**Utilisez Ansible pour automatiser vos tâches de configuration**

Vous êtes administrateur système et vous en avez assez des **tâches répétitives** ? Vous souhaitez mettre en place des pratiques DevOps d'**automatisation** dans vos déploiements et vos configurations ? Ce cours est fait pour vous !

**Ansible** est un outil de gestion de configuration qui permet d'automatiser des tâches avec des scripts d'automatisation.



**Plan du cours**

1. Dans la première partie de ce cours, vous apprendrez à identifier ce que vous pouvez automatiser et comment le découper en **tâches distinctes**. Puis, vous **installerez** un environnement d'automatisation fonctionnel avec Ansible.
2. Dans la deuxième partie, vous apprendrez à rédiger les scripts de configuration de tâches avec les **modules** en utilisant la syntaxe **YAML**, et vous les pousserez sur vos serveurs avec les **playbooks**.
3. Enfin, dans la troisième partie, nous verrons un exemple de **module personnalisé**et comment le créer vous-même.

Tout au long du cours, nous nous baserons sur **un cas concret** qui vous permettra de **mettre en pratique** ce que vous apprendrez !

Vous êtes prêt à **gagner du temps** en automatisant vos tâches ? Alors c'est parti !

**À la fin de ce cours, vous serez capable de :**

* installer et configurer un node manager Ansible ;
* automatiser le déploiement d'une application avec Ansible ;
* créer un module Ansible personnalisé avec Python.

**Prérequis**

Pour suivre ce cours, vous devez être familier avec les **systèmes Linux**, notamment l'utilisation d'un **terminal** en ligne de commande et la gestion des **droits d'accès**.

Vous n'avez pas besoin de connaître la syntaxe YAML ou le Python, tout vous sera expliqué. :)

## Partie 1 - Automatisez vos opérations d'installation et de configuration

### Identifiez ce que vous pouvez automatiser

### Installez Ansible dans votre environnement

### Préparez la communication avec les nodes

Quiz : Testez vos compétences sur Ansible et son installation !

## Partie 2 - Déployez automatiquement une application avec Ansible

### Organisez vos opérations de déploiement

### Contrôlez l'exécution des opérations et enchaînez plusieurs actions

### Assemblez les opérations avec les playbooks pour automatiser le déploiement

Quiz : Testez vos compétences sur l'automatisation de déploiement avec Ansible !

## Partie 3 - Créez votre propre module Ansible personnalisé avec Python

### Créez votre propre module Ansible personnalisé avec Python

## Identifiez ce que vous pouvez automatiser

### Faites connaissance avec votre nouvelle entreprise

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 !

#### Quel sera votre rôle ?

Vous êtes ingénieur DevOps chez une entreprise. 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**.

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

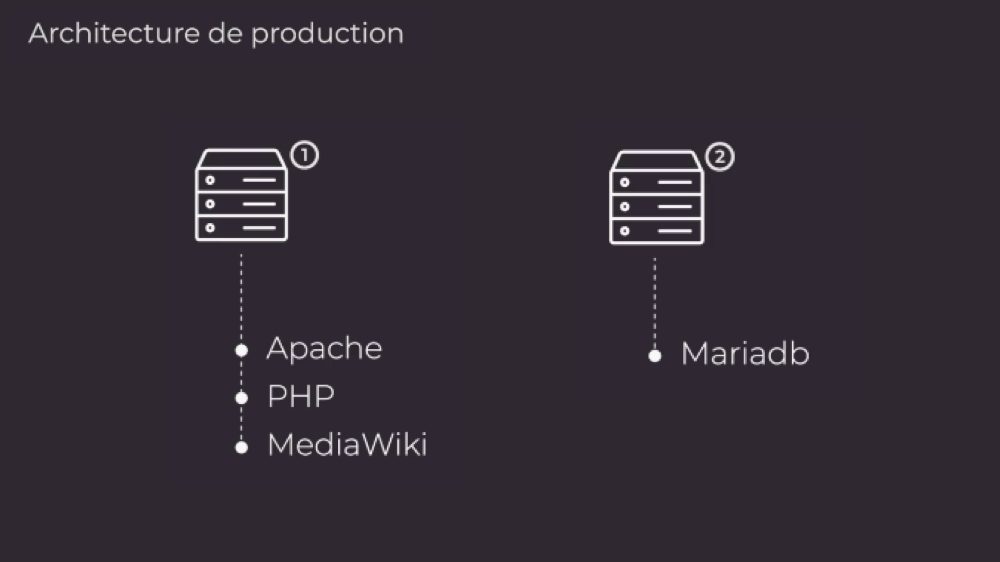
1. Installer un **serveur web** pour servir les pages à un navigateur web.
2. Installer **PHP** pour exécuter le logiciel.
3. Installer une **base de données** pour stocker les pages.
4. Télécharger les **fichiers sources** de MediaWiki et les mettre sur le serveur web.
5. Configurer le serveur web pour pointer vers l’**URL MediaWiki**.
6. 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 (**Centos**) avec un service web (**Apache, PHP**) et un serveur Linux (**Centos**)  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 :

Architecture de production du wiki à mettre en place

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’adminsys en chef le soin de mettre en place l’infrastructure des serveurs ; c’est-à-dire **installer Linux** (Centos) et mettre en place le **réseau entre les deux serveurs**.

L’adminsys vous donnera alors un compte **root** sur les 2 serveurs et leurs **adresses IP**.

Vous allez donc devoir automatiser les étapes suivantes avec **Ansible** :

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

Il faudra donc automatiser les **6 étapes** nécessaires à l’installation de **MediaWiki**.

### 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 le premier chapitre**, vous avez identifié la façon de s'approprier l’installation d’un wiki et comment aborder l’automatisation de son déploiement : en décomposant les étapes **d’architecture**, **d’installation** et de **configuration**. En finissant par  faire le choix d’un outil d’automatisation adapté à vos besoins : **Ansible**.

