

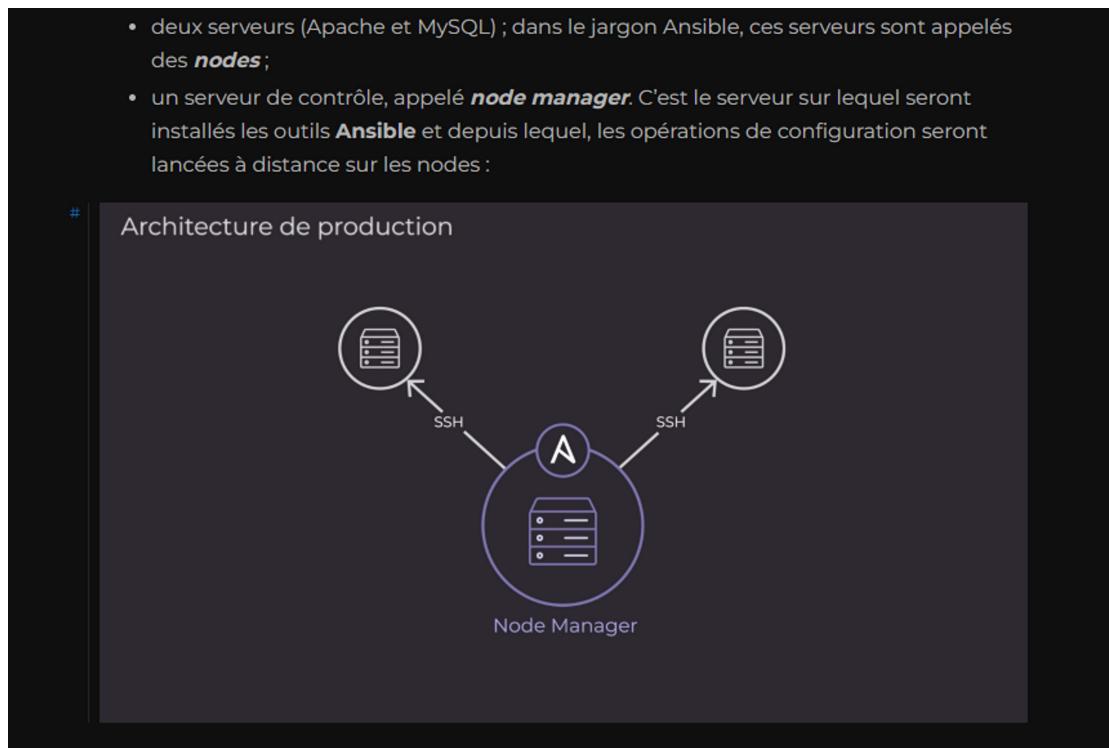
Utilisez Ansible pour automatiser vos tâches de configuration

dimanche 2 octobre 2022 10:04

Lien du cours : <https://openclassrooms.com/fr/courses/2035796-utilisez-ansible-pour-automatiser-vos-taches-de-configuration>

Travaux Pratique

Pour réaliser ce cours, nous allons devoir monter cette infrastructure sous Linux :



- node manager : 192.168.1.80 (Debian 11)
- serveur 1 : 192.168.1.81 (Debian 11)
- serveur 2 : 192.168.1.82 (Debian 11)

Pour ce TP, je vais devoir créer mes VM en effectuant un **PONT/BRIDGE**

Mode d'accès	Communication				
	VM à VM	VM vers hôte	Hôte vers VM	VM vers LAN	LAN vers VM
NAT	-	+	Redirection de port	+	Redirection de port
Réseau NAT	+	+	Redirection de port	+	Redirection de port
Pont / Bridge	+	+	+	+	+
Host-Only	+	+	+	-	-
Réseau interne	+	-	-	-	-
Aucune connexion	-	-	-	-	-

Log des machines :

Mot de passe session Vel : desirless

Mot de passe root : formation

Suite à la création de la VM, il a 3 étapes à effectuer :

- Installation Nano
- Changement adresse IP

```
root@nodeManager:~# apt get-install nano
E: L'opération get-install n'est pas valable
root@nodeManager:~# apt install nano
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
nano est déjà la version la plus récente (5.4-2+deb11u1).
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
root@nodeManager:~# apt update
Atteint :1 http://security.debian.org/debian-security bullseye-security InRelease
Atteint :2 http://ftp.u-picardie.fr/debian bullseye InRelease
Atteint :3 http://ftp.u-picardie.fr/debian bullseye-updates InRelease
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Tous les paquets sont à jour.
root@nodeManager:~# apt upgrade
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Calcul de la mise à jour... Fait
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
root@nodeManager:~# _
```

```
1 apt update
2 apt get-install nano
3 apt install nano
4 apt update
5 apt upgrade
```

Voici les différentes informations à rentrer pour la modification d'adresse IP :

- Adresse IP : 192.168.1.10 (11 et 12 pour les deux autres)
- Netmask : 255.255.255.0
- Gateway : 192.162.1.254
- DNS : 192.168.1.254

```
root@nodeManager:~# ip route show
default via 192.162.122.1 dev enp0s3
169.254.0.0/16 dev enp0s3 scope link metric 1000
192.162.122.0/24 dev enp0s3 proto kernel scope link src 192.162.122.5
```

Image avant modification :

```
GNU nano 5.4                               /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug enp0s3
iface enp0s3 inet dhcp
```

Après modification :

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug enp0s3
iface enp0s3 inet static
    address 192.168.122.10/24
    gateway 192.168.122.1
    dns-nameservers 192.168.1.254
```

Voici la commande à exécuter (adaptez le nom de l'interface réseau) :

```
sudo systemctl restart networking.service ; sudo ifup enp0s3
```

Faire ensuite différents test de ping :

```
root@nodeManager:~# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=119 time=1045 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=119 time=14.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=119 time=7.83 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=119 time=9.88 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3034ms
rtt min/avg/max/mdev = 7.828/269.420/1045.394/448.015 ms, pipe 2
root@nodeManager:~# ping amazon.com
PING amazon.com (54.239.28.85) 56(84) bytes of data.
64 bytes from 54.239.28.85 (54.239.28.85): icmp_seq=1 ttl=232 time=94.5 ms
64 bytes from 54.239.28.85 (54.239.28.85): icmp_seq=2 ttl=232 time=91.3 ms
64 bytes from 54.239.28.85 (54.239.28.85): icmp_seq=3 ttl=232 time=93.3 ms
^C
--- amazon.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 91.276/93.002/94.464/1.314 ms
```

<https://www.it-connect.fr/comment-configurer-une-adresse-ip-fixe-sur-debian-11/>

Création clé ssh :

```
aresv@MSI MINGW64 ~/ssh
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/aresv/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/aresv/.ssh/id_rsa
Your public key has been saved in /c/Users/aresv/.ssh/id_rsa.pub
```

```
The key fingerprint is:
SHA256:0P9Dh8GZeKgBbyWd/fHtXB1mNuhEHZQ1kDyr9MZGyj8 aresv@MSI
The key's randomart image is:
+---[RSA 3072]---+
| . . . ooo==o |
| + oo+.0o**+ |
| . = o O.*+= |
| o + o * . = |
| S + O .o. |
| * * o |
| * |
| E |
+---[SHA256]---+
```

Clé SSH : SHA256:0P9Dh8GZeKgBbyWd/fHtXB1mNuhEHZQ1kDyr9MZGyj8 aresv@MSI

Identifiez ce que vous pouvez automatiser



Avant de commencer, voici un peu de contexte pour comprendre quel rôle vous aurez dans ce cours. Ce rôle nous permettra de rendre concret chacun des concepts que nous aborderons :



Praxis est une société coopérative et participative (SCOP) qui développe une plateforme permettant de cartographier les cavistes qui vendent du vin totalement naturel.

Quel sera votre rôle ?

Vous êtes **ingénieur DevOps** chez Praxis. Parmi les salariés de l'entreprise, il y a plusieurs **experts DevOps** qui ont besoin de partager de l'information et de faire circuler les bonnes pratiques de développement au sein de l'entreprise.

Comme vous êtes le petit nouveau, une décision collégiale vous a désigné comme la personne qui allait mettre en place un wiki pour répondre à ce besoin !

Une recherche rapide sur le web vous permet de comprendre qu'**un wiki est un gestionnaire de contenu orienté rédaction collaborative et de partage d'informations structurées**. Ça y est, vous tenez votre première mission !

Démarrez votre première mission

Mais bien évidemment, de nombreuses questions vous viennent à l'esprit :

- Quel wiki installer ?
- Comment installer un wiki ?
- **Comment automatiser le déploiement de ce wiki, et avec quel outil ?**

La méthodologie à suivre pour répondre à toutes ces questions est de se concentrer sur des critères prioritaires qui sont à définir en fonction du contexte de l'entreprise.

Dans votre cas, vous faites le choix d'**installer MediaWiki**, car c'est une solution **open source** éprouvée et modulaire. Ce qui **colle parfaitement avec le contexte de l'entreprise**.

Étudiez l'installation de MediaWiki

La meilleure solution pour comprendre comment installer MediaWiki est de se rendre sur le site de MediaWiki à la recherche du guide d'installation : https://www.mediawiki.org/wiki/Manual:Installation_guide/fr.

En synthèse, vous résumez l'installation de MediaWiki avec les étapes suivantes :

- **Installer un serveur web** pour servir les pages à un navigateur web.
- **Installer PHP** pour exécuter le logiciel.
- **Installer une base de données** pour stocker les pages.
- **Télécharger les fichiers sources de MediaWiki** et les mettre sur le serveur web.
- **Configurer le serveur web pour pointer vers l'URL MediaWiki**.
- **Finaliser l'installation de MediaWiki via le script d'installation**.

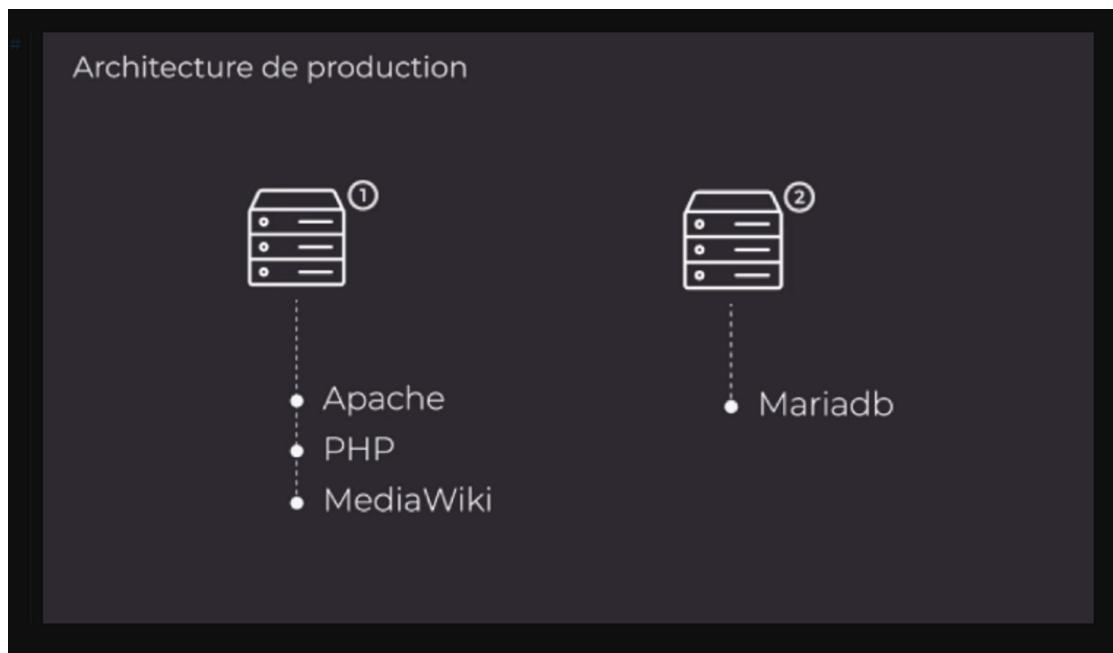
Élaborez l'architecture technique de MediaWiki

Maintenant que vous avez la bonne recette pour installer MediaWiki, vous allez élaborer l'architecture technique de MediaWiki.

Pour être conforme aux critères de départ, vous avez besoin de mettre en place une **architecture modulaire**. Vous faites donc le choix d'installer un **serveur Linux (Debian)** avec un **service web (Apache, PHP)** et un **serveur Linux (Debian) avec une base de données (MariaDB)**. Vous utilisez Centos, Apache et MariaDB, car ces

solutions sont compatibles avec l'installation de MediaWiki, et elles sont déjà utilisées dans l'entreprise.

Voici le schéma de l'architecture :



Pourquoi séparer les services sur deux serveurs, alors qu'un serveur aurait pu suffire ?

Eh bien, dans une optique de **scalabilité**, il est toujours intéressant de **séparer les services** et d'**intégrer une architecture modulaire dès la conception**. Ainsi, il sera **très facile pour faire évoluer l'infrastructure**, d'**ajouter un serveur supplémentaire**. Par exemple, si un seul serveur web n'est pas suffisant pour absorber tout le trafic web, alors il sera possible d'ajouter un serveur web supplémentaire. Avec **Ansible**, c'est extrêmement simple.

Préparez l'automatisation du déploiement

Tout est calé ! Il faut maintenant se lancer et installer MediaWiki.

Mais, **pourquoi faire une installation manuelle alors que vous pourriez scripter le déploiement et l'automatiser** ? Vous êtes un pro de l'automatisation, après tout !

C'est d'ailleurs le principe de l'**Infrastructure-As-Code** ; c'est un concept qui permet de **gérer les tâches d'administration automatiquement**, via du code, à l'aide de **fichiers de définition** plutôt que faire des traitements manuels. C'est très populaire en ce moment, c'est donc un bon moyen de s'y mettre.

Ainsi, vous pourrez réutiliser ces scripts pour un autre déploiement ; vous gagnerez du temps et vous pourrez également partager les scripts avec vos collègues.

Choisissez votre outil d'automatisation

Avec quel outil allez-vous automatiser le déploiement ?

Il faut que vous trouviez un **outil de gestion de configuration**. Il en existe plusieurs, mais les plus connus et les plus utilisés sont **Chef**, **Puppet** et **Ansible**.

Ils ont tous des avantages et des inconvénients ; le choix doit porter sur :

- **le contexte de l'environnement technique** (un outil est déjà utilisé en interne, le nombre de serveurs à

gérer est important...) ;

- **les connaissances internes des équipes techniques** : Chef et Puppet utilisent Ruby, alors que Ansible utilise Python. Ansible est simple à utiliser ; les autres outils, un peu moins, le temps d'appropriation est donc un peu plus long ;
- **le besoin** : est-ce que vous voulez faire de la **conformité**, du **déploiement**, de la **performance**, de la **simplicité**, de l'**intégration**, de la **sécurité**... ?

Comme il n'y a pas encore d'outil de gestion de configuration dans l'entreprise, vous choisissez Ansible pour sa **simplicité de prise en main** pour vous et les équipes.

En effet, vous connaissez Python, vous avez pour habitude de privilégier la simplicité, et vous ne voulez pas que l'intégration de cet outil soit une contrainte sur les serveurs. Vous imaginez même proposer d'utiliser Ansible pour la gestion de l'ensemble de l'infrastructure de l'entreprise.

Qu'allez-vous automatiser ?

Vous vous concentrez sur l'installation de MediaWiki, vous laissez à l'adminsyst en chef le soin de mettre en place l'infrastructure des serveurs ; c'est-à-dire **installer Linux (Debian)** et **mettre en place le réseau entre les deux serveurs**.

L'adminsyst vous donnera alors un compte root sur les 2 serveurs et leurs adresses IP.

Attention, pour la simplicité de ce cours, il sera nécessaire de pouvoir se connecter directement avec le compte root sur ces 2 serveurs. **Pensez donc bien à modifier la directive PermitRootLogin yes dans le fichier /etc/ssh/sshd_config.**

```
# Authentication:  
  
