hôtes.

Voici un exemple de scénario qui m'est déjà arrivé aussi : Mon but était d'installer un serveur NGINX sur cinq serveurs différents et l'un de ces serveurs avait un verrou Yum, les quatre autres serveurs n'ont pas pu donc continuer leurs tâches. Pour éviter ce type de problème, nous utiliserons la stratégie **free** (libre), pour ce faire, il suffit d'utiliser l'instruction **strategy** dans votre playbook, comme suit :

```
- hosts: test

 strategy: free

 tasks:

 # suite de votre playbookCopier
```

### Attention

Pour les playbooks avec des dépendances entre les nœuds, cela pourrait être dangereux, mais pour la plupart des cas d'utilisation, une stratégie libre est idéale.

## Conclusion

---

Ansible est un outil logiciel d'automatisation fantastique, mais l'automatisation est encore meilleure lorsqu'elle fonctionne aussi vite qu'elle le peut :).

J'espère que ces conseils vous aideront à accélérer considérablement les exécutions de vos playbooks Ansible. Sachez juste que ces astuces d'optimisations s'adaptent à des besoins précis, et certains d'entre eux nécessitent soit au préalable quelques prérequis à respecter, soit quelques mesures de précautions à l'utilisation. N'oubliez pas aussi que d'autres gains de vitesse sont possibles en optimisant le code de votre playbook ;).

[Chapitre précédent](#)

# Conclusion du cours Ansible

Clap de fin ! Vous connaissez dès à présent la plupart des concepts de base d'Ansible. Je vous présente ici mon message de conclusion.

## Introduction

---

Vous voici arrivé(e) au terme de ce cours et je tiens vraiment à vous remercier d'avoir pris le temps de suivre ce cours et à vous féliciter pour avoir tenu jusqu'au bout ☐.

À travers ce cours, nous avons vu tous les aspects majeurs d'Ansible, rendant ainsi son utilisation très simple, intéressante et très puissante.

C'est un outil qui se veut simple à l'utilisation mais suffisamment puissant pour automatiser des environnements d'applications complexes à plusieurs niveaux.

Nous avons eu l'occasion de partir sur la base d'un projet Web, que nous avons amélioré et sécurisé à travers les différentes options proposées par Ansible. Nous avons aussi finit par augmenter la performance de notre playbook en étudiant du mieux le protocole ssh mais également les possibilités offertes par le fichier de configuration Ansible.

## La suite ?

---

Ansible reste un outil qui s'intègre très facilement avec plusieurs autres technologies, nous avons par exemple eu l'occasion dans [cet article](#) de provisionner notre cluster Kubernetes à l'aide de vagrant et configurer nos différentes machines virtuelles à l'aide d'ansible.

Des projets Ansible sont sûrement à venir dans de nouveaux articles, donc restez informés en vous abonnant à la newsletter ! Sur ce, je vous souhaite une très bonne continuation !



[Chapitre précédent](#)

[Mon jeu](#)