**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.

### Découvrez Ansible

#### 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**.

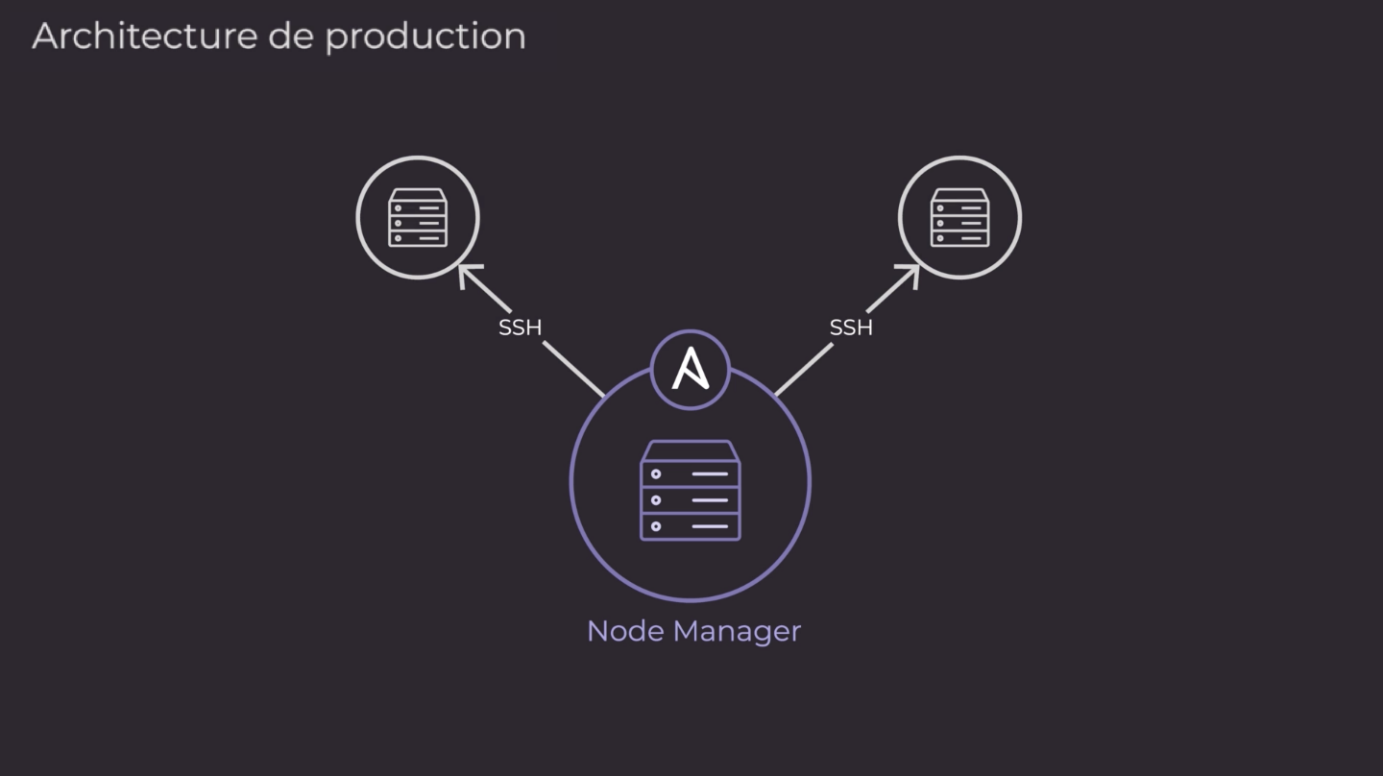
Red Hat a été racheté par **IBM** en **2018**. Donc, Ansible appartient désormais à IBM.

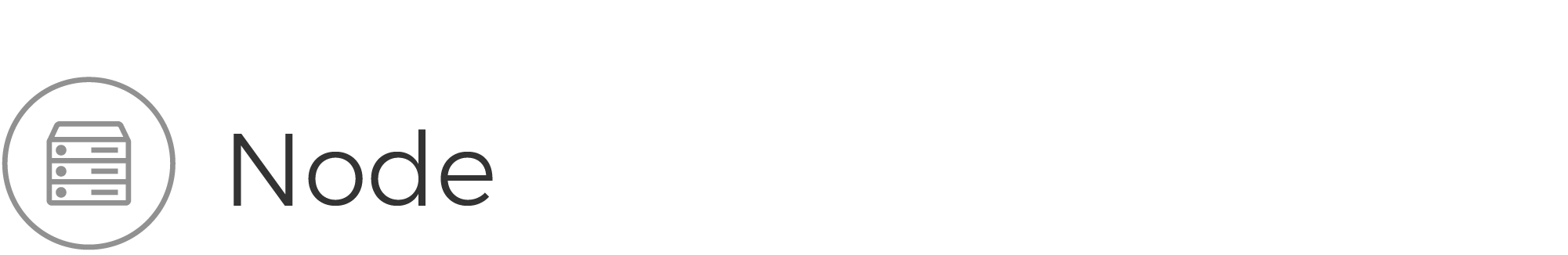
#### Complétez votre architecture technique pour Ansible

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 :





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.



Un **node manager**, ou **control node**, est un poste qui contrôle les nodes grâce à sa connexion SSH. Il a 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 9**, 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** : 192.168.122.10 (Centos 7) ;
* **serveur 1** : 192.168.122.11  (Centos 7) ;
* **serveur 2** : 192.168.122.12 (Centos 7).

À ce stade, vous devez avoir **3 serveurs** disponibles (1 Centos 7 et 2 Centos 7) qui communiquent sur le sous-réseau 192.168.122.0/24 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.

### Installez 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 **centos**) 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](https://releases.ansible.com/ansible/) 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 privilégie 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**:

$ ssh root@192.168.122.10

root@192.168.122.10's password:

The programs included with the Debian GNU/Linux system are free software;

the exact distribution terms for each program are described in the

individual files in /usr/share/doc/\*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent

permitted by applicable law.

Last login: Sun May 5 21:33:24 2019 from 192.168.122.1

root@node-manager:~#

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.