#LoginGraceTime 2m  
PermitRootLogin yes  
#StrictModes yes  
#MaxAuthTries 6  
#MaxSessions 10
```

Vous allez donc devoir automatiser les étapes suivantes avec Ansible :

- **Installer un serveur web Apache sur le premier serveur.**
- **Installer PHP également sur le premier serveur.**
- **Installer une base de données MariaDB sur le deuxième serveur.**
- **Télécharger les fichiers sources de MediaWiki et les mettre sur le serveur web Apache.**
- **Configurer le serveur web Apache pour pointer vers l'URL <http://http1/mediawiki>.**
- **Finaliser l'installation de MediaWiki avec un script d'installation qui est détaillé dans la documentation.**

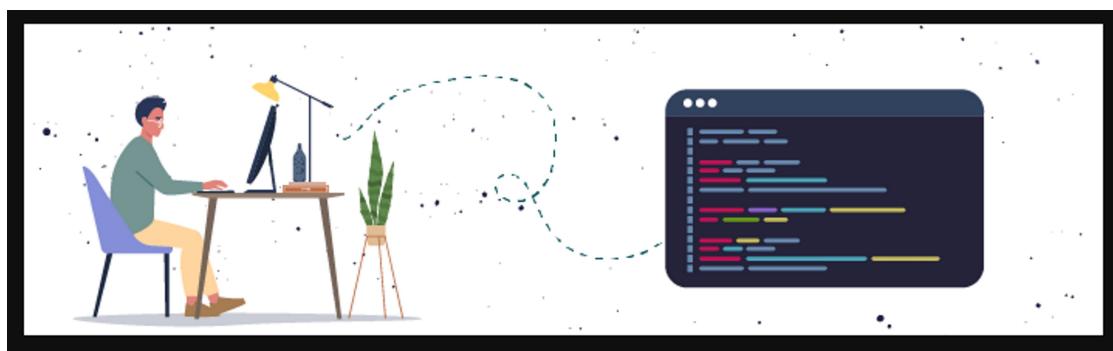
Résumé :

Dans ce chapitre, vous avez découvert ce que vous pouvez automatiser :

- vous avez fait connaissance avec votre environnement de travail ;
- vous avez été chargé d'installer un wiki ;
- vous avez retenu d'installer MediaWiki selon vos critères de choix ;
- vous avez choisi **Ansible** comme outil pour automatiser le déploiement de MediaWiki ;
- vous avez également **proposé une architecture technique et identifié les étapes d'installation qu'il convenait d'automatiser avec Ansible**.

Dans le prochain chapitre, vous allez apprendre à installer et utiliser Ansible dans le contexte de déploiement de la solution MediaWiki. Vous comprendrez comment fonctionne Ansible, et les prérequis nécessaires à un déploiement Ansible.

Installez Ansible dans votre environnement



Dans ce chapitre, vous allez faire connaissance avec l'outil de gestion Ansible, vous allez comprendre comment **déployer automatiquement** MediaWiki sur 2 serveurs avec Ansible, et vous allez **apprendre à installer Ansible dans un environnement de travail virtuel**.

Qu'est-ce qu'Ansible ?

Ansible est un **outil d'automatisation informatique** écrit en **Python**. Il peut **configurer des systèmes, déployer des logiciels et orchestrer des tâches informatiques avancées**, telles que des **déploiements continus**.

Son créateur s'appelle Michael DeHaan ; la première version de Ansible date de 2012. Depuis, Ansible s'enrichit constamment et une version majeure est proposée approximativement tous les deux mois.

Le nom Ansible est tiré d'un roman de science-fiction écrit par Ursula Le Guin, et qui désigne un moyen de communication plus rapide que la lumière.

Entretemps, Ansible a été racheté en 2015 par Red Hat ; la communauté compte plus de 3 500 contributeurs.

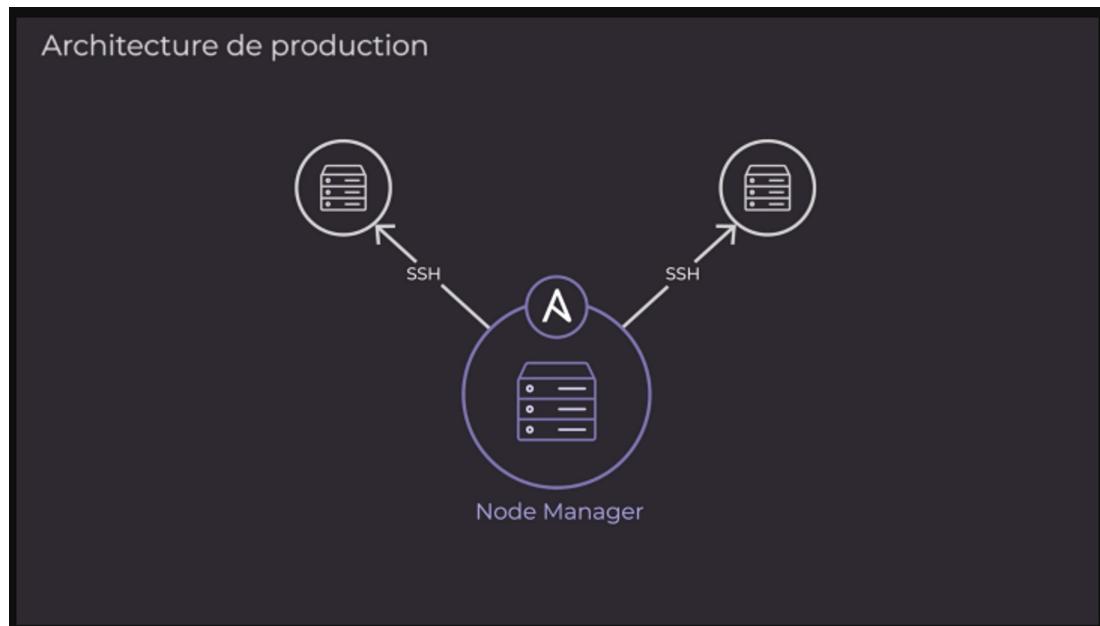
Red Hat a été racheté par IBM en 2018. Donc, **Ansible appartient désormais à IBM**.

Complétez votre architecture technique pour Ansible

Votre architecture technique est pour le moment constituée de 2 serveurs. Vous allez **ajouter un nouveau serveur** qui deviendra votre tour de contrôle Ansible.

Votre architecture ressemble donc maintenant au schéma suivant :

- deux serveurs (Apache et MySQL) ; dans le jargon Ansible, ces serveurs sont appelés des nodes
- un serveur de contrôle, appelé node manager. C'est le serveur sur lequel seront installés les outils Ansible et depuis lequel, les opérations de configuration seront lancées à distance sur les nodes



Node : Un **node** (ou **managed node**, ou **host**) est un **poste connecté au node manager en SSH**, et sur lequel Ansible viendra pousser les tâches d'automatisation. Ansible n'est pas installé sur les nodes.



Node manager

Node manager : Un **node manager**, ou **control node**, est un **poste qui contrôle les nodes grâce à sa connexion SSH**. Il dispose d'une **version Ansible** d'installé pour leur pousser les tâches d'automatisation grâce aux **commandes ansible et ansible-playbook**. Ça peut être n'importe quelle machine Linux, mais pas Windows.

Ansible est un outil **agentless**, c'est-à-dire qu'il **n'installe pas d'agent sur les nodes**. Il travaille donc **en mode push** : il **pousse les installations sur les nodes**. Pour cela, il n'utilise que les outils déjà présents sur la plupart des systèmes Linux : **SSH et Python**.

L'**inverse du mode push est le mode pull**. Par exemple, une marketplace d'applications comme le Play Store ou l'AppStore d'Apple sont des systèmes en **mode pull** : le **client (le smartphone)** tire les **applications ou les mises à jour vers lui**.

Dans le cadre de ce cours, vous utiliserez votre ordinateur comme Node Manager. Mais en pratique, il est conseillé d'avoir un serveur de référence sur lequel vous configurez un environnement d'automatisation

toujours opérationnel et sécurisé, capable de se connecter aux nodes de votre infrastructure. De plus, si vous travaillez à plusieurs sur l'infrastructure de l'entreprise avec Ansible, c'est beaucoup plus simple de contrôler les accès et les scripts depuis un point unique.

Gérez vos configurations avec le Node Manager

Vous demandez au sysadmin de vous créer en plus un **nouveau serveur Debian 11**, qui deviendra le **node manager**.

Ça tombe bien, car le sysadmin est disponible pour le faire tout de suite et en profite pour vous fournir les adresses IP de tous les serveurs et leurs comptes root ; vous allez pouvoir travailler sur le node manager :

- **node manager** : 192.168.1.80 (Debian 11)
- **serveur 1** : 192.168.1.81 (Debian 11)
- **serveur 2** : 192.168.1.82 (Debian 11).

À ce stade, vous devez avoir **3 serveurs disponibles** qui communiquent sur le sous-réseau 192.168.1.0/24 et qui sont capables de sortir et de résoudre sur Internet.

Le node manager va donc être votre tour de contrôle. Vous allez y installer **Ansible** et tous ses outils, pour déployer automatiquement MediaWiki sur les nodes.

Le node manager et les nodes peuvent avoir des systèmes d'exploitation différents. Le node manager peut être un système Debian, et les nodes peuvent être des systèmes Centos, Windows, Ubuntu ou autre. Il n'y a pas de corrélation entre le système du node manager et le système des nodes.

Donc sur le **node manager**, vous trouverez les **outils Ansible** et les **scripts d'automatisation**. Tous les scripts seront lancés depuis le **node manager**. Ce qui aura pour effet d'**exécuter des opérations de configuration à distance sur les nodes**.

Installer Ansible

Installez Ansible sur le node manager

Vous allez installer **Ansible** sur votre **node manager**. Il y a plusieurs façons d'installer Ansible :

- via les **packages logiciels sur un système Linux**
- via **pip de Python** dans un virtualenv ou pas
- via les **sources officielles (Archives ou Git)** maintenues par **Red Hat**.

Ansible peut être installé avec ces trois méthodes nativement sur des systèmes de type **UNIX (Linux ou macOS)**.

Pour Windows, il faudra passer par un **émulateur Unix de type Cygwin** pour installer Ansible.

Installez les prérequis

L'installation de **Ansible** sur le **node manager** (un serveur Debian) se fera avec la méthode **pip de Python dans un virtualenv**.

Un **virtualenv** est un **outil Python** qui permet de créer des environnements de travail virtuels isolés. **Virtualenv** crée un dossier qui contient les fichiers exécutables Python, et une copie de la bibliothèque pip.

pip est un **système de gestion de paquets** utilisé pour **installer et gérer les paquets logiciels écrits en Python**.

Avec cette méthode, **vous créez un environnement de travail virtuel cloisonné**, dans lequel vous pourrez installer la version Ansible de votre choix (basée sur le [release repository de Ansible](#)).

À chaque sortie de la dernière version de Ansible, **il y a des nouveautés et des dépréciations** ; il est donc **important de tester la compatibilité de vos scripts avant de mettre à jour les outils Ansible**. Avec **virtualenv** c'est très pratique ! Par exemple, **vous pouvez créer un virtualenv par version de Ansible**.

Sur **Debian** par exemple, **le paquet disponible pour installer Ansible n'est pas la toute dernière version (Debian priviliege la sécurité et la stabilité plutôt que la nouveauté, c'est un parti pris !)**. C'est parfois un inconvénient quand on attendait une fonctionnalité tout juste sortie mais non disponible en upgrade sur Debian. **La méthode pip permet de s'en affranchir et de disposer de la dernière version de Ansible**.

Connectez-vous sur le node manager en root :

```
vel@nodeManager:~$ su - root
Mot de passe :
root@nodeManager:~#
```

Installez le paquet **python-virtualenv**, ce qui permettra de créer des environnement de travail virtuel : **virtualenv**.

Vous profiterez d'installer le paquet **sshpass** qui servira ultérieurement pour se connecter en SSH avec Ansible.

Il faut utiliser la commande *pip install virtualenv* pour installer *virtualenv*

```
root@nodeManager:~# pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.16.5-py3-none-any.whl (8.8 MB)
    |████████| 8.8 MB 4.4 MB/s
Collecting distlib<1,>=0.3.5
  Downloading distlib-0.3.6-py2.py3-none-any.whl (468 kB)
    |████████| 468 kB 64.9 MB/s
Collecting filelock<4,>=3.4.1
  Downloading filelock-3.8.0-py3-none-any.whl (10 kB)
Collecting platformdirs<3,>=2.4
  Downloading platformdirs-2.5.2-py3-none-any.whl (14 kB)
Installing collected packages: platformdirs, filelock, distlib, virtualenv
Successfully installed distlib-0.3.6 filelock-3.8.0 platformdirs-2.5.2 virtualenv-20.16.5
root@nodeManager:~#
```

Créez un simple utilisateur

Pour ne pas travailler en root sur le node manager (**ce n'est vraiment pas recommandé, le compte root peut tout faire sans aucune limite, une erreur est vite arrivée !**), vous allez donc créer un simple utilisateur **user-ansible** :

```
root@nodeManager:~# adduser user-ansible
Ajout de l'utilisateur « user-ansible » ...
Ajout du nouveau groupe « user-ansible » (1001) ...
Ajout du nouvel utilisateur « user-ansible » (1001) avec le groupe « user-ansible » ...
Création du répertoire personnel « /home/user-ansible »...
Copie des fichiers depuis « /etc/skel »...
Nouveau mot de passe :
Retapez le nouveau mot de passe :
passwd: password updated successfully
Changing the user information for user-ansible
Enter the new value, or press ENTER for the default
    Full Name []: user-ansible
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Cette information est-elle correcte ? [0/n]0
root@nodeManager:~#
```

Maintenant que l'utilisateur est créé, vous pouvez l'utiliser avec la commande suivante :

```
root@nodeManager:~# su - user-ansible
user-ansible@nodeManager:~$
```

Vous travaillez maintenant avec l'utilisateur **user-ansible** sur le **node-manager**.

La ligne de commande commence par un # quand vous êtes en root, et par un \$ quand vous êtes en simple utilisateur.

Créez votre environnement de travail virtuel

Comme indiqué plus haut, **vous utiliserez un environnement de travail virtuel pour cloisonner l'installation et l'exécution d'Ansible**. Ceci permettra de gérer les dépendances avec la version de Python et d'installer une version particulière de Ansible.

Vous allez installer la dernière version en date de Ansible.

Vous créez alors un environnement de travail virtuel nommé ansible.

Le nom est arbitraire. Vous pouvez mettre ce que vous voulez, mais donner du sens à ce que vous faites, c'est mieux.

Sur le **node manager**, lancez la commande suivante :

```
user-ansible@nodeManager:~$ virtualenv ansible
created virtual environment CPython3.9.2.final.0-64 in 573ms
  creator CPython3Posix(dest=/home/user-ansible/ansible, clear=False, no_vcs_ignore=False,
global=False)
    seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy,
app_data_dir=/home/user-ansible/.local/share/virtualenv)
      added seed packages: pip==22.2.2, setuptools==65.3.0, wheel==0.37.1
      activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivat
or,PythonActivator
```

Vous utilisez la commande **virtualenv** pour **créer l'environnement ansible** dans lequel les **outils**, les **ressources** et le **gestionnaire de paquets** sont installés.

Pour activer l'environnement virtuel, il faut activer la source :

```
user-ansible@nodeManager:~$ source ansible/bin/activate  
(ansible) user-ansible@nodeManager:~$
```

Le prompt a changé au profit de **(ansible)** user-ansible@node-manager:~\$. Ce qui signifie que vous êtes dans l'environnement de travail virtuel **Ansible**.

Installez Ansible dans votre environnement virtuel

Installez maintenant **Ansible** avec **pip** avec la commande suivante :

- Pip install ansible

```
(ansible) user-ansible@nodeManager:~$ pip install ansible  
  
Collecting jinja2>=3.0.0  
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)  
    ━━━━━━━━━━━━━━━━ 133.1/133.1 kB 25.1 MB/s eta 0:00:00  
Collecting cryptography  
  Downloading cryptography-38.0.1-cp36-abi3-manylinux_2_28_x86_64.whl (4.2 MB)  
    ━━━━━━━━━━━━━━ 4.2/4.2 MB 49.1 MB/s eta 0:00:00  
Collecting PyYAML>=5.1  
  Downloading PyYAML-6.0-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x8  
6_64.manylinux2010_x86_64.whl (661 kB)  
    ━━━━━━━━━━━━━━ 661.8/661.8 kB 70.5 MB/s eta 0:00:00  
Collecting packaging  
  Downloading packaging-21.3-py3-none-any.whl (40 kB)  
    ━━━━━━━━━━━━ 40.8/40.8 kB 7.6 MB/s eta 0:00:00  
Collecting MarkupSafe>=2.0  
  Downloading MarkupSafe-2.1.1-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (25  
kB)  
Collecting cffi>=1.12  
  Downloading cffi-1.15.1-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (441 kB)  
    ━━━━━━━━━━━━ 441.2/441.2 kB 11.6 MB/s eta 0:00:00  
Collecting pyparsing!=3.0.5,>=2.0.2  
  Downloading pyparsing-3.0.9-py3-none-any.whl (98 kB)  
    ━━━━━━━━━━ 98.3/98.3 kB 12.3 MB/s eta 0:00:00  
Collecting pycparser  
  Downloading pycparser-2.21-py2.py3-none-any.whl (118 kB)  
    ━━━━━━━━━━ 118.7/118.7 kB 25.7 MB/s eta 0:00:00  
Installing collected packages: resolvelib, PyYAML, pyparsing, pycparser, MarkupSafe, packag  
ing, jinja2, cffi, cryptography, ansible-core, ansible  
Successfully installed MarkupSafe-2.1.1 PyYAML-6.0 ansible-6.4.0 ansible-core-2.13.4 cffi-1  
.15.1 cryptography-38.0.1 jinja2-3.1.2 packaging-21.3 pycparser-2.21 pyparsing-3.0.9 resolv  
elib-0.8.1  
(ansible) user-ansible@nodeManager:~$ exit()
```

Vérifiez la version de Ansible avec la commande suivante : **ansible --version**

```
(ansible) user-ansible@nodeManager:~$ ansible --version  
ansible [core 2.13.4]  
  config_file: None
```

Dans le répertoire bin de votre environnement virtuel, vous pouvez constater que 11 outils Ansible sont installés :