$ yum install python-virtualenv sshpass

[~]

Lecture des listes de paquets... Fait

Construction de l'arbre des dépendances

Lecture des informations d'état... Fait

The following additional packages will be installed:

python-pip-whl python-pkg-resources python3-virtualenv virtualenv

Paquets suggérés :

python-setuptools

Les NOUVEAUX paquets suivants seront installés :

python-pip-whl python-pkg-resources python-virtualenv python3-virtualenv virtualenv

0 mis à jour, 5 nouvellement installés, 0 à enlever et 21 non mis à jour.

Il est nécessaire de prendre 1 711 ko dans les archives.

Après cette opération, 2 501 ko d'espace disque supplémentaires seront utilisés.

Souhaitez-vous continuer ? [O/n]

À la question "Continuer ?", répondez **0ui**, et l’installation de **python-virtualenv** et **sshpass** se lance.

#### 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** :

# useradd 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 »...

Entrez le nouveau mot de passe UNIX :

Retapez le nouveau mot de passe UNIX :

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 ? [O/n]

Entrez un mot de passe, confirmez-le, complétez les informations demandées, puis validez avec **Oui**.

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

# su - user-ansible

user-ansible@node-manager:~$

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 **version 2.7.10** (la dernière stable à l'écriture de ces lignes).

Vous créez alors un environnement de travail virtuel nommé **ansible2.7.10**.

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@node-manager:~$ virtualenv ansible2.7.10

Running virtualenv with interpreter /usr/bin/python2

New python executable in /home/user-ansible/ansible2.7.10/bin/python2

Also creating executable in /home/user-ansible/ansible2.7.10/bin/python

Installing setuptools, pkg\_resources, pip, wheel...done.

Vous utilisez la commande **virtualenv** pour créer l’environnement **ansible2.17.10**, dans lequel les outils, les ressources et le gestionnaire de paquets sont installés.

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

$ source ansible2.7.10/bin/activate

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

Le prompt a changé au profit de  (ansible2.7.10) user-ansible@node-manager:~$**.** Ce qui signifie que vous êtes dans l'environnement de travail virtuel **Ansible2.7.10**.

### Installez Ansible dans votre environnement virtuel

Installez maintenant **Ansible2.7.0** avec **pip** avec la commande suivante :

(ansible2.7.10) user-ansible@node-manager:~$ pip install ansible==2.7.10

DEPRECATION: Python 2.7 will reach the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7 won't be maintained after that date. A future version of pip will drop support for Python 2.7.

Collecting ansible==2.7.10

Downloading https://files.pythonhosted.org/packages/9a/9d/5e3d67bd998236f32a72f255394eccd1e22b3e2843aa60dc30dd164816d0/ansible-2.7.10.tar.gz (11.8MB)

|████████████████████████████████| 11.8MB 3.5MB/s

[~]

[~]

Downloading Successfully built ansible PyYAML pycparser

Installing collected packages: MarkupSafe, jinja2, PyYAML, six, pycparser, cffi, pynacl, enum34, asn1crypto, ipaddress, cryptography, bcrypt, pyasn1, paramiko, ansible

Successfully installed MarkupSafe-1.1.1 PyYAML-5.1 ansible-2.7.10 asn1crypto-0.24.0 bcrypt-3.1.6 cffi-1.12.3 cryptography-2.6.1 enum34-1.1.6 ipaddress-1.0.22 jinja2-2.10.1 paramiko-2.4.2 pyasn1-0.4.5 pycparser-2.19 pynacl-1.3.0 six-1.12.0

Attention de bien vérifier que le prompt est bien **(ansible2.7.10) user-ansible@node-manager:~$** avant de lancer les commandes.

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

(ansible2.7.10) user-ansible@node-manager:~$ ansible --version

ansible 2.7.10

Tout est OK, Ansible est installé !

### Vérifiez votre installation

Dans le répertoire **bin** de votre environnement virtuel, vous pouvez constater que **10 outils** Ansible sont installés :

$ ls ansible2.7.10/bin/ansible\* -l

ansible

ansible-config

ansible-connection

ansible-console

ansible-doc

ansible-galaxy

ansible-inventory

ansible-playbook

ansible-pull

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/python2.7/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.

### 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 :

1. Définir un **inventaire** des nodes.
2. Vérifier la **connexion SSH** entre le node manger et les nodes.
3. Lancer un **ping** avec Ansible sur les nodes.
4. Vérifier que **Python** est installé sur les nodes.
5. Créer un **simple utilisateur** sur les nodes.
6. Attribuer les **droits sudo** à cet utilisateur.
7. Créer une **paire de clés SSH** pour cet utilisateur.
8. Copier la **clé publique SSH** sur les nodes.

### Définissez un inventaire des nodes

Il est temps de baptiser vos serveurs :

* sur le serveur 1 seront installés **Apache, PHP et MediaWiki** ;nous l'appellerons donc **http1** ;
* sur le serveur 2 sera installé **MariaDB** ; nous l'appellerons donc **bdd1**.

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**:

$ ssh root@192.168.122.10

root@192.168.122.10's password:

Linux node-manager 4.9.0-6-amd64 #1 SMP Debian 4.9.88-1+deb9u1 (2018-05-07) x86\_64

The programs included with the Debian GNU/Linux system are free software;

the exact distribution terms for each program are described in the

individual files in /usr/share/doc/\*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent

permitted by applicable law.

Last login: Sun May 5 21:33:24 2019 from 192.168.122.1

root@node-manager:~#

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

127.0.0.1 localhost

127.0.1.1 node-manager

# le node http1

192.168.122.11 http1

# le node bdd1

192.168.122.12 bdd1

Cette action vous permettra d’utiliser les noms **http1** et **bdd1** 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 :

$ source ansible2.7.10/bin/activate

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

Puis éditez avec la commande **vi** le fichier **inventaire.ini** :

(ansible2.7.10) user-ansible@node-manager:~$ vi inventaire.ini

Mettez dans ce fichier les 2 noms des nodes :

http1

bdd1

Vi n’est pas obligatoire ; si vous êtes plus à l’aise avec un autre éditeur, utilisez-le !

Le fichier **inventaire** est au **format INI** par défaut, il suit donc [la syntaxe de ce format](https://fr.wikipedia.org/wiki/Fichier_INI). 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](https://docs.ansible.com/ansible/latest/plugins/inventory.html).

### 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**.

$ ssh root@bdd1

The authenticity of host 'bdd1 (192.168.122.12)' can't be established.

ECDSA key fingerprint is SHA256:oo8D6y+dItnQPSZSc1hJGCbgOZAKkt9Hn8z3NKCRMRY.

Are you sure you want to continue connecting (yes/no)?

Répondez **yes** et faites la même chose pour **http1**.

### 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](https://docs.ansible.com/ansible/latest/user_guide/intro_adhoc.html) pour approfondir.

Lancez maintenant un **ping** avec **Ansible** dans votre **environnement de travail virtuel** :

$ ansible -i inventaire.ini -m ping http1 --user root --ask-pass

SSH password:

http1 | SUCCESS => {

"changed": false,

"ping": "pong"

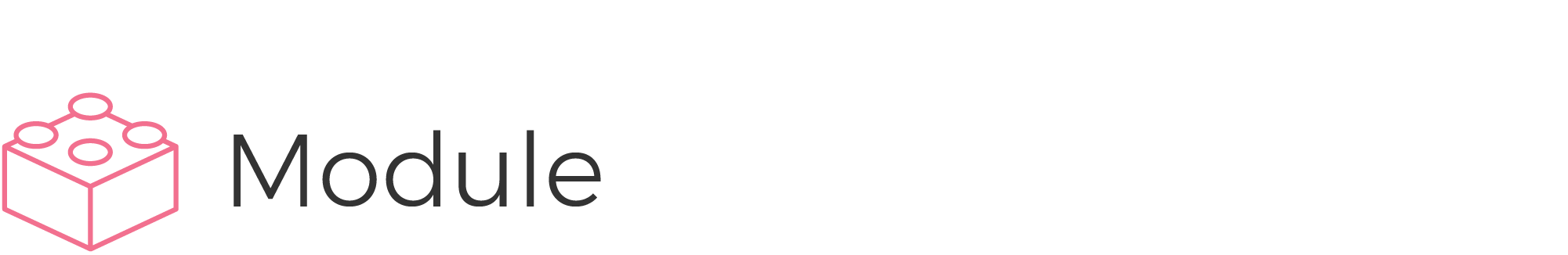
}

Vous venez d’utiliser la commande **ansible** avec plusieurs options :

* **-i** : indique à Ansible l’emplacement du fichier **i**nventaire ;
* **-m** : indique à Ansible d’utiliser le **m**odule 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.

**

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](https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html) ou avec la commande  $ ansible-doc --list.

Tous les modules officiels d’Ansible sont téléchargés sur votre poste lors de l’installation d’Ansible. Lorsque vous utilisez un module, Ansible ira chercher le code à exécuter dans le dossier du module sur votre poste. Si vous avez installé Ansible dans un environnement virtuel avec pip, les modules se trouvent à l’adresse  ./ansible2.7.10/lib/python2.7/site-packages/ansible.

La [syntaxe et les options de la commande ansible](https://docs.ansible.com/ansible/latest/cli/ansible.html) sont disponibles sur la documentation d'Ansible.

### Vérifiez que Python est installé sur les nodes

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 "yum install -y python2" http1 --user root --ask-pass

SSH password:

http1 | CHANGED | rc=0 >>

Modules complémentaires chargés : fastestmirror

Loading mirror speeds from cached hostfile

\* base: centos.mirrors.benatherton.com

\* extras: mirror.in2p3.fr

\* updates: ftp.pasteur.fr

Le paquet python-2.7.5-77.el7\_6.x86\_64 est déjà installé dans sa dernière version

Rien à faire

Shared connection to http1 closed.

Vous avez utilisé l’option **-m** pour appeler le module **raw** avec l’argument **“yum install -y python2”**, qui a pour effet d’installer Python en version 2 sur le node **http1**.

Depuis sa version 2.2, Ansible est compatible avec Python 3 (l'adaptation des modules est en cours). Au niveau des versions minimales, vous devez disposer d’une version 3.5 pour Python 3 et 2.6 pour Python 2.

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', 'sceretsalt') }}"

localhost | SUCCESS => {

"msg": "$6$sceretsalt$tBcfGEgifQpQZsg5CIGZ79XC55h5vHy7UWrys7cAF37KNCQQbm7iCvy58MlLQLaS2fLF6ZjqDVHhVrkMdRi0f0"

}

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**. “Sceretsalt” 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=$6$sceretsalt$tBcfGEgifQpQZsg5CIGZ79XC55h5vHy7UWrys7cAF37KNCQQbm7iCvy58MlLQLaS2fLF6ZjqDVHhVrkMdRi0f0' --user root --ask-pass all

bdd1 | SUCCESS => {

"append": false,

"changed": false,

"comment": "",

"group": 1000,

"home": "/home/user-ansible",

"move\_home": false,

"name": "user-ansible",

"password": "NOT\_LOGGING\_PASSWORD",

"shell": "/bin/bash",

"state": "present",

"uid": 1000

}

http1 | SUCCESS => {

"append": false,

"changed": false,

"comment": "",

"group": 1000,

"home": "/home/user-ansible",

"move\_home": false,

"name": "user-ansible",

"password": "NOT\_LOGGING\_PASSWORD",

"shell": "/bin/bash",

"state": "present",

"uid": 1000

}

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 **http1**, et regardez le fichier **/etc/sudoers** ;il contient  les configurations de **sudo** :

Vous pouvez voir la ligne suivante :