```
(ansible) user-ansible@nodeManager:~$ ls ansible/bin* -l
total 108
-rw-r--r-- 1 user-ansible user-ansible 2145 3 oct. 13:48 activate
-rw-r--r-- 1 user-ansible user-ansible 1437 3 oct. 13:48 activate.csh
-rw-r--r-- 1 user-ansible user-ansible 3022 3 oct. 13:48 activate.fish
-rw-r--r-- 1 user-ansible user-ansible 2563 3 oct. 13:48 activate.nu
-rw-r--r-- 1 user-ansible user-ansible 1754 3 oct. 13:48 activate.ps1
-rw-r--r-- 1 user-ansible user-ansible 1175 3 oct. 13:48 activate_this.py
-rwxr-xr-x 1 user-ansible user-ansible 237 3 oct. 13:52 ansible
-rwxr-xr-x 1 user-ansible user-ansible 257 3 oct. 13:52 ansible-community
-rwxr-xr-x 1 user-ansible user-ansible 238 3 oct. 13:52 ansible-config
-rwxr-xr-x 1 user-ansible user-ansible 267 3 oct. 13:52 ansible-connection
-rwxr-xr-x 1 user-ansible user-ansible 239 3 oct. 13:52 ansible-console
-rwxr-xr-x 1 user-ansible user-ansible 235 3 oct. 13:52 ansible-doc
-rwxr-xr-x 1 user-ansible user-ansible 238 3 oct. 13:52 ansible-galaxy
-rwxr-xr-x 1 user-ansible user-ansible 241 3 oct. 13:52 ansible-inventory
-rwxr-xr-x 1 user-ansible user-ansible 240 3 oct. 13:52 ansible-playbook
-rwxr-xr-x 1 user-ansible user-ansible 236 3 oct. 13:52 ansible-pull
-rwxr-xr-x 1 user-ansible user-ansible 1721 3 oct. 13:52 ansible-test
-rwxr-xr-x 1 user-ansible user-ansible 237 3 oct. 13:52 ansible-vault
```

Regardons de plus près trois d'entre eux :

- **ansible** : cette commande permet de lancer des actions Ansible en mode ad-hoc (en ligne de commande) ;
- **ansible-config** : cette commande permet de manager la configuration de Ansible :

si vous lancez la commande **\$ ansible-config list**, vous allez lister la **configuration de Ansible**. Toutes ces variables sont contenues dans **./lib/pythonX.Y/site-packages/ansible/constants.py** ;

- **ansible-doc** : cette commande permet d'obtenir de l'aide pour utiliser Ansible ; la documentation est très bien faite, c'est plutôt pratique pour se guider quand on commence, surtout que vous pouvez y trouver des exemples concrets.

Pour sortir du **virtualenv** : **deactivate**

Pour revenir dans le **virtualenv** : **source ansible/bin/activate**

```
(ansible) user-ansible@nodeManager:~$ deactivate
user-ansible@nodeManager:~$ []
```

Résumé

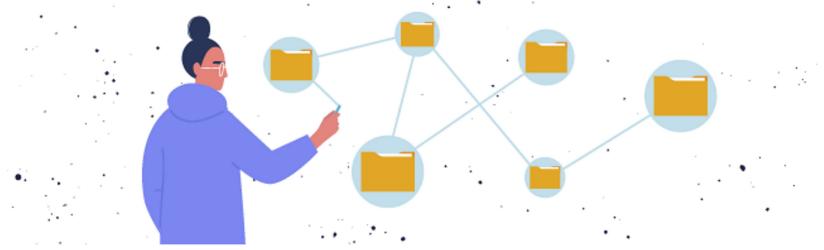
Dans ce chapitre, vous avez découvert l'automatisation avec Ansible, et plus particulièrement :

- la genèse de Ansible ;
- l'architecture Ansible avec le node manager et les nodes ;
- comment installer Ansible dans un environnement de travail virtuel ;

trois outils fondamentaux qui composent une partie des outils Ansible.

Dans le prochain chapitre, vous allez préparer la communication avec les nodes. Il s'agira de mettre en place les prérequis nécessaires pour établir la communication entre le node manager et les nodes.

Préparez la communication avec les nodes



Dans le chapitre précédent, vous avez fait connaissance avec Ansible, vous avez préparé votre architecture pour déployer MediaWiki sur 2 serveurs à l'aide d'un node manager, vous avez installé les outils Ansible sur le node manager et regardé d'un peu plus près les 3 outils Ansible dont vous allez avoir besoin par la suite.

Dans ce chapitre, vous allez préparer la communication avec les nodes. C'est-à-dire que vous allez installer les prérequis pour que le node manager puisse communiquer avec les nodes et leur envoyer les commandes Ansible.

Préparez la communication

Vous allez donc **déployer automatiquement MediaWiki sur les 2 nodes**, en utilisant **Ansible et les scripts d'automatisation** depuis le **node manager**.

Ansible travaille avec des outils déjà très utilisés sur les systèmes Linux. En effet, Ansible a uniquement besoin de SSH et de Python pour fonctionner. Il faut donc au minimum que ces 2 outils soient installés et fonctionnels sur le node manager et les nodes.

Pour établir la communication et avoir le droit de lancer des commandes à distance, vous avez besoin de :

- **Définir un inventaire des nodes.**
- **Vérifier la connexion SSH entre le node manager et les nodes.**
- **Lancer un ping avec Ansible sur les nodes.**
- **Vérifier que Python est installé sur les nodes.**
- **Créer un simple utilisateur sur les nodes.**
- **Attribuer les droits sudo à cet utilisateur.**
- **Créer une paire de clés SSH pour cet utilisateur.**
- **Copier la clé publique SSH sur les nodes.**

Définissez un inventaire des nodes

Comme vous n'avez pas accès au **DNS** de l'entreprise, vous décidez de configurer la résolution de nom via le fichier **/etc/hosts** sur le **node manager**.

Dans l'absolu, il faut utiliser **les noms de machine plutôt que les adresses IP**. Les noms des **nodes** sont toujours à enregistrer dans un **DNS**. Pour l'exercice qui se concentre sur la pratique de **Ansible**, vous prenez un raccourci en utilisant le fichier **/etc/hosts**. Ce fichier permet de mettre en place **des correspondances entre les noms des serveurs et leurs adresses IP**.

Connectez-vous sur le **node manager en root** :

```
○ ve1@nodeManager:~$ su -
Mot de passe :
root@nodeManager:~# 
```

Ajoutez dans le fichier **/etc/hosts** sur le **node manager** l'enregistrement des **2 nodes** :

```
root@nodeManager:~# cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      nodeManager

#node srvApache

192.168.1.81    srvApache

#node srvSql

192.168.1.82    srvSql
```

Cette action vous permettra d'utiliser les noms **srvApache** et **srvSql** pour communiquer avec les nodes.

Créez le fichier inventaire Ansible

À ce stade, vous allez commencer à utiliser les commandes **Ansible** pour faire la suite des actions.

Pour fonctionner, Ansible a besoin d'un **fichier inventaire**. Ce fichier contient la liste des **nodes**. Vous allez donc enregistrer le nom des **nodes** dans ce fichier.

Placez-vous dans votre environnement virtuel :

```
user-ansible@nodeManager:~$ source ansible/bin/activate
(ansible) user-ansible@nodeManager:~$ 
```

Puis **éditez** avec la commande **vi** (ou **nano**) le fichier **inventaire.ini** :

- **nano inventaire.ini**

Mettez dans ce fichier les 2 noms des nodes :

srvApache
srvSql

```
(ansible) user-ansible@nodeManager:~$ nano inventaire.ini
(ansible) user-ansible@nodeManager:~$ cat inventaire.ini
srvApache
srvSql
```

Le fichier inventaire est au format INI par défaut, il suit donc [la syntaxe de ce format](#).

Ansible sait travailler avec d'autres formats de données, il suffit d'activer les bons plugins. Vous trouverez [la liste des plugins d'inventaire ici](#).

Vérifiez que vous communiquez avec les nodes

Avant d'utiliser Ansible pour commencer à automatiser des tâches, **lancez une connexion SSH sur les nodes pour enregistrer la fingerprint** (l'empreinte du serveur qui doit être vérifiée pour être sûr de se connecter au bon node) **sur le node manager**. Il faut le faire au moins une fois, sinon Ansible râlera à la première connexion, car **le paquet sshpass ne sait pas gérer le fingerprint**.

```
(ansible) user-ansible@nodeManager:~$ ssh root@srvApache
The authenticity of host 'srvapache (192.168.1.81)' can't be established.
ECDSA key fingerprint is SHA256:vSKOKUpQN8DolJ0giWtOYQPW+PJpCqcsH00E3RH
TUqw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'srvapache,192.168.1.81' (ECDSA) to the list
of known hosts.
root@srvapache's password: []
```

Répondez yes et faites la même chose pour bdd1.

Utilisez la commande ansible en mode ad-hoc

Vous allez commencer à utiliser **Ansible** (en mode **ad-hoc**, c'est-à-dire avec des **commandes manuelles** plutôt que des scripts) pour mettre en place les **prérequis**. Ceci vous permettra de les **automatiser** et de les **appliquer à tous les nodes en même temps**.

Les commandes **ad-hoc** sont des **actions rapides** qui ne nécessitent pas forcément de les sauvegarder pour plus tard. Vous pouvez vous référer à [la documentation d'Ansible sur les commandes ad-hoc](#) pour approfondir

Lancez maintenant un **ping** avec **Ansible** dans votre environnement de travail virtuel avec la commande :
ansible -i inventaire.ini -m ping srvApache --user root --ask pass

```
(ansible) user-ansible@node-manager:~$ ansible -i inventaire.ini -m ping http1 --user root
SSH password:
http1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Vous venez d'utiliser la commande ansible avec plusieurs options :

- **i** : indique à Ansible l'emplacement du fichier inventaire ;
- **m** : indique à Ansible d'utiliser le module ping ;
- **http1** : indique à Ansible de faire l'action sur le node http1 ;
- **--user** : indique à Ansible d'utiliser l'utilisateur root pour se connecter au node (pas le choix pour le moment, car c'est le seul compte dont vous disposez) ;
- **--ask-pass** : indique à Ansible de demander le mot de passe SSH ;
- **http1** : indique à Ansible de lancer la commande sur le node http1.

Le retour de la commande vous indique que l'action est un succès et répond **pong** au **ping** ! Le node **http1** est bien joignable.

Ansible ne lance pas la commande ping, il lance un module qui fait la même chose que la commande ping.

Mot de passe ssh : formation

Module : Un module est un programme utilisé pour exécuter une tâche ou une commande Ansible. Chaque tâche utilise un module et un seul, qui peut prendre des arguments pour être exécuté de manière personnalisée. Ansible fournit de nombreux modules, mais vous pouvez créer le vôtre, personnalisé.

Tous les modules sont accessibles sur [la documentation d'Ansible](#) ou avec la commande **(ansible) user-ansible@node-manager:~\$ ansible-doc--list**

Vérifiez que Python est installé sur les nodes

Vérifier que sshpass est bien installé sur l'hôte

Il arrive parfois que **Python** ne soit pas installé sur le **node** ; dans ce cas, vous pouvez utiliser un module spécial : **raw**, qui permet de passer des commandes Ansible sans utiliser Python : **ansible -i inventaire.ini -m raw -a "apt install -y python3" srvApache --user root --ask-pass**

```
(ansible) user-ansible@nodeManager:~$ ansible -i inventaire.ini -m raw -a "apt install -y python3" srvApache --user root --ask-pass  
SSH password:  
srvApache | CHANGED | rc=0 >>  
Lecture des listes de paquets... Fait  
Construction de l'arbre des dépendances... Fait  
Lecture des informations d'état... Fait  
python3 est déjà la version la plus récente (3.9.2-3).  
python3 passé en « installé manuellement ».  
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.  
Shared connection to srvapache closed.
```

Vous avez utilisé l'option **-m** pour appeler le module **raw** avec l'argument “**apt install -y python3**”, qui a pour effet d'installer **Python** en **version 3** sur le **node** **srvApache**.

En réalité, les **modules Ansible** peuvent être écrits avec un autre langage que **Python**. Le module **raw**, écrit en **Shell**, en est la preuve ! Un atout supplémentaire en faveur d'Ansible.

Créez l'utilisateur user-ansible sur les nodes

Vous allez suivre les **bonnes pratiques** en **créant un simple utilisateur pour ne pas travailler directement avec le compte root**.

Il est déconseillé d'utiliser le compte **root** directement pour faire l'ensemble des opérations (bien que cela soit possible avec Ansible et nécessaire pour mettre en place les prérequis). Il faut plutôt privilégier de travailler avec un simple utilisateur et lui donner les droits **sudo**. En effet, les commandes lancées avec **sudo** se feront avec un mot de passe préalable et peuvent être limitées à certaines actions. De plus, le compte utilisateur étant nominatif, il sera beaucoup plus facile de tracer les actions dans les logs pour debugger.

Mais avant de créer un utilisateur, vous allez **générer un mot de passe chiffré** (au format reconnu par Linux) avec **Ansible** :

- **ansible localhost -i inventaire.ini -m debug -a"msg={{'passforce'|password_hash('sha512','secretsalt')}}"**

```
(ansible) user-ansible@nodeManager:~$ ansible localhost -i inventaire.ini  
-m debug -a "msg={{'passforce'|password_hash('sha512','secretsalt')}}"  
localhost | SUCCESS => {  
    "msg": "$6$secretsalt$X5YDmUgDphPxnMkByvHbNaiP4T5Uk0WjEZ9TukWKQnXmXN81jG3DcGZnNJiSz9ltgPhplH92HOR/RqgmyS.zN1"  
}
```

- \$6\$secretsalt
\$X5YDmUgDphPxnMkByvHbNaiP4T5Uk0WjEZ9TukWKQnXmXN81jG3DcGZnNJiSz9ltgPhplH92HOR/RqgmyS.zN1

Dans cette commande, nous avons utilisé le module **debug** avec l'argument **msg** pour **transformer le mot de passe "passforce" en une chaîne chiffrée avec l'algorithme sha512**. “**Secretsalt**” est ce qu'on appelle le **sel**. C'est un mot qui permet de renforcer la sécurité de l'algorithme en apportant une inconnue en plus dans le processus de cryptage du mot de passe.

L'option “**localhost**” a permis d'indiquer à Ansible de lancer la commande sur **localhost** (en **local** sur le **node manager**).

Vous allez maintenant pouvoir créer l'utilisateur **user-ansible** avec un **mot de passe chiffré**, grâce à la commande suivante :

- ansible -i inventaire.ini -m user -a 'name=user-ansible password=\$6\$secretsalt\$X5YDmUgDphPxnMkByvHbNaiP4T5Uk0WjEZ9TukWKQnXmXN81jG3DcGZnNjSz9ltgPhplH92HOR/RqgmyS.zN1' --user root --ask-pass all

```
(ansible) user-ansible@nodeManager:~$ ansible -i inventaire.ini -m user -a 'name=user-ansible  
password=$6$secretsalt$X5YDmUgDphPxnMkByvHbNaiP4T5Uk0WjEZ9TukWKQnXmXN81jG3DcGZnNjSz9ltgPhplH92HOR/RqgmyS.zN1' --user root --  
ask-pass all  
SSH password:  
srvApache | CHANGED => {  
    "ansible_facts": {  
        "discovered_interpreter_python": "/usr/bin/python3"  
    },  
    "changed": true,  
    "comment": "",  
    "create_home": true,  
    "group": 1001,  
    "home": "/home/user-ansible",  
    "name": "user-ansible",  
    "password": "NOT_LOGGING_PASSWORD",  
    "shell": "/bin/sh",  
    "state": "present",  
    "system": false,  
    "uid": 1001  
}  
srvSql | CHANGED => {  
    "ansible_facts": {  
        "discovered_interpreter_python": "/usr/bin/python3"  
    },  
    "changed": true,  
    "comment": "",  
    "create_home": true,  
    "group": 1001,  
    "home": "/home/user-ansible",  
    "name": "user-ansible",  
    "password": "NOT_LOGGING_PASSWORD",  
    "shell": "/bin/sh",  
    "state": "present",  
    "system": false,  
    "uid": 1001  
}
```

Vous avez utilisé le **module user** avec les arguments **name** et **password**, et vous avez demandé à Ansible de lancer la commande sur **all** (sur tous les nodes présents dans le fichier inventaire).

Donnez les droits sudo à user-ansible

Connectez-vous au node **srvApache**, installez le **package sudo** (`apt get install sudo`), et regardez le fichier **/etc/sudoers** : il contient les **configurations de sudo** :

Vous pouvez voir la ligne suivante :

Le groupe **sudo** a les droits **sudo**. Vous allez donc ajouter l'utilisateur **user-ansible** dans le groupe **sudo** sur tous les **nodes** :

```
# Allow members of group sudo to execute any command  
%sudo    ALL=(ALL:ALL) ALL
```

Le groupe **sudo** a les droits **sudo**. Vous allez donc ajouter l'utilisateur **user-ansible** dans le groupe **sudo** sur tous les **nodes** :

- **ansible -i inventaire.ini -m user -a 'name=user-ansible groups=sudo append=yes' --user root --ask-pass all**

```
(ansible) user-ansible@nodeManager:~$ ansible -i inventaire.ini -m user -a 'name=user-ansible groups=sudo append=yes' --user root --ask-pass all
SSH password:
srvApache | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "append": true,
    "changed": true,
    "comment": "",
    "group": 1001,
    "groups": "sudo",
    "home": "/home/user-ansible",
    "move_home": false,
    "name": "user-ansible",
    "shell": "/bin/sh",
    "state": "present",
    "uid": 1001
}
srvSql | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "append": true,
    "changed": true,
    "comment": "",
    "group": 1001,
    "groups": "sudo",
    "home": "/home/user-ansible",
    "move_home": false,
    "name": "user-ansible",
    "shell": "/bin/sh",
    "state": "present",
    "uid": 1001
}
```

Un moyen de vérifier que **user-ansible** a bien les droits **sudo** est de relancer la commande précédente, mais en ajoutant de nouvelles options :

- **ansible -i inventaire.ini -m user -a 'name=user-ansible groups=sudo append=yes' --user user-ansible --ask-pass --become --ask-become-pass all**

```
(ansible) user-ansible@node-manager:~$ ansible -i inventory -m user -a "name=user-ansible state=present groups=sudo"
SSH password:
BECOME password[defaults to SSH password]:
bdd1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "append": true,
    "changed": false,
    "comment": "",
    "group": 1001,
    "groups": "sudo",
    "home": "/home/user-ansible",
    "move_home": false,
    "name": "user-ansible",
    "shell": "/bin/sh",
    "state": "present",
    "uid": 1001
}
http1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "append": true,
    "changed": false,
    "comment": "",
    "group": 1001,
    "groups": "sudo",
    "home": "/home/user-ansible",
    "move_home": false,
    "name": "user-ansible",
    "shell": "/bin/sh",
    "state": "present",
    "uid": 1001
}
```

La commande s'est bien passée, car le retour est un SUCCESS ! Vous avez les priviléges sudo.