## Allows people in group wheel to run all commands

%wheel ALL=(ALL) ALL

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

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

SSH password:

http1 | CHANGED => {

"append": true,

"changed": true,

"comment": "",

"group": 1000,

"groups": "wheel",

"home": "/home/user-ansible",

"move\_home": false,

"name": "user-ansible",

"shell": "/bin/bash",

"state": "present",

"uid": 1000

}

bdd1 | CHANGED => {

"append": true,

"changed": true,

"comment": "",

"group": 1000,

"groups": "wheel",

"home": "/home/user-ansible",

"move\_home": false,

"name": "user-ansible",

"shell": "/bin/bash",

"state": "present",

"uid": 1000

}

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=wheel append=yes ' --user user-ansible --ask-pass --become --ask-become-pass all

SSH password:

SUDO password[defaults to SSH password]:

bdd1 | SUCCESS => {

"append": true,

"changed": false,

"comment": "",

"group": 1000,

"groups": "wheel",

"home": "/home/user-ansible",

"move\_home": false,

"name": "user-ansible",

"shell": "/bin/bash",

"state": "present",

"uid": 1000

}

http1 | SUCCESS => {

"append": true,

"changed": false,

"comment": "",

"group": 1000,

"groups": "wheel",

"home": "/home/user-ansible",

"move\_home": false,

"name": "user-ansible",

"shell": "/bin/bash",

"state": "present",

"uid": 1000

}

La commande s’est bien passée, car le retour est un **SUCCESS** ! Vous avez les privilèges **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 mettre en utilisateur **user-ansible** avec la commande suivante :

# su - user-ansible

user-ansible@node-manager:~$

Et lancez la création des clés avec la commande suivante :

user-ansible@node-manager:~$ ssh-keygen -t ecdsa

Generating public/private ecdsa key pair.

Enter file in which to save the key (/home/user-ansible/.ssh/id\_ecdsa):

Created directory '/home/user-ansible/.ssh'.

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:geBOUCBR+4ljdMOn5QVmq9R7dpgmwYsAv995BtTWCYg user-ansible@node-manager

The key's randomart image is:

+---[ECDSA 256]---+

|+o+oo.+..    |

| + =E\*.= o . |

|  = B X = o  |

| . X @ \* +   |

|  = B \* S .  |

| . o . B . |

| . o o    |

|   o     |

|             |

+----[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

SSH password:

SUDO password[defaults to SSH password]:

http1 | CHANGED => {

"changed": true,

"comment": null,

"exclusive": false,

"follow": false,

"key": "ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBCqrZM+NlkMaOToUnfSup0jqzA6MuMy2K9L5bIDzBPgD2o6iNzS2ze+SNrkhVA5J58SCwFRPVY+54ZhAmly4gHY= user-ansible@node-manager",

"key\_options": null,

"keyfile": "/home/user-ansible/.ssh/authorized\_keys",

"manage\_dir": true,

"path": null,

"state": "present",

"unique": false,

"user": "user-ansible",

"validate\_certs": true

}

bdd1 | CHANGED => {

"changed": true,

"comment": null,

"exclusive": false,

"follow": false,

"key": "ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBCqrZM+NlkMaOToUnfSup0jqzA6MuMy2K9L5bIDzBPgD2o6iNzS2ze+SNrkhVA5J58SCwFRPVY+54ZhAmly4gHY= user-ansible@node-manager",

"key\_options": null,

"keyfile": "/home/user-ansible/.ssh/authorized\_keys",

"manage\_dir": true,

"path": null,

"state": "present",

"unique": false,

"user": "user-ansible",

"validate\_certs": true

}

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 pass 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

SUDO password:

http1 | SUCCESS => {

"changed": false,

"comment": null,

"exclusive": false,

"follow": false,

"key": "ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBCqrZM+NlkMaOToUnfSup0jqzA6MuMy2K9L5bIDzBPgD2o6iNzS2ze+SNrkhVA5J58SCwFRPVY+54ZhAmly4gHY= user-ansible@node-manager",

"key\_options": null,

"keyfile": "/home/user-ansible/.ssh/authorized\_keys",

"manage\_dir": true,

"path": null,

"state": "present",

"unique": false,

"user": "user-ansible",

"validate\_certs": true

}

bdd1 | SUCCESS => {

"changed": false,

"comment": null,

"exclusive": false,

"follow": false,

"key": "ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBCqrZM+NlkMaOToUnfSup0jqzA6MuMy2K9L5bIDzBPgD2o6iNzS2ze+SNrkhVA5J58SCwFRPVY+54ZhAmly4gHY= user-ansible@node-manager",

"key\_options": null,

"keyfile": "/home/user-ansible/.ssh/authorized\_keys",

"manage\_dir": true,

"path": null,

"state": "present",

"unique": false,

"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 :

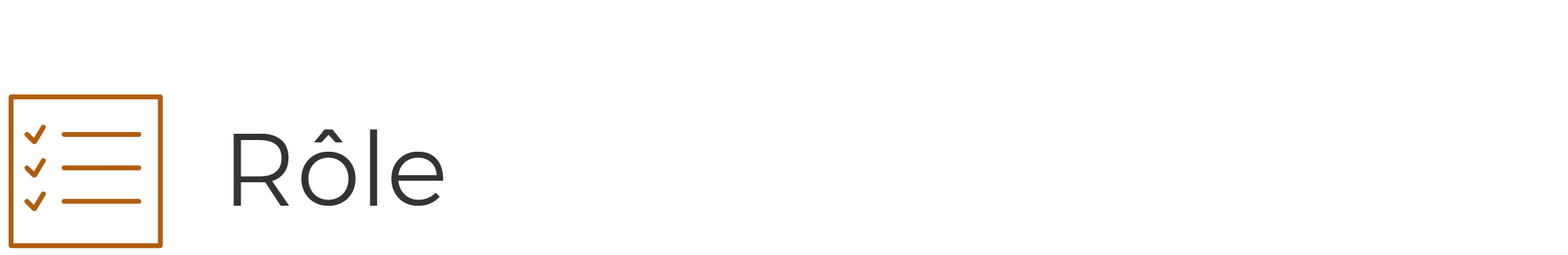
* créé un fichier **inventaire** et ajouté les **nodes** dans ce fichier ;
* utilisé la commande **ansible** en mode **ad-hoc** pour vérifier la présence de **Python** sur les nodes et lancer un **ping** ;
* créé un utilisateur **user-ansible** sur les **nodes**;
* créé des **clés SSH** pour l'utilisateur **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.

## Organisez vos opérations de déploiement

**Dans la première partie**, vous avez pris connaissance du **contexte** dans lequel vous alliez 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**.



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.

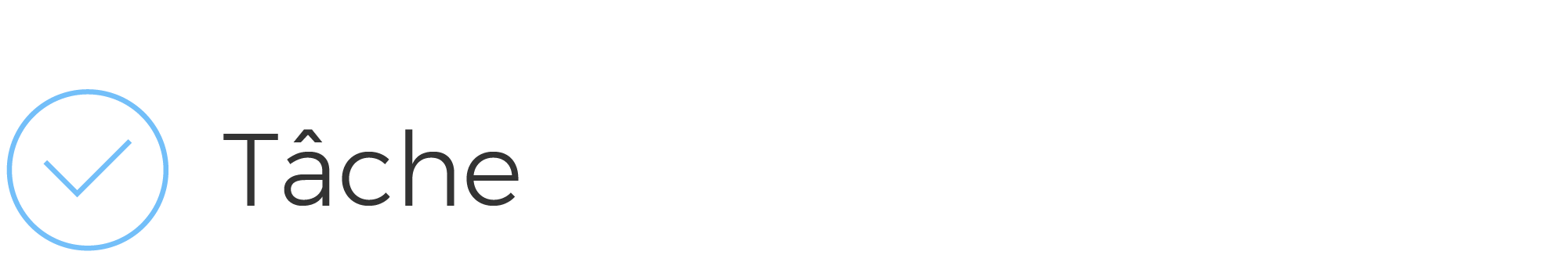
Un rôle est donc un ensemble de fichiers organisés dans une structure arborescente.

Mais 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.

**

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](https://yaml.org/) (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 :

1. Installer un serveur web **Apache** sur le premier serveur (**http1**).
2. Installer **PHP** également sur le premier serveur (**http1**).
3. Installer une base de données **MariaDB** sur le deuxième serveur (**bdd1**).
4. Télécharger les **fichiers sources de MediaWiki** et les mettre sur le **serveur web** (**http1**).
5. Configurer le **serveur web** (**http1**) pour pointer vers l’URL **http://http1/mediawiki**.
6. Finaliser l’installation de MediaWiki via **le script d’installation** sur **http1**.

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 :

1. Un rôle pour **installer** Apache : **apache**.
2. Un rôle pour **installer** MariaDB : **mariadb**.
3. Un rôle pour **configurer** Apache pour MediaWiki : **confapache**.
4. Un rôle pour **configurer** MariaDB pour MediaWiki : **confdb**.
5. 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.

$ mkdir roles

$ cd 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](https://galaxy.ansible.com/) 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**, quant à lui, 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

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

$ tree apache

Ce qui donne le résultat suivant :

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$

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 :

$ cd apache &&rm-r files meta templates vars defaults tests README.md

voici ce que donne la nouvelle arborescence :

roles/apache/

├── handlers

│ └── main.yml

└── tasks

├── main.yml

#### 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 :

$ mkdir -p mariadb/tasks/

$ touch mariadb/tasks/main.yml

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

Voici l’arborescence :

├── 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 :

$ mkdir mediawiki

Ce qui donne à ce stade :

../roles/

├── apache

│ ├── handlers

│ │ └── main.yml

│ ├── meta

│ │ └── main.yml

│ └── tasks

│ └── main.yml

├── 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

Ce qui donne :

└── mediawiki

├── common

│ └── 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 :

confdb

├── meta

│ └── main.yml

└── tasks

└── main.yml

#### 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

Ce qui donne l'arborescence suivante :

└── confapache

├── meta

│ └── main.yml

└── tasks

└── main.yml

### 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

Activez l'environnement virtuel :

$ source ansible2.7.10/bin/activate

Éditez le fichier **inventaire.ini**

$ vi inventaire.ini

[apache]

http1

[db]

bdd1

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 prédéfinis, 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 :

roles

├── apache

│ ├── handlers

│ │ └── main.yml

│ └── tasks

│ ├── main.yml

│ └── php7-install.yml

├── mariadb

│ └── tasks

│ └── main.yml

└── mediawiki

├── commun

│ └── defaults

│ └── main.yml

├── confapache

│ ├── meta

│ │ └── main.yml

│ └── tasks

│ └── 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 **9 fichiers YAML** pour les **5 rôles** dans ce chapitre :

1. Le rôle **apache**:
   * 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 ;
   * un fichier **tasks/php7-install.yml** contient les actions pour installer PHP ;
   * un fichier **handler/main.yml** contient les actions pour redémarrer le service Apache.
2. Le rôle **mariadb** :
   * un fichier **tasks/main.yml** contient les actions pour installer MariaDB.
3. Le rôle **commun** de MediaWiki :
   * un fichier **defaults/main.yml** contient les variables d'installation qui seront utilisées dans les rôles suivants.
4. 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.
5. 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, à l’instar du fichier **php7-install.yml** dédié à l’installation de PHP. L’installation de PHP aurait pu être intégrée au **main.yml**, mais pour une question de lisibilité et de modularité, elle a été faite 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 Centos, vous utiliserez Yum. Ça tombe bien, Yum a un module Ansible permettant d'automatiser l'installation de paquets.

**Yum** est le **gestionnaire de paquets** Centos ; vous allez pouvoir l'utiliser pour installer le paquet Apache qui s’appelle, sur Centos, ***httpd***.

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

$ ansible-doc yum

### 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**.

$ vi roles/apache/tasks/main.yml

Le fichier est commenté pour en comprendre le sens.