Ici, vous avez utilisé le **module user** pour ajouter **user-ansible** dans le groupe **wheel** et vous avez utilisé deux nouvelles options :

- **--become** : Ansible vous permet de "devenir" un autre utilisateur en utilisant sudo
- **--ask-become-pass** : Ansible demande le mot de passe sudo (qui est le même que le mot de passe de user-ansible).

Dorénavant, nous n'utiliserons plus root, mais l'utilisateur **user-ansible** en mode **sudo** pour passer les commandes Ansible.

Créez les clés SSH

Pour se connecter en **SSH**, il est recommandé d'utiliser une **paire de clés** plutôt que d'utiliser un mot de passe.

La communication **SSH** est établie sur la base de **clés SSH** ; cette pratique est **conseillée**, car elle permet un niveau d'authentification beaucoup plus sûr que l'authentification par mot de passe.

Vous allez maintenant créer une **paire de clés SSH** de type **ecdsa** pour l'utilisateur **user-ansible**.

Elliptic Curve Digital Signature Algorithm (ECDSA) est un algorithme de signature numérique qu'il est conseillé d'utiliser aujourd'hui pour avoir un niveau de sécurité satisfaisant.

Commencez par vous remettre en utilisateur **user-ansible** avec la commande suivante si c'est nécessaire :

Et lancez la création des clés avec la commande suivante :

- **ssh-keygen -t ecdsa**

```
(ansible) user-ansible@nodeManager:~$ ssh-keygen -t ecdsa
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/user-ansible/.ssh/id_ecdsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user-ansible/.ssh/id_ecdsa
Your public key has been saved in /home/user-ansible/.ssh/id_ecdsa.pub
The key fingerprint is:
SHA256:3/1Z8+rG4rwer86bcWCMEl9VEFjaSasn1fQ/C2yTvK0 user-ansible@nodeManager
The key's randomart image is:
+---[ECDSA 256]---+
|          +B+o|
|          .* +.|
|          . + = o|
|          o +o+. .|
|          S o =B ..|
|          o o.o= o|
|          . +oo+.|
|          o.*+.=|
|          +@E+oo|
+---[SHA256]---
```

Vous pouvez laisser tout par défaut, sans rien changer. Il n'est pas nécessaire de mettre de **passphrase**.

La **passphrase** est utilisée pour renforcer la connexion SSH par un mot de passe. Dans notre cas, vous ne l'utilisez pas, ce qui vous fera gagner du temps lors des exercices pratiques.

Ajoutez la clé publique de l'utilisateur user-ansible sur les nodes

Vous allez utiliser le module **authorized_key** pour enregistrer la clé publique de l'utilisateur **user-ansible** sur tous les **nodes**. Ainsi, **vous pourrez vous connecter aux nodes sans saisir de mot de passe SSH**.

- **ansible -i inventaire.ini -m authorized_key -a 'user=user-ansible state=present key="{{ lookup("file", "/home/user-ansible/.ssh/id_ecdsa.pub") }}" --user user-ansible --ask-pass --become --ask-become-pass all**

Vous avez utilisé de nouvelles options :

- **-m authorized_key** : module authorized_key (ajoute ou supprime les clés SSH pour des utilisateurs) ;
- **user=user-ansible** : l'utilisateur concerné est user-ansible ;
- **state=present** : indique d'ajouter le fichier ;
- **key="{{ lookup("file", "/home/user-ansible/.ssh/id_ecdsa.pub") }}"** : utilise la commande lookup pour rechercher le fichier concerné.

Relancez la commande mais cette fois sans **--ask-pass** (**demande le mot de passe SSH**) :

- **ansible -i inventaire.ini -m authorized_key -a 'user=user-ansible state=present key="{{lookup("file", "/home/user-ansible/.ssh/id_ecdsa.pub") }}" --user user-ansible --become --ask-become-pass all**

```
(ansible) user-ansible@node-manager:~$ ansible -i inventaire.ini -m authorized_key -a 'use_become=True'
BECOME password:
http1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "comment": null,
    "exclusive": false,
    "follow": false,
    "key": "ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIbmlzdHAyNTYAAABBBA0S",
    "key_options": null,
    "keyfile": "/home/user-ansible/.ssh/authorized_keys",
    "manage_dir": true,
    "path": null,
    "state": "present",
    "user": "user-ansible",
    "validate_certs": true,
}
bdd1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "comment": null,
    "exclusive": false,
    "follow": false,
    "key": "ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIbmlzdHAyNTYAAABBBA0S",
    "key_options": null,
    "keyfile": "/home/user-ansible/.ssh/authorized_keys",
    "manage_dir": true,
    "path": null,
    "state": "present",
    "user": "user-ansible",
    "validate_certs": true,
}
```

Un seul mot de passe vous a été demandé, celui de sudo uniquement.

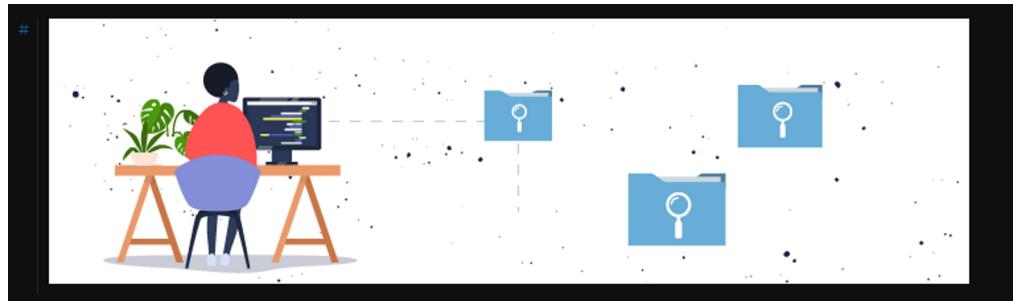
Voilà, vous pouvez exécuter avec Ansible des commandes distantes sur les nodes avec le privilège sudo. À vous l'automatisation sans limite !

Résumé :

Dans ce chapitre, vous avez préparé la communication avec les nodes ; vous avez notamment :

- créé un fichier inventaire et ajouté les nodes dans ce fichier ;
- utilisé la commande ansible en mode ad-hoc pour vérifier la présence de Python sur les nodes et lancer un ping ;
- créé un utilisateur user-ansible sur les nodes ;
- créé des clés SSH pour l'utilisateur user-ansible ;
- donné les privilèges sudo à user-ansible pour pouvoir exécuter des commandes en tant que super utilisateur sur les nodes ;
- enregistré la clé publique de user-ansible sur les nodes pour sécuriser les communications.

Dans le prochain chapitre, vous allez dérouler le déploiement d'une application avec Ansible.



Dans la première partie, vous avez pris connaissance du contexte dans lequel vous allez travailler, et vous avez mis en place un environnement technique favorable pour **automatiser le déploiement de MediaWiki à l'aide d'Ansible**.

Dans ce chapitre, vous allez **transposer et organiser les opérations nécessaires à l'installation de MediaWiki, à l'aide des rôles Ansible**.

Le **node manager** et le **node** sont maintenant opérationnels. Votre travail va être d'**installer le gestionnaire de wiki interne MediaWiki sur 2 serveurs**, afin de le rendre utilisable par tous les salariés de l'entreprise. Pour cela, nous allons avoir besoin d'**organiser nos opérations**, afin de les **exécuter séquentiellement avec Ansible**.



Rôle : Un rôle est une **structure arborescente constituée de répertoires et de fichiers de configuration YAML**, qui vont avoir pour **fonction d'installer tel ou tel système**. Les rôles peuvent être imbriqués et interdépendants les uns des autres. Un rôle est donc un **ensemble de fichiers organisés dans une structure arborescente**.

À quoi ça sert ?

Le but des rôles est de pouvoir **agglomérer des opérations cohérentes** (dans les fichiers YAML), afin de pouvoir les réutiliser de façon **modulaire**. Vous pouvez voir un rôle comme un **ensemble d'opérations** qui ont un **rôle commun**, comme par exemple le **rôle d'installer Apache**, ou le **rôle de configurer MariaDB**.

Les **répertoires** sont tous **optionnels**, excepté le **répertoire tasks** qui doit contenir le fichier **main.yml**. Ansible va traiter en premier ce fichier à l'appel d'un rôle.

Certains **répertoires** doivent **obligatoirement** contenir un **fichier main.yml** pour être pris en compte.



Tâche : Une tâche est une instruction décrite en YAML dans un **fichier de configuration**. Chaque tâche utilise un **module** ainsi que **quelques éventuels arguments supplémentaires**.

De façon schématique, vous pouvez retenir que :

- un rôle contient un ou plusieurs fichiers de configuration (YAML)
- un fichier de configuration contient une ou plusieurs tâches
- une tâche fait appel à un module.

Le **YAML** (Yet Another Markup Language). YAML permet d'écrire des structures de données qui peuvent être spécifiées sous la forme de listes.

Structurez votre déploiement avec les rôles Ansible

Identifiez les étapes pour installer MediaWiki

Vous avez vu dans la première partie de ce cours que l'installation de MediaWiki nécessitait 6 étapes que je reprends ici :

- Installer un serveur web Apache sur le premier serveur (srvApache).
- Installer PHP également sur le premier serveur (srvApache).
- Installer une base de données MariaDB sur le deuxième serveur (srvSql).
- Télécharger les fichiers sources de MediaWiki et les mettre sur le serveur web (srvApache).
- Configurer le serveur web (srvApache) pour pointer vers l'URL <http://srvApache/mediawiki>.
- Finaliser l'installation de MediaWiki via le script d'installation sur srvApache.

Ces 6 étapes comportent des **opérations d'installation** (étapes 1 à 3) et des **opérations de configuration** (étapes 4 à 6).

Transposez les étapes en rôles Ansible

Vous allez maintenant transposer cette logique à Ansible en utilisant les rôles.

Vous allez créer 5 rôles :

- Un rôle pour installer Apache : apache.
- Un rôle pour installer MariaDB : mariadb.
- Un rôle pour configurer Apache pour MediaWiki : confapache.
- Un rôle pour configurer MariaDB pour MediaWiki : confdb.
- Un rôle qui contiendra les variables globales : commun.

Vous allez donc suivre cette liste pour créer chaque rôle un par un.

Créez votre premier rôle

Vous allez commencer par **créer une arborescence de fichiers de configuration** qui permettra de dérouler les étapes de déploiement de MediaWiki et ensuite, dans le chapitre suivant, vous allez **écrire les scripts Ansible** dans ces fichiers de **configuration**. Je choisis volontairement cette approche pour que vous ayez une vision

structurelle avant de vous lancer dans l'écriture du code.

Où placer les rôles ?

Par défaut, **Ansible** va chercher les rôles dans le répertoire "roles" qui est placé dans le **répertoire de travail** (répertoire courant) : **celui dans lequel les commandes Ansible sont lancées**.

Créez le répertoire qui contiendra tous les rôles

Commencez par créer le répertoire “roles” dans votre espace de travail. Ce répertoire contiendra tous les rôles que vous allez créer.

```
(ansible) user-ansible@node-manager:~$ mkdir roles ; cd roles  
(ansible) user-ansible@node-manager:~/roles$
```

```
(ansible) user-ansible@nodeManager:~$ mkdir roles ; cd roles  
(ansible) user-ansible@nodeManager:~/roles$ []
```

Cependant, il est tout à fait possible d'indiquer à **Ansible** un **chemin différent en modifiant la variable d'environnement `DEFAULT_ROLES_PATH`, ou en modifiant l'option "roles_path"** (par défaut : `/etc/ansible/roles`) dans le fichier de configuration **Ansible** (`/etc/ansible/ansible.cfg`).

Pour vous aider à créer une arborescence complète, vous pouvez utiliser la commande : **ansible-galaxy**

Ansible Galaxy fait référence au site web de Galaxy où les utilisateurs peuvent partager des rôles (de la même façon que GitHub propose aux développeurs de stocker et de partager, publiquement ou non, le code qu'ils créent).

ansible-galaxy est un outil de ligne de commande pour télécharger, créer et gérer les rôles **Ansible**. Il aurait été tout à fait possible d'utiliser un rôle de la Galaxy pour faire l'installation d'Apache, par exemple.

Utilisez ansible-galaxy pour créer automatiquement un rôle

Allez-y, maintenant ! Créez votre premier rôle pour l'installation d'Apache avec la commande **ansible-galaxy** et l'option **init** : **ansible-galaxy init apache**

```
(ansible) user-ansible@nodeManager:~/roles$ ansible-galaxy init apache  
- Role apache was created successfully
```

Utilisez la commande **tree** pour afficher l'arborescence des fichiers : **tree apache**

```

bash: tree : commande introuvable
(ansible) user-ansible@nodeManager:~/roles$ tree apache
apache
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
└── tests
    ├── inventory
    └── test.yml
└── vars
    └── main.yml

```

L'arborescence contient les répertoires suivants :

- **files** : tous les fichiers à copier sur le node
- **templates** : tous les fichiers de template Jinja
- **tasks** : liste des instructions à exécuter (**le fichier main.yml est obligatoire**)
- **handlers** : même chose pour les instructions handlers (**le fichier main.yml est obligatoire**)
- **vars** : fichier contenant des déclarations de variables (**le fichier main.yml est obligatoire**) ; les variables définies ici sont prioritaires par rapport aux variables définies dans l'inventaire
- **defaults** : valeurs par défaut (**le fichier main.yml est obligatoire**) avec une priorité moindre
- **meta** : dépendances du rôle et informations (auteur, licence, plateformes...) sur le rôle (**le fichier main.yml est obligatoire**).

À la création de l'arborescence, **les fichiers sont vides** ! Il conviendra de les compléter en fonction des opérations à réaliser. **Vous avez compris qu'un rôle est avant tout une arborescence de fichiers qui contiendra des actions Ansible.**

Créez l'arborescence de vos rôles MediaWiki

Vous allez créer, un à un, la structure des 5 rôles pour le déploiement de MediaWiki.

Créez le rôle Apache

Assurez-vous que vous êtes bien dans le répertoire roles.

(ansible2.7.10) user-ansible@node-manager:~/roles\$

```

0 directories, 0 files
(ansible) user-ansible@nodeManager:~/roles$ 

```

Vous avez déjà créé la structure du rôle Apache.

Pour **installer Apache et PHP**, vous avez besoin d'**installer des paquets logiciels sur Linux** et de **redémarrer le service Apache** pour que les modifications soient prises en compte.

Vous auriez pu **conserver cette arborescence telle quelle**, mais pour votre simple besoin qui consiste à installer

Apache et PHP, vous allez faire un peu de toilettage pour **garder uniquement les répertoires nécessaires**.

Seuls les répertoires **handlers**, **meta** et **tasks** seront conservés :

- **handlers** : contient les tâches à exécuter après une notification (**redémarrer le service Apache**)
- **tasks** : contient les tâches à exécuter pour installer Apache.

Après avoir supprimé les répertoires inutiles avec la commande :

```
(ansible) user-ansible@node-manager:~/roles/apache$ rm -rf defaults/ files/ meta/ template/ tests/ vars/  
README.md  
'ansible) user-ansible@node-manager:~/roles/apache$
```

```
(ansible) user-ansible@nodeManager:~/roles$ ls -la  
total 12  
drwxr-xr-x  3 user-ansible user-ansible 4096  4 oct. 14:14 .  
drwxr-xr-x  9 user-ansible user-ansible 4096  4 oct. 14:11 ..  
drwxr-xr-x 10 user-ansible user-ansible 4096  4 oct. 14:14 apache
```

```
(ansible) user-ansible@nodeManager:~/roles/apache$ rm -rf defaults/ files/ meta/ template/ tests/ vars/ README.md  
(ansible) user-ansible@nodeManager:~/roles/apache$
```

voici ce que donne la nouvelle arborescence :

```
(ansible) user-ansible@node-manager:~/roles/apache$ tree  
. |  
└── handlers  
    └── main.yml  
└── tasks  
    └── main.yml
```

```
(ansible) user-ansible@nodeManager:~/roles/apache$ tree  
. |  
└── handlers  
    └── main.yml  
└── tasks  
    └── main.yml  
└── templates
```

Créez le rôle mariadb

Pour le rôle **mariadb**, vous avez besoin d'installer des paquets logiciels sur Linux et démarrer un service.