---

#1. Cette tâche permet d’installer Apache (httpd) à l’aide du module yum

- name: "apache installation"

yum:

name: "httpd"

state: "present"

#2. Cette tâche active le service Apache

- name: "apache service activation"

service:

name: "httpd"

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 php7 packages"

include: "php7-install.yml"

when: php\_install|default(False)|bool

Voyons en détail chacune des tâches décrites dans ce fichier :

1. La **première tâche,** "**apache installation**" va installer le serveur Apache avec le module “**yum**”. Le **name: "httpd"** indique le paquet concerné et le **state: "present"** spécifie qu’il faut l’installer.
2. La **deuxième tâche,** “**apache service activation**” va activer le service Apache avec le module “**service**”. Le **name: "httpd"** indique le service concerné, le **state: "started"** indique que le service sera démarré et le **enabled: yes** indique que le service sera activé.
3. La **troisième tâche,** “**install php7 packages**” inclut un fichier de configuration externe pour installer PHP. La tâche fait appel avec l’option “**include**” au fichier **php7-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.

Les filtres Ansible sont sous la forme [Jinja2](http://jinja.pocoo.org/docs/2.10/templates/#builtin-filters), 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.**

$ vi roles/apache/tasks/php7-install.yml

---

#1. Cette tâche installe le dépôt EPEL (Extra Packages for Enterprise Linux)

- name: "epel activation"

yum:

name: "epel-release"

state: present

#2. Cette tâche installe le dépôt REMI pour bénéficier du paquet PHP7

- name: "remi repo activation"

yum:

name: "https://rpms.remirepo.net/enterprise/remi-release-7.rpm"

state: present

#3. Cette tâche installe PHP7 et ses extensions

- name: "install php70 packages"

yum:

name: "php,php-mysql,php-xml,php-mbstring,php-mcrypt,php-gd,php-intl"

state: latest

enablerepo: "remi-php70"

changed\_when: yes

notify: [ "apache restart" ]

Voyons en détail chacune des tâches décrites dans ce fichier :

1. La **première tâche,** "**epel activation**", va activer le dépôt **epel** avec le module “**yum**”. Le **name: "epel-release"** indique le dépôt concerné et le **state: present** spécifie qu’il faut l’installer.   
   Sur **Centos**, tous les logiciels ne sont pas disponibles par défaut. C’est le cas pour **PHP7**; il faut donc ajouter le dépôt **EPEL** qui va contenir le paquet **PHP7**.
2. La **deuxième tâche,** “**remi repo activation**”, va installer le paquet remi-release-7.rpm avec le module “**yum**”. Le nom indique le service concerné ; le **state: present** indique que le paquet sera installé.  
   Sous **Centos**, les paquets logiciels sont sous la forme **RPM**. Pour installer **PHP7**, vous avez besoin du paquet **remi-release-7.rpm** qui est dans le dépôt **REMI** qui dépend du dépôt **EPEL**.
3. La **troisième tâche,** “**install php7 packages**”, installe les paquets php7 avec le module “**yum**”. 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 ;
   * **enablerepo: "remi-php70"** indique d’utiliser le dépôt **remi-php70** pour les installer ;
   * **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**.

---

- name: "apache restart"

service:

name: "httpd"

state: "restarted"

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

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

Toutes les informations relatives à l’utilisation du module service sont détaillées dans [la documentation Ansible](https://docs.ansible.com/ansible/latest/modules/service_module.html).

### Créez le fichier YAML pour le rôle mariadb

#### Écrivez le code pour installer MariaDB

$ vi roles/confdb/tasks/main.yml

---

# Installation des paquets mariadb serveur et son extension Python

- name: "mariadb-server installation"

yum:

name: "mariadb-server,MySQL-python"

state: "installed"

# Active le service MariaDB

- name: "start mariadb service"

service:

name: "mariadb"

state: "started"

enabled: yes

La première tâche "**mariadb-server installation**" va installer **mariadb** serveur et son extension **Python** avec le module “**yum**”. Le **name:  "mariadb-server,MySQL-python"** 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.**

$ viroles/mediawiki/commun/defaults/main.yml

---

# nom de la base de données

mediawiki\_db\_name: "mediawiki"

# nom de l’utilisateur de la base de données et son mot de passe

mediawiki\_db\_user: "mediawiki"

mediawiki\_db\_password: !vault |

$ANSIBLE\_VAULT;1.1;AES256 36663934633330396131376162363633666635376633313430616164303662633233303232353839 3932666230633632626434363734623537623463363631320a636662363032376563303163643366 37626237383534653035663335623233646364326365326565366361343835663030316533376664 6633376539663563310a386439653931623333316532326262343562326638313266343236346536

3032

# nom et mot de passe de l’administrateur Mediawiki

mediawiki\_admin\_user: "admin"

mediawiki\_admin\_password: !vault |

$ANSIBLE\_VAULT;1.1;AES256 36663934633330396131376162363633666635376633313430616164303662633233303232353839 3932666230633632626434363734623537623463363631320a636662363032376563303163643366 37626237383534653035663335623233646364326365326565366361343835663030316533376664 6633376539663563310a386439653931623333316532326262343562326638313266343236346536

3032

# nom du Mediawiki et son titre

mediawiki\_name: "mediawiki"

mediawiki\_title: "ELS"

# l’emplacement du répertoire d'installation de Mediawiki

mediawiki\_directory: "/var/www/html/{{mediawiki\_name}}"

# répertoire de maintenance de Mediawiki

mediawiki\_maintenance\_directory: "{{mediawiki\_directory}}/maintenance"

# Definie 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/1.31/mediawiki-1.31.1.tar.gz"

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 : **Ansible2019**, 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 **Ansible2019** et confirmez-la.