Vous n'avez donc pas besoin d'utiliser ansible-galaxy pour le rôle **mariadb**, un seul fichier de configuration est nécessaire ici :

```
(ansible) user-ansible@node-manager:~/roles$ mkdir -p mariadb/tasks  
(ansible) user-ansible@node-manager:~/roles$ touch mariadb/tasks/main.yml
```

```
(ansible) user-ansible@nodeManager:~/roles$ mkdir -p mariadb/tasks
(ansible) user-ansible@nodeManager:~/roles$ touch mariadb/tasks/main.yml
1
(ansible) user-ansible@nodeManager:~/roles$ ls -la
-bash: ls -la : commande introuvable
(ansible) user-ansible@nodeManager:~/roles$ ls -la
total 16
drwxr-xr-x 4 user-ansible user-ansible 4096 4 oct. 14:34 .
drwxr-xr-x 9 user-ansible user-ansible 4096 4 oct. 14:11 ..
drwxr-xr-x 5 user-ansible user-ansible 4096 4 oct. 14:30 apache
drwxr-xr-x 3 user-ansible user-ansible 4096 4 oct. 14:34 mariadb
```

Le fichier **main.yml** contiendra les tâches à exécuter pour installer **MariaDB**

Voici l'arborescence : **tree mariadb/mariadb/**

Erreur dans la commande du cour : il faut utiliser **tree mariadb/**

```
0 directories, 1 file
(ansible) user-ansible@nodeManager:~/roles$ tree mariadb/
mariadb/
└── tasks
    └── main.yml
```

Comme les opérations de **configuration** et les **variables** concernent spécifiquement le **déploiement de MediaWiki**, les **rôles de configuration** et **commun** vont être placés spécifiquement dans un répertoire **mediawiki**.

Nous créons donc un répertoire mediawiki, ce qui donne à ce stade :

```
(ansible) user-ansible@nodeManager:~/roles$ tree
.
├── apache
│   ├── handlers
│   │   └── main.yml
│   ├── tasks
│   │   └── main.yml
│   └── templates
├── mariadb
│   └── tasks
│       └── main.yml
└── mediawiki
```

Créez le rôle commun

Le rôle **commun** contiendra des variables partagées entre les rôles **confapache** et **confdb**. Ainsi, au lieu de définir les mêmes variables à deux endroits différents, il est préférable de créer un **rôle commun**, et d'utiliser une dépendance avec les autres rôles.

Pour cela, vous avez besoin du répertoire **defaults** et du fichiers **main.yml** qui contiendront les variables globales :

```
$ mkdir -p mediawiki/commun/defaults/
$ touch mediawiki/commun/defaults/main.yml
```

```
/ directories, 3 files
(ansible) user-ansible@nodeManager:~/roles$ mkdir -p mediawiki/commun/defaults/
(ansible) user-ansible@nodeManager:~/roles$ touch mediawiki/commun/defaults/main.yml
```

Ce qui donne :

```
(ansible) user-ansible@nodeManager:~/roles$ tree mediawiki/
mediawiki/
└── commun
    └── defaults
        └── main.yml
```

Créez le rôle confdb

Pour le rôle **confdb**, vous avez **besoin de créer une base de données et d'attribuer des droits sur cette base.**

Et **vous créerez une dépendance avec le rôle commun pour partager des variables globales.**

Pour cela, vous avez besoin des répertoires **meta** et **tasks** :

```
$ mkdir -p mediawiki/confdb/meta mediawiki/confdb/tasks
```

et des fichiers **main.yml** qui contiendront les actions :

```
$ touch mediawiki/confdb/tasks/main.yml
$ touch mediawiki/confdb/meta/main.yml
```

Ce qui donne :

```
/-- directories, 1 file
(ansible) user-ansible@nodeManager:~/roles$ mkdir -p mediawiki/confdb/meta mediawiki/confdb/tasks
(ansible) user-ansible@nodeManager:~/roles$ touch mediawiki/confdb/tasks/main.yml
(ansible) user-ansible@nodeManager:~/roles$ touch mediawiki/confdb/meta/main.yml
(ansible) user-ansible@nodeManager:~/roles$ tree mediawiki/
mediawiki/
├── commun
│   └── defaults
│       └── main.yml
└── confdb
    ├── meta
    │   └── main.yml
    └── tasks
        └── main.yml

5 directories, 3 files
```

Créez le rôle confapache

Vous allez partager des variables globales avec le rôle **confdb**, alors vous ajouterez une **dépendance avec le rôle commun.**

Vous allez également créer le **répertoire d'installation de MediaWiki, télécharger les fichiers MediaWiki sur le site officiel, lancer le script d'installation et mettre à jour la base de données.**

Pour cela, vous avez besoin des répertoires **meta** et **tasks** :

```
~/roles/mediawiki$ mkdir -p confapache/meta confapache/tasks
```

et des fichiers main.yml qui contiendront les actions :

```
$ touch confapache/tasks/main.yml confapache/meta/main.yml
```

```
(ansible) user-ansible@nodeManager:~/roles$ cd mediawiki/
(ansible) user-ansible@nodeManager:~/roles/mediawiki$ mkdir -p confapache/meta confapache/tasks
(ansible) user-ansible@nodeManager:~/roles/mediawiki$ touch confapache/tasks/main.yml confapache/meta/main.yml
```

Ce qui donne l'arborescence suivante :

```
(ansible) user-ansible@node-manager:~/roles$ tree mediawiki/
mediawiki/
├── commun
│   └── defaults
│       └── main.yml
├── confapache
│   ├── meta
│   │   └── main.yml
│   └── tasks
│       └── main.yml
└── confdb
    ├── meta
    │   └── main.yml
    └── tasks
        └── main.yml

8 directories, 5 files
```

Ma version, qui a bon ?

```
(ansible) user-ansible@nodeManager:~/roles$ tree mediawiki/
mediawiki/
└── commun
    └── defaults
        └── main.yml
    └── confapache
        ├── meta
        │   └── main.yml
        └── tasks
            └── main.yml
    └── confdb
        ├── meta
        │   └── main.yml
        └── tasks
            └── main.yml

8 directories, 5 files
```

Améliorez l'inventaire en vue du déploiement

Toujours dans l'esprit de **modularité**, vous allez **modifier le fichier inventaire.ini** pour **séparer les deux nodes dans deux groupes distincts**.

Ce qui permettra par la suite de **s'adresser à un groupe plutôt qu'aux nodes directement**. De cette façon, si vous voulez **ajouter un node**, vous aurez simplement à ajouter le node dans le bon groupe dans le fichier **inventaire**, et à **rejouer vos scripts sans rien changer d'autre**. Magique !

Connectez-vous sur le node manager :

```
$ ssh user-ansible@node-manager
```

```
...  
root@nodeManager:~# su - user-ansible  
user-ansible@nodeManager:~$ [ ]
```

Activez l'environnement virtuel :

```
$ source ansible2.7.10/bin/activate
```

Éditez le fichier inventaire.ini

```
$ vi inventaire.ini
```

```
GNU nano 5.4  
[apache]  
srvApache  
[db]  
srvSql
```

J'ai modifié directement dans ma session ansible/bin/activate

Nous avons ici **deux groupes entre crochets** qui **contiennent un node** chacun.

On déclare un groupe de serveurs en mettant le nom du groupe entre crochets[].

Le fichier **inventaire** est **très souple**, il permet des combinaisons intéressantes et surprenantes : **node[1:2]** pour **2 nodes** ou **[linux:children]** pour un groupe qui contient d'autres groupes.

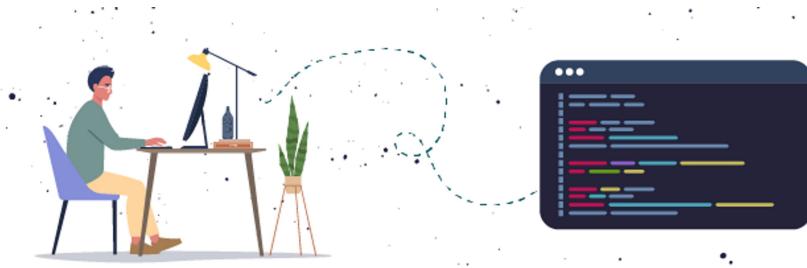
Résumé

Dans ce chapitre, vous avez **organisé les opérations d'automatisation avec les rôles Ansible** :

- vous avez **réparti votre installation de MediaWiki sur deux groupes de serveurs**, dans l'optique de pouvoir faire **évoluer votre infrastructure**
- vous avez compris **comment créer un rôle et identifier le contenu de l'arborescence d'un rôle**
- vous avez élaboré **une arborescence globale pour les rôles prédefinis**, en vue de faire un **déploiement structuré et automatisé de MediaWiki**.

Dans le prochain chapitre, nous allons voir comment contrôler l'exécution des opérations et enchaîner plusieurs actions en paramétrant les rôles.

Contrôlez l'exécution des opérations et enchaînez plusieurs actions



Dans le chapitre précédent, vous avez organisé le déploiement de MediaWiki en **créant des rôles structurés**. Vous avez transposé les **6 étapes nécessaires** au déploiement de MediaWiki en opérations d'installation et de configuration. Ensuite, vous avez créé **5 rôles** pour intégrer ces opérations dans une structure arborescente.

Dans ce chapitre, vous allez **construire les scripts d'automatisation** en complétant les fichiers `main.yml` présents dans chaque rôle. Ce qui permettra d'**exécuter des tâches** et d'**enchaîner plusieurs actions**. Pour vous aider à **construire les scripts**, vous suivrez les **6 étapes** d'installation de MediaWiki qui sont détaillées dans le guide d'installation MediaWiki.

Reprenez l'arborescence des rôles construite précédemment :

```
(ansible) user-ansible@nodeManager:~/roles$ tree
.
├── apache
│   ├── handlers
│   │   └── main.yml
│   ├── tasks
│   │   └── main.yml
│   └── templates
└── mariadb
    └── tasks
        └── main.yml
mediawiki
└── commun
    └── defaults
        └── main.yml
confapache
└── meta
    └── main.yml
confdb
└── meta
    └── main.yml
    └── tasks
        └── main.yml
```

On distingue, dans cette arborescence, **5 rôles** correspondant à **9 fichiers YAML** décrivant les rôles : **apache**, **mariadb**, **commun**, **confapache** et **confdb**.

Nous allons donc créer **8 fichiers YAML** pour les **5 rôles** dans ce chapitre :

- Le rôle **apache** :
 - o un fichier **tasks/main.yml** contient les **actions pour installer Apache**, un **appel à un fichier de configuration pour installer PHP** et une **notification pour redémarrer Apache**
 - o un fichier **handler/main.yml** contient les actions pour redémarrer le service **Apache**.
- Le rôle **mariadb** :
 - o un fichier **tasks/main.yml** contient les actions pour installer **MariaDB**.
- Le rôle **commun** de MediaWiki :

- un fichier **defaults/main.yml** contient les **variables d'installation** qui seront utilisées dans les rôles suivants.
- Le rôle **confapache** de MediaWiki :
 - un fichier **meta/main.yml** contient la dépendance avec le rôle commun
 - un fichier **tasks/main.yml** contient les actions pour configurer Apache pour MediaWiki.
- Le rôle **confdb** de MediaWiki :
 - un fichier **meta/main.yml** contient la dépendance avec le rôle commun
 - un fichier **tasks/main.yml** contient les actions pour configurer MariaDB pour MediaWiki.

Il est possible de créer dans le répertoire **tasks** autant de **fichiers de configuration** que vous voulez.

Vous pouvez par exemple, créer un fichier **php-install.yml** dédié à l'**installation de PHP** dans le répertoire **roles/apache/tasks** avec la commande suivante :

- **touch apache/tasks/pup-install.yml**

L'**installation de PHP** aurait pu être intégrée au **main.yml**, mais pour une question de **lisibilité** et de **modularité**, je vous propose ici de la gérer dans un **fichier** à part.

Les rôles peuvent être **indépendants** ou **dépendants** les uns des autres. Par exemple, les rôles **apache**, **mariadb** et **commun** sont **indépendants** et peuvent être **utilisés séparément**. Par contre, les rôles **confapache** et **confdb** dépendent du rôle commun.

Vous allez maintenant **compléter chaque fichier de configuration YAML**.

Construisez vos fichiers YAML

Un fichier de configuration **YAML** contenu dans les **rôles** peut contenir une **liste de tâches** ou une **liste de variables**.

Les fichiers **YAML** commencent toujours par **3 tirets** (**---**). Ensuite, vous avez les **différentes tâches successives** qui commencent par **1 tiret** (**-**) et le nom de la tâche. Chaque tâche utilise un module avec ses **arguments** ou ses **options**. **Les arguments ou les options sont décalés à la ligne de 2 espaces**.

Pour **construire** votre **fichier**, vous devez chercher dans la **documentation Ansible** quel **module** utiliser.

Pour installer **Apache**, il faut utiliser un **gestionnaire de paquets**. Sur **Debian**, vous utiliserez **APT**. Ça tombe bien, **APT** a un module **Ansible** permettant d'**automatiser l'installation de paquets**.

APT est le **gestionnaire de paquets Debian** ; vous allez pouvoir l'utiliser pour **installer le paquet Apache** qui s'appelle, sur Debian, **Apache2**.

Si vous voulez en savoir plus sur l'utilisation d'un **module Ansible**, comme par exemple pour le **module apt**, vous pouvez **utiliser la commande suivante** :

```
$ ansible-doc apt
```

Créez le fichier YAML pour le rôle apache

Écrivez le code pour installer Apache

Voici le fichier **tasks/main.yml** qui contient les actions d'installation d'**Apache** et de **PHP**.

\$ nano roles/apache/tasks/main.yml

Le fichier est **commenté** pour en comprendre le sens.

```
---
---
#1. Cette tâche permet d'installer Apache (srvApache) à l'aide du module apt
- name: "apache installation"
  apt:
    name: "apache2"
    state: "present"

#2. Cette tâche active le service Apache
- name: "apache service activation"
  service:
    name: "apache2"
    state: "started"
    enabled: yes

#3. Cette tâche fait appel à un autre fichier de configuration pour installer PHP. Elle est exécutée uniquement si la variable php_install est à vraie (par défaut, elle est à faux)
- name: "install php packages"
  include: "php-install.yml"
  when: php_install|default(False)|bool

# tasks file for apache
```

Voyons en détail chacune des tâches décrites dans ce fichier :

- La première tâche, "**apache installation**" va installer le serveur Apache avec le module "**apt**". Le **name: "apache2"** indique le paquet concerné et le **state: "present"** spécifie qu'il faut l'installer.
- La deuxième tâche, "**apache service activation**" va activer le service Apache avec le module "**service**". Le **name: "apache2"** indique le service concerné, le **state: "started"** indique que le service sera démarré et le **enabled: yes** indique que le service sera activé.
- La troisième tâche, "**install php packages**" inclut un fichier de configuration externe pour installer **PHP**. La tâche fait appel avec l'option "**include**" au fichier **php-install.yml** qui est placé dans le répertoire **tasks** à coté de **main.yml**. La condition **when** avec le filtre (**php_install|default(False)|bool**) permettent de conditionner l'installation de **PHP**.

La variable **php_install** est un **booléen** par défaut à **false**. Cette variable va permettre d'indiquer si **PHP** doit être installé ou non. **Cette variable sera initialisée à l'appel du rôle**.

Attention, l'**indentation est importante**. Il faut décaler de **deux espaces** chaque ligne pour respecter l'**alignement logique de chaque tâche**. Si vous utilisez un éditeur de code de type Sublime Text ou **Visual Studio Code**, l'indentation sera automatique en sélectionnant la **syntaxe YAML** dans votre éditeur.

Au début, vous allez forcément rencontrer des erreurs liées à cette rigueur imposée par le langage python sur l'indentation (https://www.w3schools.com/python/python_syntax.asp). Pour plus de lisibilité, je vous proposerai également à chaque fichier une copie d'écran du résultat. Par exemple, ici, pour le fichier **roles/apache/tasks/main.yml** .

```
(ansible) user-ansible@node-manager:~$ cat roles/apache/tasks/main.yml
...
#1. Cette tâche permet d'installer Apache (httpd) à l'aide du module apt
- name: "apache installation"
  apt:
    name: "apache2"
    state: "present"

#2. Cette tâche active le service Apache
- name: "apache service activation"
  service:
    name: "apache2"
    state: "started"
    enabled: yes

#3. Cette tâche fait appel à un autre fichier de configuration pour installer PHP.
#   Elle est exécutée uniquement si la variable php_install est à vraie (par défaut, elle est à faux)
- name: "install php packages"
  include: "php-install.yml"
  when: php_install|default(False)|bool
(ansible) user-ansible@node-manager:~$
```

Capture d'écran pour les actions d'installation d'Apache

Les filtres Ansible sont sous la forme Jinja2, qui transforme une information en une expression régulière.

Ajoutez un autre fichier de configuration pour l'installation de PHP

Vous allez ajouter un fichier pour l'installation spécifique de PHP (`php7-install.yml`) afin de séparer l'installation d'Apache et l'installation de PHP.

```
(ansible) user-ansible@node-manager:~$ cat roles/apache/tasks/php-install.yml
...
#1. Cette tâche installe PHP et ses extensions
- name: "install php packages"
  apt:
    name: "php,php-mysql,php-xml,php-mbstring,php-gd,php-intl"
    state: latest
    changed_when: yes
    notify: [ "apache restart" ]
(ansible) user-ansible@node-manager:~$
```

Capture d'écran pour les actions d'installation de PHP

Voyons en détail chacune des tâches décrites dans ce fichier :

- “**install php packages**”, installe les paquets **php** avec le module “**apt**”. Et les options :
 - le **name** indique l'ensemble des paquets à installer
 - **state: latest** indique qu'il faut installer les dernières versions disponibles des paquets
 - **changed_when** : force le changement d'état, c'est-à-dire qu'avec cette **condition à yes**, l'**exécution de la tâche provoquera un changement**
 - **notify: ["apache restart"]** indique que si la tâche change d'état (et uniquement si la tâche a engendré un changement), **notify fait appel au handler "apache restart" pour relancer le service Apache**.

Ansible est **idempotent**, cela signifie qu'une opération a le même effet, qu'on l'applique une ou plusieurs fois. Si l'état demandé dans l'action est conforme à ce qui est demandé, alors Ansible ne fait rien.

L'idempotence est possible car Ansible gère 3 états d'exécution pour une tâche : **skipping**, **ok** ou **changed**.

Créez le script handler pour relancer un service

Les **handlers**, ou **script handlers**, sont communément utilisés pour **regrouper des actions récurrentes**, comme

des **redémarrages de services**, par exemple.

Les **handlers** sont exécutés uniquement si une tâche les appelle à travers un **notify**, une **notification déclenchée par la tâche**. Le **notify** se déclenche si l'état de la tâche est **changed**.

C'est plutôt utile pour **redémarrer un service si et seulement si l'état du serveur a changé**, ce qui évite des **interruptions de service inutiles**.

Dans le fichier **roles/apache/handlers/main.yml**, vous allez utiliser le module **service** pour relancer le service Apache.

```
(ansible) user-ansible@node-manager:~$ cat roles/apache/handlers/main.yml
---
- name: "apache restart"
  service:
    name: "apache2"
    state: "restarted"
(ansible) user-ansible@node-manager:~$
```

Capture d'écran des scripts handlers

Rien de spécial ici, juste une tâche assez simple pour relancer le service.

Le module **service** est utilisé pour relancer le service **apache2** avec l'option **state: "restarted"**.

Toutes les informations relatives à l'utilisation du module **service** sont détaillées dans la documentation Ansible.

Créez le fichier YAML pour le rôle mariadb

Écrivez le code pour installer MariaDB

\$ nano roles/mediawiki/mariadb/tasks/main.yml

```
# (ansible) user-ansible@node-manager:~$ cat roles/mariadb/tasks/main.yml
---

# Installation des paquets mariadb serveur et son extension Python
- name: "mariadb-server installation"
  apt:
    name: "mariadb-server,python3-mysqldb"
    state: "present"

# Active le service MariaDB
- name: "start mariadb service"
  service:
    name: "mariadb"
    state: "started"
    enabled: yes

# Active l'écoute sur tous les ports IPv4
- name: "change 50-server.cnf"
  command:
    /usr/bin/sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mysql/mariadb.conf.d/50-server.cnf

# Restart mariadb
- name: "restart mariadb"
  service:
    name: "mariadb"
    state: restarted
(ansible) user-ansible@node-manager:~$
```

Capture d'écran code d'installation de MariaDB

La première tâche "**mariadb-server installation**" va installer **mariadb serveur** et son **extension Python** avec le module "**apt**". Le **name: "mariadb-server,python3-mysqldb"** indique les paquets concernés et **state: "installed"** indique que les paquets doivent être installés.

state peut prendre indifféremment la valeur **present** ou **installed**. Ce sont **deux alias**.

La deuxième tâche “**start mariadb service**” va activer et démarrer le service **MariaDB** avec le module “**service**”. Le **name: "mariadb"** indique le service concerné, **state: "started"** indique de démarrer le service et **enabled:yes**, de l'activer.

Créez le fichier YAML pour le rôle commun

Créez le fichier des variables globales

Les variables globales vont être utilisées dans les rôles **confapache** et **confdb**. Elles vont donc être définies dans le rôle **commun**, dans le répertoire **defaults** et dans le fichier **main.yml**.

Pour connaître les **variables** à définir, il faut **étudier le guide d'installation** et **repérer** ce qui est potentiellement une **variable**. C'est très souvent des noms de répertoire, de base de données ou des logins, mots de passe, etc. Elles définissent tout ce qui est **variable**, c'est-à-dire ce qui peut **changer** ou être **personnalisé**.

Bien que les variables puissent être définies à plusieurs endroits, les bonnes pratiques recommandent de les mettre dans le répertoire **defaults**.

```
$ nano roles/mediawiki/commun/defaults/main.yml
```

```
(ansible) user-ansible@node-manager:~$ cat roles/mediawiki/commun/defaults/main.yml
---

# Base de données
mediawiki_db_name: "mediawiki"

# User et password
mediawiki_db_user: "mediawiki"
mediawiki_db_password: !vault |
$ANSIBLE_VAULT;1.1;AES256
3735386532636238312396331636337396335393838396362636462613533323832323136316433
6637313835626265393861656137353639626431326164350a303362666161333234623239353035
3237386138373166303937623965326131626464393336530653366363836326463323062643064
6163633234666566360a616135306337363532623263373034303430396431316432373039386561
61623630333735616632313835656466363139386531633534313631313364326436

# User admin et password
mediawiki_admin_user: "admin"
mediawiki_admin_password: !vault |
$ANSIBLE_VAULT;1.1;AES256
3735386532636238312396331636337396335393838396362636462613533323832323136316433
66373138373166303937623965326131626464393336530653366363836326463323062643064
6163633234666566360a616135306337363532623263373034303430396431316432373039386561
61623630333735616632313835656466363139386531633534313631313364326436

# nom du Mediawiki et son titre
mediawiki_name: "mediawiki"
mediawiki_title: "Mon super mediawiki !"

# Répertoire d'installation
mediawiki_directory: "/var/www/html/{{mediawiki_name}}"

# Répertoire maintenance
mediawiki_maintenance_directory: "{{mediawiki_directory}}/maintenance"

# Définie le premier node du groupe mariadb
mediawiki_db_host: "{{groups.db.0}}"

# l'url des sources Mediawiki
mediawiki_archive_url: "https://releases.wikimedia.org/mediawiki/1.37/mediawiki-1.37.1.tar.gz"
(ansible) user-ansible@node-manager:~$
```

Capture d'écran montrant les variables globales

(pas fait pendant les exemples)

Au format **YAML**, les variables sont définies par un **couple “clé: valeur”** séparé par deux points (:).

Vous pouvez voir également dans ce fichier, que deux valeurs sont utilisées : **{{mediawiki_name}}** et **{{mediawiki_directory}}**, qui font respectivement appel aux deux variables **mediawiki_name** et **mediawiki_directory** contenues dans le fichier.

Pour exploiter les variables **YAML**, il faut les **définir entre accolades**.

En regardant encore un peu plus près le fichier, vous constatez que les mots de passe sont chiffrés. En effet, il n'est vraiment pas recommandé de mettre des mots de passe en clair dans les fichiers de configuration.

Utilisons la commande **ansible-vault** pour chiffrer des chaînes de caractères.

Chiffrez vos mots de passe

Vous allez retenir le mot de passe suivant : **AnsibleOC**, car il va vous servir à chiffrer vos mots de passe.

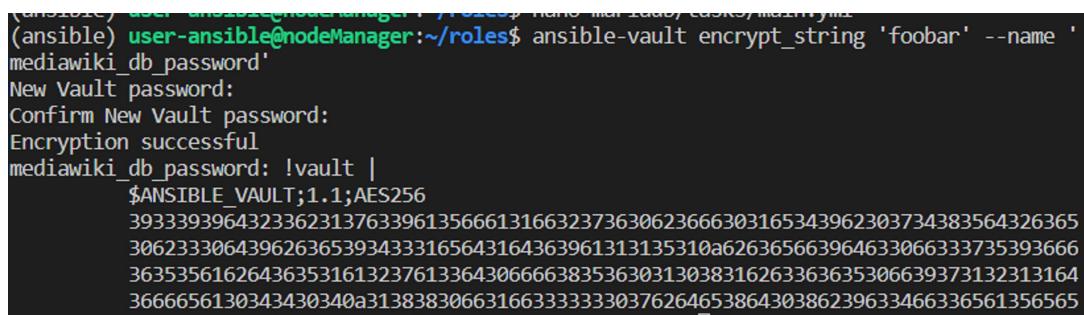
Pour chiffrer le mot de passe qui sera utilisé dans le fichier de variables, vous pouvez utiliser la commande suivante :

```
$ ansible-vault encrypt_string 'foobar' --name 'mediawiki_db_password'  
New Vault password:  
Confirm New Vault password:
```

Saisissez la clé de chiffrement AnsibleOC et confirmez-la.

Voici la chaîne générée par Ansible :

```
mediawiki_db_password: !vault |  
$ANSIBLE_VAULT;1.1;AES256  
39333939643233623137633961356661316632373630623666303165343962303734383564326365  
3062333064396263653934333165643164363961313135310a626365663964633066333735393666  
36353561626436353161323761336430666638353630313038316263363635306639373132313164  
3666656130343430340a3138383066316633333037626465386430386239633466336561356565
```



```
(ansible) user-ansible@nodeManager:~/roles$ ansible-vault encrypt_string 'foobar' --name 'mediawiki_db_password'  
New Vault password:  
Confirm New Vault password:  
Encryption successful  
mediawiki_db_password: !vault |  
$ANSIBLE_VAULT;1.1;AES256  
39333939643233623137633961356661316632373630623666303165343962303734383564326365  
3062333064396263653934333165643164363961313135310a626365663964633066333735393666  
36353561626436353161323761336430666638353630313038316263363635306639373132313164  
3666656130343430340a3138383066316633333037626465386430386239633466336561356565
```

Il faudra la reporter dans le fichier des variables, comme cela a été fait plus haut, en veillant à bien copier et coller l'intégralité de la chaîne sans en déformer le format (c'est souvent source d'erreur).

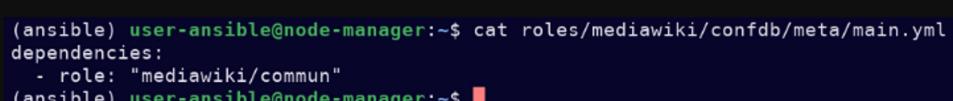
Créez le fichier YAML pour le rôle confdb

Déclarez la dépendance avec le rôle commun

Pour que le rôle **confdb** puisse utiliser les variables enregistrées dans le rôle **commun**, il faut déclarer cette dépendance dans le répertoire **meta** :

```
$ nano roles/mediawiki/confdb/meta/main.yml
```

La déclaration se fait de la façon suivante :



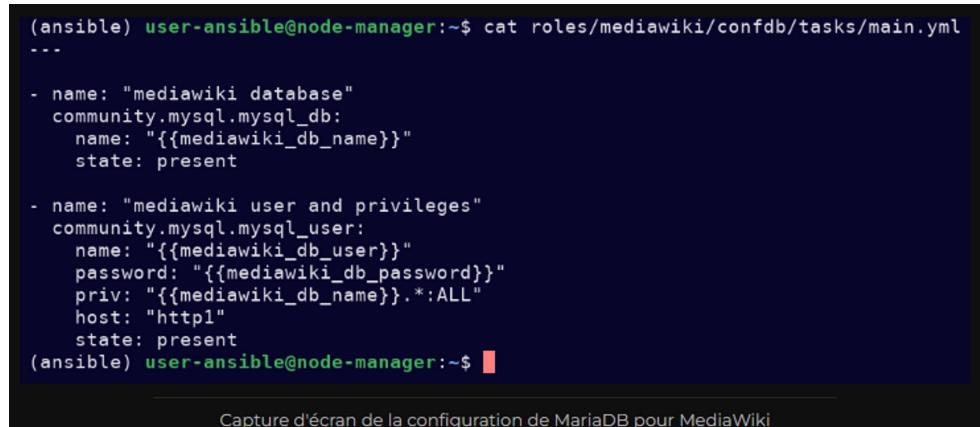
```
(ansible) user-ansible@node-manager:~$ cat roles/mediawiki/confdb/meta/main.yml  
dependencies:  
  - role: "mediawiki/commun"  
(ansible) user-ansible@node-manager:~$ █
```

Capture d'écran de la déclaration de dépendance avec le rôle commun

Écrivez le code pour configurer MariaDB pour MediaWiki

La configuration va se faire dans le fichier suivant :

```
$ nano roles/mediawiki/confdb/tasks/main.yml
```



```
(ansible) user-ansible@node-manager:~$ cat roles/mediawiki/confdb/tasks/main.yml
---
- name: "mediawiki database"
  community.mysql.mysql_db:
    name: "{{mediawiki_db_name}}"
    state: present

- name: "mediawiki user and privileges"
  community.mysql.mysql_user:
    name: "{{mediawiki_db_user}}"
    password: "{{mediawiki_db_password}}"
    priv: "{{mediawiki_db_name}}.*:ALL"
    host: "http1"
    state: present
(ansible) user-ansible@node-manager:~$
```

Capture d'écran de la configuration de MariaDB pour MediaWiki

(pas fait pendant les exemples, à faire pour tout finir : remplac " hhttp1 par srvApache)

Voyons ensemble le détail de ces tâches :

- La première tâche "mediawiki database" va créer la base de données avec le module "mysql_db". Et les options :
 - le **name**: "{{mediawiki_db_name}}" indique le nom de la base de données qui est récupéré de la variable **mediawiki_db_name** définie dans le rôle commun
 - le **state: present** indique de créer la base de données.
- La deuxième tâche "mediawiki user+privileges" va créer un accès utilisateur et des privilèges associés sur la base de données pour tous les nodes avec le module "mysql_user". Et les options :
 - le **name**: "{{mediawiki_db_user}}" indique le nom d'utilisateur à créer
 - **password**: "{{mediawiki_db_password}}" indique le mot de passe à utiliser
 - **priv**: "{{mediawiki_db_name}}.*:ALL" indique les privilèges à donner à l'utilisateur
 - **host**: "{{item}}" indique le node concerné par les accès
 - **state:present** indique de créer l'utilisateur et ses accès.

Vous remarquez que les variables communes sont utilisées ici. Cela reprend l'idée de modularité et de réutilisation des scripts.

Une nouvelle notion s'est ajoutée à la configuration :

- **with_items**: La boucle **with_items** va parcourir le groupe Apache et remplacer successivement la variable **item** par les noms des **nodes** présents dans le groupe
- "{{groups.apache}}": cette variable "magique" est tirée directement du fichier inventaire.

[Créez le fichier YAML pour le rôle confapache](#)

Déclarez la dépendance avec le rôle commun

Pour que le rôle **confapache** puisse utiliser les variables enregistrées dans le rôle **commun**, il faut déclarer cette dépendance dans le répertoire **meta** :

nano roles/mediawiki/confapache/meta/main.yml

```
(ansible) user-ansible@nodeManager:~$ nano roles/mediawiki/confapache/meta/main.yml
(ansible) user-ansible@nodeManager:~$ cat roles/mediawiki/confapache/meta/main.yml
dependencies:
  - role: "mediawiki/commun"
(ansible) user-ansible@nodeManager:~$
```

Nous sommes bientôt au bout ! Vous allez maintenant terminer en **configurant les fichiers MediaWiki dans Apache** :

```
(ansible) user-ansible@node-manager:~$ cat roles/mediawiki/confapache/tasks/main.yml
---

#0. Ajout du compte user-ansible dans le groupe www-data
# pour la gestion des droits sur l'arborescence mediawiki
- name: "add user-ansible"
  command:
    usermod -a -G www-data user-ansible

#1. Création du répertoire pour l'installation des fichiers Mediawiki
- name: "mediawiki directory"
  file:
    path: "{{mediawiki_directory}}"
    owner: "www-data"
    group: "www-data"
    state: directory

#2. Décomprime le fichier source archive Mediawiki et le formate sans extension
- name: "uncompress mediawiki archive"
  unarchive:
    src: "{{mediawiki_archive_url}}"
    dest: "{{mediawiki_directory}}"
    owner: "www-data"
    group: "www-data"
    remote_src: yes
    # supprime mediawiki-1.xx.x/ du chemin
    extra_opts: --transform=s/mediawiki-[0-9\.]*\///

#3. Ajoute les droits d'écriture pour le groupe www-data
- name: "add g+w sur le répertoire mediawiki"
  command:
    chmod -R g+w {{mediawiki_directory}}

#4. Exécute la tâche avec l'utilisateur ansible, se place dans le répertoire
# de maintenance et exécute la commande de configuration si le fichier
# LocalSettings.php n'existe pas
- name: "mediawiki configuration"
  become: yes
  become_user: "user-ansible"
  args:
    creates: "{{mediawiki_directory}}/LocalSettings.php"
    chdir: "{{mediawiki_maintenance_directory}}"
  command:
    php install.php --scriptpath/{{mediawiki_name}}
    --dbname mediawiki --lang fr
    --dbuser {{mediawiki_db_user}}
    --dbpass {{mediawiki_db_password}}
    --pass {{mediawiki_admin_password}}
    --dbserver {{mediawiki_db_host}}
    --server http://http1
    {{mediawiki_title}} {{mediawiki_admin_user}}
  run_once: yes
  delegate_to: "{{item}}"
  with_items: "{{groups.apache}}"
```

```

#5. Exécute la tâche avec l'utilisateur ansible, se place dans le répertoire
# de maintenance et exécute la commande de mise à jour de la base
# une seule fois
- name: "mediawiki db update"
  become: yes
  become_user: "user-ansible"
  command:
    php update.php --quick
  args:
    chdir: "{{mediawiki_maintenance_directory}}"
  # La mise à jour à besoin d'être lancée une seule fois
  run_once: yes
  register: resultat
  changed_when: "'...done.' in resultat.stdout"
(ansible) user-ansible@node-manager:~$ 

```

La première tâche, "mediawiki directory", va créer le répertoire dans lequel les fichiers MediaWiki seront placés avec le module "file". Et les options :

- **path:** "{{mediawiki_directory}}" indique le chemin du répertoire à créer ;
- **owner:** "www-data" indique le propriétaire du répertoire ;
- **group:** "www-data" indique le groupe du répertoire ;
- **state:** directory indique de créer un répertoire.