Voici la chaîne générée par Ansible :

mediawiki\_db\_password: !vault |

$ANSIBLE\_VAULT;1.1;AES256 36663934633330396131376162363633666635376633313430616164303662633233303232353839 3932666230633632626434363734623537623463363631320a636662363032376563303163643366 37626237383534653035663335623233646364326365326565366361343835663030316533376664 6633376539663563310a386439653931623333316532326262343562326638313266343236346536

3032

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** :

$ vi roles/mediawiki/confdb/meta/main.yml

La déclaration se fait de la façon suivante :

dependencies:

- role: "mediawiki/commun"

#### Écrivez le code pour configurer MariaDB pour MediaWiki

La configuration va se faire dans le fichier suivant :

$ vi roles/mediawiki/confdb/tasks/main.yml

---

#1. Installation de la base de donnée Mediawiki

- name: "mediawiki database"

mysql\_db:

name: "{{mediawiki\_db\_name}}"

state: present

#2. Création d’un accès utilisateur et attribution des privilèges sur la base Mediawiki

- name: "mediawiki user+privileges"

mysql\_user:

name: "{{mediawiki\_db\_user}}"

password: "{{mediawiki\_db\_password}}"

priv: "{{mediawiki\_db\_name}}.\*:ALL"

host: "{{item}}"

state: present

with\_items: "{{groups.apache}}"

Voyons ensemble le détail de ces tâches :

1. 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.
2. 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.

Vous trouverez la liste des variables magiques sur la page [Special Variables de la doc d’Ansible](https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html).

### 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** :

$ vi roles/mediawiki/confapache/meta/main.yml

La déclaration se fait de la façon suivante :

dependencies:

- role: "mediawiki/commun"

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

$ viroles/mediawiki/confapache/tasks/main.yml

---

#1. Création du repertoire pour l’installation des fichiers Mediawiki

- name: "mediawiki directory"

file:

path: "{{mediawiki\_directory}}"

owner: "apache"

group: "apache"

state: directory

#2. Décompresse le fichier source archive Mediawiki et le formate sans extension

- name: "uncompress mediawiki archive"

unarchive:

src: "{{mediawiki\_archive\_url}}"

dest: "{{mediawiki\_directory}}"

owner: "apache"

group: "apache"

remote\_src: yes

# supprime mediawiki-1.xx.x/ du chemin

extra\_opts: --transform=s/mediawiki-[0-9\.]\*\///

#3. Exécute la tâche avec l'utilisateur apache, se place dans le répertoire de maintenance et exécute la commande de configuration si le fichier localsetting.php n’existe pas

- name: "mediawiki configuration"

become: yes

become\_user: "apache"

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}}

{{mediawiki\_title}} {{mediawiki\_admin\_user}}

run\_once: yes

delegate\_to: "{{item}}"

with\_items: "{{groups.apache}}"

#4. Exécute la tâche avec l'utilisateur apache, se place dans le répertoire de maintenance et exécute la commande de mise à jour de la baseune seule fois

- name: "mediawiki db update"

become: yes

become\_user: "apache"

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"

Voyons le détail de ces tâches :

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: "apache"** indique le propriétaire du répertoire ;
* **group: "apache"** 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: "apache"** indique le propriétaire des fichiers ;
* **group: "apache"** 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**: "apache" 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’inventair,e 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 **SElinux** et le **firewalling** ne sont pas activés sur les nodes Centos. 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** **SElinux :** setenforce 0  ;
* pour désactiver **définitivement SElinux** : éditez le fichier  /etc/selinux/config  puis ajoutez-y la ligne SELINUX=permissive ;
* pour désactiver **temporairement** le **firewalling** : systemctl stop firewalld.service  ;
* pour désactiver **définitivement** le **firewalling** : systemctl disable firewalld.service .

#### 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 :

192.168.122.11 http1

192.168.122.12 bdd1

Ainsi, vous pouvez pinger **http1** et **bdd1** depuis les nodes. Ce qui évite d’utiliser directement les adresses IP.

#### Créez les playbooks pour exécuter les rôles



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 :

1. Un pour **installer Apache**.
2. Un pour **installer MariaDB**.
3. 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 :

$ vi install-apache.yml

Le contenu du fichier ressemble à celui-ci :

---

- name: "Installation apache"

hosts: http1

roles:

- role: "apache"

php\_install: yes

#### 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](https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html).

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: http1  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

SUDO password:

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 commandeansible utilisée précédemment.

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

PLAY [Installation apache] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [http1]

TASK [apache : apache installation] \*\*\*\*\*\*\*\*\*

changed: [http1]

TASK [apache : apache service activation] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [http1]

TASK [apache : epel activation] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [http1]

TASK [apache : remi repo activation] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [http1]

TASK [apache : install php70 packages] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [http1]

RUNNING HANDLER [apache : apache restart] \*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [http1]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

http1 : ok=7 changed=6 unreachable=0 failed=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 connaitre la **version de PHP** :

$ ssh user-ansible@http1

Last login: Mon May 13 21:20:26 2019 from 192.168.122.10

[user-ansible@http1 ~]$ php --version

PHP 7.0.33 (cli) (built: Apr 30 2019 13:10:17) ( NTS )

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 en version **7**.

### 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 :

---

- name: "Installation MariaDB"

hosts: bdd1

gather\_facts: no

roles:

- role: 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 connection suivantes :

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

SUDO password:

Ce qui donne en retour :

PLAY [Install MariaDB] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [mariadb : mariadb-server installation] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [bdd1]

TASK [mariadb : start mariadb service] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [bdd1]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bdd1 : ok=2 changed=2 unreachable=0 failed=0

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

- 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"

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-playbook -i inventaire.ini --user user-ansible --become --ask-become-pass --ask-vault-pass install-mediawiki.yml

SUDO password:

Vault password:

Rien de nouveau ici par rapport aux commandes précédentes, seul le nom du playbook change.

Ce qui donne le résultat suivant :

PLAY [MediaWiki db configuration] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [mediawiki/confdb : mediawiki database] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [bdd1]

TASK [mediawiki/confdb : mediawiki user+privileges] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*va

changed: [bdd1] => (item=http1)

PLAY [MediaWiki apache configuration] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [mediawiki/confapache : mediawiki directory] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [http1]

TASK [mediawiki/confapache : uncompress mediawiki archive] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [http1]

TASK [mediawiki/confapache : mediawiki configuration] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [http1 -> http1] => (item=http1)

TASK [mediawiki/confapache : mediawiki db update] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [http1]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bdd1 : ok=2 changed=2 unreachable=0 failed=0

http1 : ok=4 changed=4 unreachable=0 failed=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 !

Voila, 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.