La deuxième tâche, "mediawiki directory", va décompresser le fichier source archive MediaWiki et le renommer avec le module "unarchive". Et les options :

- **src:** "{{mediawiki_archive_url}}" indique l'adresse de l'archive à télécharger ;
- **dest:** "{{mediawiki_directory}}" indique le chemin où doit être décompressée l'archive ;
- **owner:** "www-data" indique le propriétaire des fichiers ;
- **group:** "www-data" indique le groupe des fichiers ;
- **remote_src:** yes indique que la source est externe ;
- **extra_opts:** --transform=s/mediawiki-[0-9\.]*/// indique de renommer le répertoire avec le nom mediawiki.

La troisième tâche, "mediawiki configuration", va configurer MediaWiki à l'aide d'une commande PHP en utilisant le module command. Et les arguments et options :

- **become:** yes indique d'utiliser un autre utilisateur ;
- **become_user:** "www-data" indique d'utiliser le compte Apache pour faire les actions ;
- **args:** indique les arguments à utiliser ;
- **creates:** "{{mediawiki_directory}}/LocalSettings.php" indique de ne rien faire si le fichier LocalSettings.php existe déjà ;
- **chdir:** "{{mediawiki_maintenance_directory}}" indique de se placer dans le répertoire Maintenance,
- **command:** indique d'utiliser le module command pour lancer le script install.php avec les options de configuration ;
- **run_once:** yes indique de l'exécuter une seule fois ;

- **delegate_to**: "{{item}}" indique de déléguer l'action à un item de la liste ;
- **with_items**: "{{groups.apache}}" indique le nom du node du groupe Apache défini dans l'inventaire et le place dans Item.

La quatrième tâche, "mediawiki db update", va mettre à jour la base de données pour finir la configuration de MediaWiki avec le module command. Et les options :

- **become**: yes indique de prendre l'identité d'un autre utilisateur ;
- **become_user**: "apache" indique d'utiliser le compte Apache ;
- **command**: indique de lancer le script update.php à l'aide de la commande PHP et l'option rapide ;
- **args**: indique de se placer dans le répertoire de maintenance ;
- **run_once**: yes indique de faire l'action une seule fois ;
- **register**: resultat indique de stocker le résultat dans la variable result ;
- **changed_when**: "'...done.' in resultat.stdout" indique que le changement est effectif quand l'action retourne done.

Quelques notions particulières de ce fichier sont à préciser :

args : la section args va mettre d'**indiquer à la tâche les prérequis à l'exécution des actions** ;

run_once : l'action sera **exécutée** une seule fois ;

delegate_to : la tâche est **déléguée au premier node du groupe Apache**. Couplée avec **run_once**, l'action sera **exécutée sur un seul node**, ce qui évitera des corruptions de base, si les actions sont jouées sur deux nodes en même temps ;

register : va stocker le résultat de l'action dans la variable **resultat**. Couplé à **changed_when**, l'état de la tâche sera indiqué **changed**, si l'action retourne **done**

Ansible lance les tâches en mode push (en parallèle) sur tous les serveurs en même temps (5 par 5). Par contre, les tâches sont exécutées séquentiellement les unes à la suite des autres. La première tâche sera envoyée et exécutée sur tous les serveurs avant de passer à la tâche suivante. Ce qui veut dire que le node manager attend une réponse de tous les nodes et attend qu'ils aient tous terminé l'exécution de la tâche en cours avant de passer à la suivante.

Résumé

Dans ce chapitre, vous avez contrôlé l'exécution des opérations et enchaîné plusieurs actions :

- vous avez appris comment construire des fichiers de configuration YAML pour les 5 rôles ;
- vous avez utilisé les fichiers tasks, les handers et des variables globales ;
- vous avez chiffré des mots de passe et déclaré des dépendances entre les rôles.

Dans le prochain chapitre, nous allons enfin assembler le tout pour lancer le déploiement de MediaWiki en quelques lignes de code.

Assemblez les opérations avec les playbooks pour automatiser le déploiement

Dans le chapitre précédent, vous avez **contrôlé l'exécution des opérations et enchaîné plusieurs actions**, en écrivant du **code Ansible** dans les **fichiers de configuration** présents dans les différents rôles.

Dans ce chapitre, vous allez **assembler toutes ces opérations et automatiser le déploiement de MediaWiki**. Pour cela, vous allez utiliser les **playbooks Ansible**. Ensuite, vous jouerez ces **playbooks** avec la commande **ansible-playbook**, afin de **déployer automatiquement MediaWiki**.

Vérifiez le niveau de sécurité sur les nodes

Avant de lancer les configurations sur les nodes, vérifiez que le **AppArmor** et le **firewalling** ne sont pas activés sur les nodes Debian. En effet, ces blocages de sécurité par défaut peuvent empêcher le déploiement de MediaWiki. Il est donc préférable de les désactiver.

En situation professionnelle, il convient d'ajuster la sécurité en fonction des flux autorisés. Dans notre cas, cet aspect n'est volontairement pas traité, pour se concentrer sur l'apprentissage de Ansible. Mais en temps normal, il est vivement conseillé de se rapprocher de l'expert sécurité, pour intégrer les modifications de sécurité nécessaires aux nouveaux déploiements.

Voici les commandes à passer pour s'affranchir des blocages de sécurité :

- pour désactiver temporairement AppArmor : **systemctl stop apparmor.service**
- pour désactiver définitivement AppArmor : **systemctl stop apparmor.service**
- pour désactiver temporairement le firewalling : **systemctl stop firewalld.service**
- pour désactiver définitivement le firewalling : **systemctl disable firewalld.service** .

```
root@srvApache:~# systemctl stop apparmor.service
root@srvApache:~# systemctl disable firewalld.service
Failed to disable unit: Unit file firewalld.service does not exist.
```

Activez la résolution de noms sur les nodes

Comme indiqué précédemment, comme il n'y a pas de gestionnaire de noms **DNS**, il faut **enregistrer les adresses IP et les noms des nodes sur chacun d'entre eux**, pour qu'ils puissent résoudre les noms de machines utilisées dans les configurations.

Sur tous les **nodes**, ajoutez à la suite du fichier **/etc/hosts** les éléments suivants :

```
root@srvApache:~# cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      srvApache
192.168.1.82    srvSql
192.168.1.80    nodeManager
```

```
root@srvSql:~# cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      srvSql
192.168.1.81    srvApache
192.168.1.80    nodeManager
```

Ainsi, vous pouvez pinger **srvApache** et **srvSql** depuis les nodes. Ce qui évite d'utiliser directement les adresses IP.

Créez les playbooks pour exécuter les rôles

Playbook : Un **playbook** est un **fichier de configuration YAML** contenant une **suite de jeux d'instructions**, ou **plays** en anglais. Chacun peut être constitué d'**options**, et fait appel à un ou plusieurs rôles. Il permet de décrire une **stratégie de déploiement**, ou de **configuration**, en structurant les **actions nécessaires**.

En utilisant les **playbooks**, vous avez la possibilité de **conserver le code dans un fichier** et de le réutiliser à votre façon, contrairement à la commande **ansible** qui est volatile.

Dans la suite ce chapitre, vous allez donc créer **3 playbooks** :

- Un pour **installer Apache**.
- Un pour **installer MariaDB**.
- Un pour **configurer MediaWiki**.

L'ordre d'exécution de ces **playbooks** est important, car les rôles de configuration dépendent des rôles d'installation. Les serveurs **Apache** et **MariaDB** doivent être installés avant que la configuration de **MediaWiki** puisse être lancée. De la même façon, la **configuration de MediaWiki sur le serveur MariaDB** doit être effectuée avant la configuration du serveur state: present.

Toutes les opérations qui vont suivre sont à faire sur le node manager.

Tout d'abord, connectez-vous sur le node manager :

```
$ ssh user-ansible@node-manager
```

Puis, activez l'**environnement virtuel** :

```
$ source ansible2.7.10/bin/activate
```

Créez le playbook pour installer Apache

L'installation **d'Apache** va consister à lancer le rôle **apache** et à définir la variable qui permettra d'indiquer si **PHP** doit être installé ou pas.

Créez le **playbook install-apache.yml** à la racine de votre environnement virtuel :

- nano **install-apache.yml**

```
(ansible) # user-ansible@node-manager:~$ cat install-apache.yml
---
- name: "Installation apache"
  hosts: http1
  roles:
    - role: "apache"
      php_install: yes
(ansible) user-ansible@node-manager:~$
```

La structure d'un playbook

Le **playbook** est un **fichier YAML**. Il est structuré de la façon suivante :

- il commence par **3 tirets**
- ensuite, **il peut y avoir un bloc général**, constitué d'un **en-tête composé du nom des nodes ou du groupe concernés, de variables, d'options, etc.**
- puis, vous trouvez des **blocs spécifiques** qui **définissent les jeux d'instruction** (ou **play** en anglais). Chaque **jeu d'instruction** est composé du **nom du jeu** (avec un tiret au début), et de **façon optionnelle**, d'un **en-tête** et d'une **section** qui peut prendre plusieurs formes comme des **tasks**, des **rôles**, des **handlers**... en fonction de l'action demandée.
Dans le cas du **playbook install-apache.yml** ci-dessus, il y a 2 jeux d'instruction.
- chaque **ligne** à l'intérieur d'un bloc est **indentée**, et chaque début de ligne est décalé de **deux espaces**.

Comme vous pouvez le voir, la principale **différence** entre un **playbook** et un **fichier de configuration** de type **main.yml** est que le **playbook contient une liste de jeux d'instructions (plays)** et que le **main.yml** contient une liste de **tâches (tasks)** ou de **variables**.

Vous trouverez un descriptif plus détaillé du [fonctionnement des playbooks sur sa documentation](#).

Dans **install-apache.yml**, il n'y a pas de **bloc général**. Par contre, il y a un **bloc spécifique** pour **définir le jeu d'instructions** qui consiste à lancer le rôle apache et à définir la valeur de la variable **php_install** (ce qui déclenchera l'**installation de PHP**). Il y a également un en-tête qui indique sur quel **node** il faut lancer les actions.

Le jeu d'instructions est défini de la façon suivante :

- **name: "Installation apache"** indique le nom du jeu d'instructions
- **hosts: srvApache** indique le node concerné
- **roles:** indique une section rôles
- **role: "apache"** indique le rôle à lancer
- **php_install: yes** indique la valeur de la variable **php_install**.

Lancez le playbook pour installer Apache sur http1

Lancez la commande **ansible-playbook** pour exécuter le **playbook install-apache.yml** avec les options de connexion suivantes :

- **ansible-playbook -i inventaire.ini --user user-ansible --become --ask-become-pass install-apache.yml**

Vous avez utilisé la commande **ansible-playbook** avec les mêmes options que celles utilisées avec la commande **ansible**. **Seul le node concerné est directement défini dans le playbook.**

ansible-playbook fait partie des outils installés avec Ansible. Cette commande permet de lancer des **playbooks**. Les options sont similaires à celles de la **commande ansible** utilisée précédemment.

Le résultat (appelé **callback d'affichage**) prend la forme suivante :

```
(ansible) user-ansible@node-manager:~$ ansible-playbook -i inventaire.ini --user user-ansible

BECOME password:
PLAY [Installation apache] *****
TASK [Gathering Facts] *****
ok: [http1]

TASK [apache : apache installation] *****
changed: [http1]

TASK [apache : apache service activation] *****
ok: [http1]

TASK [apache : install php packages] *****
changed: [http1]

RUNNING HANDLER [apache : apache restart] *****
changed: [http1]

PLAY RECAP *****
http1 : ok=5 changed=3 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

Vous pouvez constater que le **callback** donne la **liste des tâches pour chaque action définie** dans le rôle **apache**, avec leur **état (changed)**. Ce qui signifie que toutes les actions ont provoqué un changement sur le **node http1**.

La dernière ligne du callback indique le récapitulatif par node, de l'exécution des tâches selon les quatre états possibles (**ok, changed, unreachable, failed**).

*Ansible récupère sur le **node manager** les **variables d'environnement de chaque node** (les **facts**). C'est une tâche (**gatheringfact**) automatique par défaut au lancement du **playbook**. Ces variables permettent de connaître plus de 800 informations sur chaque node (adresse IP, version de l'OS...). Ces variables peuvent être ensuite utilisées dans les scripts.*

Il est temps de vérifier sur le node **http1**, si **Apache** et **PHP** ont bien été installés.

Connectez-vous sur le node **http1** et lancez la commande qui permet de connaître la **version de PHP** :

```
(ansible) user-ansible@node-manager:~$ ssh user-ansible@http1
$ php --version
PHP 7.4.25 (cli) (built: Oct 23 2021 21:53:50) ( NTS )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies with Zend OPcache v7.4.25, Copyright (c) 2021
```

Lancez un navigateur et connectez vous sur <http://http1> .



C'est tout bon ! Par défaut, Apache affiche une page de test, ce qui permet de vérifier que le service est opérationnel. Et **PHP** est bien installé.

Créez le playbook pour installer MariaDB

L'installation de **MariaDB** va simplement consister à lancer le rôle **mariadb**.

Créez le playbook **install-mariadb.yml** à la racine de votre environnement virtuel :

```
$ vi install-mariadb.yml
<| Le contenu du fichier : |>
(ansible) user-ansible@node-manager:~$ cat install-mariadb.yml
---
- name: "Installation MariaDB"
  hosts: bdd1
  gather_facts: no
  roles:
    - role: mariadb
(ansible) user-ansible@node-manager:~$
```

Capture d'écran du playbook pour l'installation de MariaDB

Le **playbook** contient **une seule tâche**, qui **définit le nom de l'host concerné par l'action**, l'**option gather_facts** est désactivée et le rôle à lancer est "**mariadb**".

Lancez la commande **ansible-playbook** pour exécuter le **playbook install-mariadb.yml** avec les options de connexion suivantes :

```
ansible-playbook -i inventaire.ini --user user-ansible --become --ask-become-pass install-mariadb.yml
```

```

PLAY [Installation MariaDB] ****
TASK [mariadb : mariadb-server installation] ****
changed: [bdd1]

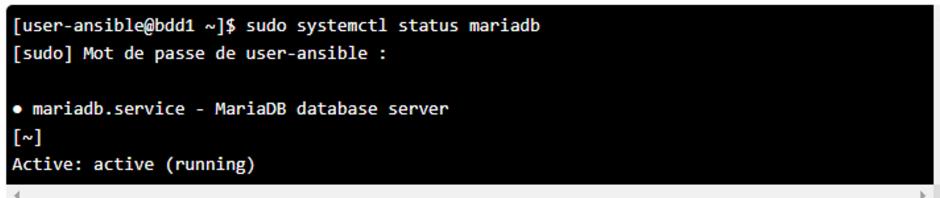
TASK [mariadb : start mariadb service] ****
ok: [bdd1]

PLAY RECAP ****
bdd1          : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 i

```

Même chose que pour le **playbook précédent**, les **différentes étapes d'exécution du rôle mariadb sont listées**, et un **résumé est donné en fin de liste avec l'état global de l'exécution**. Deux tâches ont été exécutées, elles ont provoqué un changement sur le node **bdd1**.

Pour le vérifier, connectez-vous sur le **node bdd1** et interrogez le statut du service **MariaDB** avec la commande suivante :



```
[user-ansible@bdd1 ~]$ sudo systemctl status mariadb
[sudo] Mot de passe de user-ansible :

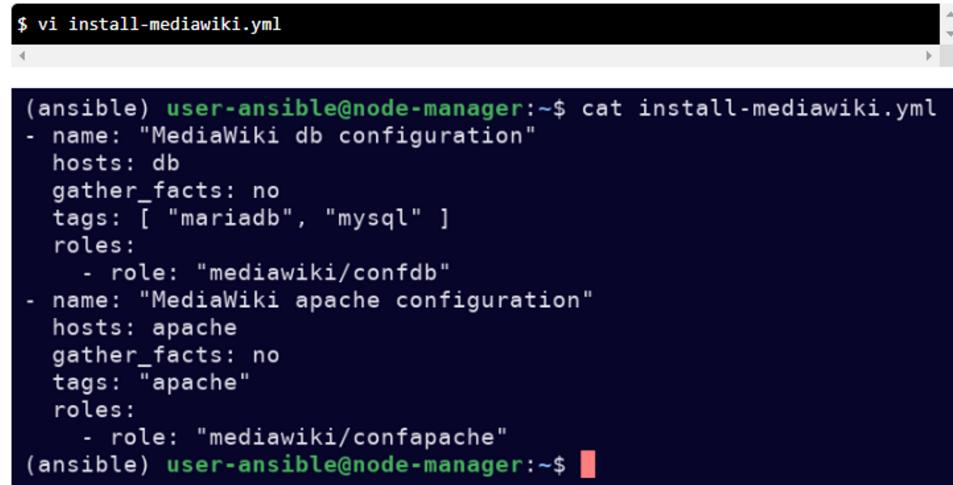
● mariadb.service - MariaDB database server
[~]
Active: active (running)
```

Le service MariaDB est bien actif. L'exécution du playbook s'est bien passée !

Créez le playbook pour configurer MediaWiki

La configuration de MediaWiki va consister à lancer les rôles **confapache** et **confdb**.

Pour cela, vous allez créer un **playbook install-mediawiki.yml**.



```
$ vi install-mediawiki.yml
<
(ansible) user-ansible@node-manager:~$ cat install-mediawiki.yml
- name: "MediaWiki db configuration"
  hosts: db
  gather_facts: no
  tags: [ "mariadb", "mysql" ]
  roles:
    - role: "mediawiki/confdb"
- name: "MediaWiki apache configuration"
  hosts: apache
  gather_facts: no
  tags: "apache"
  roles:
    - role: "mediawiki/confapache"
(ansible) user-ansible@node-manager:~$
```

Capture d'écran de la configuration de MediaWiki

Le **playbook** contient **deux tâches**, une pour chaque rôle. Il est important de commencer par la **configuration de MariaDB car la suite en dépend**. Pour configurer **MediaWiki sur Apache**, la base de données doit déjà être créée. Toutes ces instructions sont contenues dans le guide d'installation de MediaWiki.

Pour chaque tâche est défini le host concerné par les actions, le **gather_facts** est désactivé (pas utile dans notre cas), et des **tags** ont été ajoutés aux tâches.

Lorsque le **playbook** est volumineux, il peut s'avérer utile de pouvoir exécuter une partie spécifique, plutôt que de tout exécuter d'un coup. Pour cela, Ansible prend en charge un attribut « **tags** ». Le ou les tags à exécuter pourront être indiqués dans les options de la commande **ansible-playbook**.

Lancez maintenant le playbook de configuration de MediaWiki :

```
# (ansible) user-ansible@node-manager:~$ ansible-playbook -i inventaire.ini --user user-ansible install-mediawiki.yml
BECOME password:
Vault password:

PLAY [MediaWiki db configuration] ****
TASK [mediawiki/confdb : mediawiki database] ****
ok: [bdd1]

TASK [mediawiki/confdb : mediawiki user and privileges] ****
ok: [bdd1]

PLAY [MediaWiki apache configuration] ****
TASK [mediawiki/confapache : add user-ansible] ****
changed: [http1]

TASK [mediawiki/confapache : mediawiki directory] ****
ok: [http1]

TASK [mediawiki/confapache : uncompress mediawiki archive] ****
ok: [http1]

TASK [mediawiki/confapache : add g+w sur le répertoire mediawiki] ****
changed: [http1]

TASK [mediawiki/confapache : mediawiki configuration] ****
ok: [http1] => (item=http1)

TASK [mediawiki/confapache : mediawiki db update] ****
ok: [http1]

PLAY RECAP
*****
bdd1      : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
http1     : ok=6 changed=2 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

Les tâches sont exécutées séquentiellement, d'abord sur **bdd1** et ensuite sur **http1**. Les actions ont provoqué des changements sur les deux nodes et tout est OK !



Attention, il y a deux jeux d'instructions avec pour cible bdd1 puis http1, ce qui explique que les actions sont d'abord exécutées sur bdd1 puis sur http1 de façon séquentielle.



Si vous lancez votre navigateur et que vous vous rendez sur <http://http1/mediawiki> alors vous pourrez voir MediaWiki en chair et en os !

Votre MediaWiki fonctionnel !

Voilà, avec 3 lignes de commande Ansible, **vous avez déployé MediaWiki**. Vous pourrez relancer à volonté ces commandes sur votre infrastructure. Vous pourrez même remodeler vos scripts pour les adapter à **vos futurs déploiements**.

Résumé :

Dans ce chapitre, vous avez assemblé les opérations et automatisé le déploiement de MediaWiki :

- vous avez vérifié que le niveau de sécurité sur les nodes n'était pas trop élevé pour permettre le déploiement sans blocage ;
- vous avez configuré la résolution de noms sur les nodes, pour que leur nom de machine puisse être utilisé dans les fichiers de configuration ;
- vous avez créé 3 playbooks peuplés de tâches et de rôles ;
- vous avez lancé dans un ordre précis les playbooks avec la commande ansible-playbook afin de déployer automatiquement MediaWiki ;
- vous avez finalement vérifié que MediaWiki était bien installé et disponible sur <http://http1/mediawiki>

Dans la dernière partie, vous irez un peu plus loin avec Ansible, en créant votre propre module Ansible personnalisé.

[Créez votre propre module Ansible personnalisé avec Python](#)



Dans la deuxième partie, vous avez créé et utilisé des **playbooks** et des rôles pour gérer le déploiement automatique de **MediaWiki**.

Dans ce chapitre, vous allez créer un module Ansible personnalisé. En effet, vous allez créer un module capable de compter le nombre de pages MediaWiki en interrogeant directement la base de données.

Pourquoi écrire un module ?

Il existe déjà beaucoup de modules et la collection s'enrichit régulièrement. Mais dans certains cas, il est nécessaire d'aller plus loin, comme dans votre cas : le module pour compter le nombre de pages dans un **MediaWiki n'existe pas**. Dans ce cas, il est préférable de faire un module spécifique pour répondre directement à ce besoin.

Dans votre cas, vous allez créer un module qui va exécuter une requête SQL. Cette requête comptera le nombre de pages dans la base MediaWiki.

Pour créer le module, vous aurez besoin de :

- écrire un script Python
- créer un répertoire pour mettre le script
- tester le module
- documenter le module.

Comment faire un module ?

Ansible gère automatiquement toutes les étapes de fonctionnement d'un module.

En effet, quand un module est utilisé, Ansible a besoin de faire plusieurs actions, comme aller lire le fichier de configuration pour trouver l'emplacement des modules, ou établir une connexion avec les nodes, ou encore créer un fichier autoporteur et le déposer sur les nodes avant de l'exécuter.

Le fichier autoporteur est un mini programme compressé qui contient le module. C'est une manière qu'Ansible a trouvée pour optimiser le temps de transfert et contrôler l'exécution d'un module.

Donc, pour créer un module, il suffit simplement de mettre votre programme dans le répertoire library et d'écrire un programme Python en utilisant la classe **AnsibleModule**.

Les caractéristiques de [la classe AnsibleModule sont disponibles sur la documentation](#).

Emplacement du module

Comme indiqué plus haut, vous allez créer un répertoire **library** au même niveau que les **playbooks**, c'est-à-dire dans votre environnement de travail.

Dans ce répertoire, vous allez créer votre programme Python, que vous allez appeler **count_page.py**.

Ansible ira regarder dans ce répertoire lorsque le module **count_page** sera appelé par le **playbook**.

Au préalable, connectez-vous sur le node manager avec l'utilisateur user-ansible et activez l'environnement virtuel :

```
$ source ansible/bin/activate
```

Puis créez le répertoire library :

```
$ mkdir library
```

```
mot de passe :  
user-ansible@nodeManager:~$ source ansible/bin/activate  
(ansible) user-ansible@nodeManager:~$ mkdir library  
(ansible) user-ansible@nodeManager:~$ ls library  
(ansible) user-ansible@nodeManager:~$ ls -la  
total 68  
drwxr-xr-x 10 user-ansible user-ansible 4096 6 oct. 15:40 .  
drwxr-xr-x 4 root root 4096 3 oct. 13:33 ..  
-rw-r--r-- 1 user-ansible user-ansible 0 3 oct. 18:29 {'  
drwxr-xr-x 5 user-ansible user-ansible 4096 4 oct. 14:14 .ansible  
drwxr-xr-x 4 user-ansible user-ansible 4096 3 oct. 13:48 ansible  
-rw----- 1 user-ansible user-ansible 2295 6 oct. 12:26 .bash_history  
-rw-r--r-- 1 user-ansible user-ansible 220 3 oct. 13:33 .bash_logout  
-rw-r--r-- 1 user-ansible user-ansible 3526 3 oct. 13:33 .bashrc  
drwxr-xr-x 3 user-ansible user-ansible 4096 3 oct. 13:49 .cache  
-rw-r--r-- 1 user-ansible user-ansible 106 6 oct. 11:47 install-apache.yml  
-rw-r--r-- 1 user-ansible user-ansible 99 6 oct. 12:02 install-mariadb.yml  
-rw-r--r-- 1 user-ansible user-ansible 31 4 oct. 14:54 inventaire.ini  
drwxr-xr-x 2 user-ansible user-ansible 4096 6 oct. 15:40 library  
drwxr-xr-x 3 user-ansible user-ansible 4096 3 oct. 13:48 .local  
-rw-r--r-- 1 user-ansible user-ansible 807 3 oct. 13:33 .profile  
drwxr-xr-x 5 user-ansible user-ansible 4096 4 oct. 14:39 roles  
drwx----- 2 user-ansible user-ansible 4096 3 oct. 19:21 .ssh  
drwxr-xr-x 4 user-ansible user-ansible 4096 3 oct. 14:35 test  
(ansible) user-ansible@nodeManager:~$
```

Écrivez le programme Python

Vous allez écrire le programme qui sera exécuté par le module.

Vous allez structurer votre programme de la façon suivante :

- **Créer un en-tête.**
- **Définir et gérer les arguments.**
- **Établir une connexion avec la base de données pour exécuter la requête SQL.**
- **Et transmettre les résultats à Ansible.**

Éditez le fichier count_page.py :

```
$ nano library/count_page.py
```

Créez l'en-tête

L'en-tête du programme indique l'**interpréteur à utiliser pour exécuter le programme**, et le **type d'encodage du fichier**. La majorité des programmes Python commencent de cette façon :

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

Définition des arguments du module

Vous allez utiliser **deux arguments** pour indiquer au module **le nom de la base de données et la requête à utiliser** :

- **db_name** : nom de la base de données
- **request** : requête SQL à lancer.

Pour intégrer ces arguments au programme, vous avez besoin de :

- **importer la classe AnsibleModule**
- **définir la fonction main()**
- créer une instance de la classe **AnsibleModule** avec son constructeur
- utiliser le paramètre **argument_spec** pour définir les arguments pris en charge par le module, ainsi que leurs types.

```
from ansible.module_utils.basic import AnsibleModule

def main():
    module = AnsibleModule(
        argument_spec=dict(
            db_name    = dict(required=True, type='str'),
            request   = dict(required=True, type='str'),
        )
    )
```

Récupération de la valeur des arguments

Vous allez maintenant **récupérer la valeur des arguments** à l'aide des **fonctions params et gets** de l'instance **module**, et les placer dans deux variables qui portent le même nom que les arguments.

```
db_name_local = module.params.get('srvSql')
request_local = module.params.get('request')
```

Connexion à la base

Pour la connexion à la base, vous avez besoin de :

- **importer la bibliothèque MySQLdb**
- **définir un objet de connexion**
- **définir un curseur sur l'objet**
- **exécuter une requête sur le curseur**
- **stocker le résultat de la requête dans une variable**
- **fermer la connexion.**

```
import MySQLdb
```

```
db = MySQLdb.connect(db=db_name_local)
cur = db.cursor()
cur.execute(request_local)
results = cur.fetchall()
db.close()
```

```
db = MySQLdb.connect(db=db_name_local)
cur = db.cursor()
cur.execute(request_local)
results = cur.fetchall()
db.close()
```

Transmission du résultat à Ansible

La **transmission du résultat** se fait en utilisant la fonction **module.exit_json** avec les arguments **changed** et **resultat** qui indiquent respectivement l'**état de l'action** et le **résultat de la requête**.

Le programme se termine avec l'appel à la fonction **main()**.

```
module.exit_json(changed=False, resultat=results)
```

```
if __name__ == "__main__":
    main()
```

```
if __name__ == "__main__":
    main()
```

Test du module

Maintenant que tout est au point, vous allez pouvoir **tester votre module**.

Vous allez créer le **playbook** qui utilisera le module en question.

```
- $ nzno module.yml
```

À la différence des **playbooks précédents** qui utilisaient les **rôles**, celui-ci fait appel à une **section tasks**, qui contiendra un appel au module **count_page** :

```
(ansible) user-ansible@node-manager:~$ cat module.yml
---
- name: "Requete dans une base"
  hosts: bdd1
  gather_facts: no
  tasks:
    - name: "compte le nombre de pages dans le Wiki"
      count_page:
        db_name: "mediawiki"
        request: "select count(*) from page;"
        register: resultat
    - debug: var=resultat
(ansible) user-ansible@node-manager:~$
```

Capture d'écran du test du module

- **name:** "Requete dans une base" indique le nom du jeu d'instructions ;
- **hosts:** bdd1 indique le nom du node sur lequel sera exécutée l'action ;
- **gather_facts:** no indique de désactiver les facts ;
- **tasks:** indique que la section contient des tâches :
- **name:** "compte le nombre de pages dans le Wiki" indique le nom de la tâche ;
- **count_page:** indique le module à utiliser,
- **db_name:** "mediawiki" indique le nom de la base de données,
- **request:** "select count(*) from page;" indique la requête à exécuter ,
- **register:** resultat indique de stocker le résultat de la requête dans la variable request ;
- **debug: var=resultat** indique d'afficher le résultat de la requête contenue dans la variable result.

Le nom du fichier **count_page.py** correspond au nom du module **count_page**.

Lancez maintenant le playbook **module.yml** avec la commande suivante :

ansible-playbook -i inventaire.ini module.yml -K -b

Sont utilisées ici les options :

- **-i** pour l'inventaire à utiliser
- **-k** pour le password sudo
- **-b** pour passer en sudo.

```
(ansible) user-ansible@node-manager:~$ ansible-playbook -i inventaire.ini module.yml -K -b
BECOME password:

PLAY [Requete dans une base] *****

TASK [compte le nombre de pages dans le Wiki] *****
ok: [bdd1]

TASK [debug] *****
ok: [bdd1] => {
    "resultat": [
        "ansible_facts": {
            "discovered_interpreter_python": "/usr/bin/python3"
        },
        "changed": false,
        "failed": false,
        "resultat": [
            [
                1
            ]
        ]
    }
}

PLAY RECAP *****
bdd1 : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignore
```

Gestion de l'aide

Pour agrémenter votre code, vous allez **intégrer de la documentation**. Ainsi, en utilisant la commande **ansible-doc**, vous pourrez donner des informations utiles pour l'utilisation de votre module.

Vous allez ajouter 3 déclarations dans votre programme :

- **DOCUMENTATION** qui contiendra le **nom du module, la description et les options**.
- **EXAMPLES** qui contiendra un exemple pour **utiliser correctement votre module**.
- **RETURN** qui contiendra la description du résultat retourné par votre module.

```
# DOCUMENTATION='''  
module: count_page  
author: Alexandre  
description: Module qui permet d'exécuter une requête SQL  
  
options:  
    db_name:  
        description: nom de la base de données  
        required: yes  
    request:  
        description: requête à exécuter  
        required: yes  
    ...  
  
EXAMPLES='''  
- name: "SQL"  
  count_page:  
    db_name: "BDD"  
    request: "select * from user;"  
  ...  
  
RETURN = '''  
resultat:  
    description: retourne le résultat de la requête  
  ...
```

Pour vérifier ce que ça donne, vous allez utiliser la commande **ansible-doc** avec l'option **-M** pour indiquer le **chemin du module**, suivi du **nom du module** :

```
(ansible) user-ansible@node-manager:~$ ansible-doc -M library count_page

> COUNT_PAGE (/home/user-ansible/library/count_page.py)

Module qui permet d'exécuter une requête SQL

OPTIONS (= is mandatory):

= db_name
    nom de la base de données

= request
    requête à exécuter

AUTHOR: Alexandre

EXAMPLES:

- name: "SQL"
  count_page:
  db_name: "BDD"
  request: "select * from user;"


RETURN VALUES:

resultat:
    retourne le résultat de la requête
```

Fichier dans son ensemble :

```
#!/usr/bin/python

from ansible.module_utils.basic import AnsibleModule
import MySQLdb

def main():
    module = AnsibleModule(
        argument_spec=dict(
            db_name = dict(required=True, type='str'),
            request = dict(required=True, type='str'),
        )
    )

    db_name_local = module.params.get('db_name')
    request_local = module.params.get('request')

    db = MySQLdb.connect(db=db_name_local)
    cur = db.cursor()
    cur.execute(request_local)
    results = cur.fetchall()
    db.close()

    module.exit_json(changed=False, resultat=results)

if __name__ == "__main__":
    main()
```

```
DOCUMENTATION=""
module: count_page
author: Alexandre
description: Module qui permet d'exécuter une requête SQL
```

```
options:
  db_name:
    description: nom de la base de données
    required: yes
  request:
    description: requête à exécuter
    required: yes
```

...

```
EXAMPLES=""
- name: "SQL"
```

```
count_page:  
    db_name: "BDD"  
    request: "select * from user;"  
...  
  
RETURN = ""  
resultat:  
    description: retourne le résultat de la requête  
...  
...
```

Résumé

Dans ce chapitre, vous avez créé votre propre module Ansible :

- vous avez écrit un programme Python en utilisant la classe `AnsibleModule` ;
- vous avez construit votre programme grâce à plusieurs sections qui vous ont permis de définir des arguments, de vous connecter à la base de données, de lancer une requête SQL et de transmettre le résultat à Ansible ;
- vous avez ensuite placé votre programme dans le répertoire `library`, afin qu'il soit pris en compte par Ansible ;
- vous avez ensuite testé votre module à l'aide d'un `playbook` ;
- et pour finir, vous avez agrémenté votre programme avec de la documentation, ce qui permet de savoir comment utiliser votre module.