CRÉATION DE NOTRE PLAYBOOK ANSIBLE (STACK LAMP)

Introduction

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

Pourquoi les Playbooks

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

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

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

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

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

Récupérerez d'abord le projet complet ici et ensuite sans plus attendre, commençons par les explications !

Structure du projet

Quels sont les objectifs de notre Playbook ?

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

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

Arborescence du projet

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

```
|__ files
|
|___ app
|       |___ index.php
|
|___ validation.php
|
|__ templates
|   |___ db-config.php.j2
|
|___ table.sql.j2
|
|__ vars
|
|___ main.yml
|

|___ ansible.cfg

|___ hosts

|___ Playbook.yml
```

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

- **templates/** : contient des modèles de configurations dynamiques au format jinja qui sont destinés à être traités par le module template.

Le fichier inventaire

Pour ce projet, j'ai décidé de me séparer du fichier inventaire situé par défaut dans **/etc/ansible/hosts** et de **créer mon propre fichier hosts** à la racine du projet, où je me suis permis de séparer le serveur de base de données par rapport au serveur web, voici donc à quoi ressemble notre nouveau fichier inventaire :

```
[web]
slave-1 ansible_user=vagrant
[db]
slave-2 ansible_user=vagrant
```

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

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

```
[defaults]
inventory = ./hosts
```

Explication du Playbook

Pour commencer, voici déjà le contenu de notre Playbook :

```
---
# WEB SERVER

- hosts: web
  become: true
  vars_files: vars/main.yml

  tasks:
  - name: install apache and php last version
    apt:
      name:
        - apache2
        - php
        - php-mysql
      state: present
      update_cache: yes

  - name: Give writable mode to http folder
    file:
      path: /var/www/html
      state: directory
      mode: '0755'

  - name: remove default index.html
    file:
      path: /var/www/html/index.html
      state: absent

  - name: upload web app source
    copy:
      src: app/
      dest: /var/www/html/

  - name: deploy php database config
    template:
      src: "db-config.php.j2"
      dest: "/var/www/html/db-config.php"

  - name: ensure apache service is start
    service:
      name: apache2
      state: started
      enabled: yes


# DATABASE SERVER

- hosts: db
  become: true
  vars_files: vars/main.yml
  vars:
    root_password: "my_secret_password"
```

```
tasks:
- name: install mysql
  apt:
    name:
      - mysql-server
      - python-mysqldb # for mysql_db and mysql_user modules
    state: present
    update_cache: yes

- name: Create MySQL client config
  copy:
    dest: "/root/.my.cnf"
    content: |
      [client]
      user=root
      password={{ root_password }}
    mode: 0400

- name: Allow external MySQL connexions (1/2)
  lineinfile:
    path: /etc/mysql/mysql.conf.d/mysqld.cnf
    regexp: '^skip-external-locking'
    line: "# skip-external-locking"
  notify: Restart mysql

- name: Allow external MySQL connexions (2/2)
  lineinfile:
    path: /etc/mysql/mysql.conf.d/mysqld.cnf
    regexp: '^bind-address'
    line: "# bind-address"
  notify: Restart mysql

- name: upload sql table config
  template:
    src: "table.sql.j2"
    dest: "/tmp/table.sql"

- name: add sql table to database
  mysql_db:
    name: "{{ mysql_dbname }}"
    state: present
    login_user: root
    login_password: '{{ root_password }}'
    state: import
    target: /tmp/table.sql

- name: "Create {{ mysql_user }} with all {{ mysql_dbname }} privileges"
  mysql_user:
    name: "{{ mysql_user }}"
    password: "{{ mysql_password }}"
    priv: "{{ mysql_dbname }}.*:ALL"
    host: "{{ webserver_host }}"
    state: present
    login_user: root
```

```
      login_password: '{{ root_password }}'
      login_unix_socket: /var/run/mysqld/mysqld.sock

  handlers:
    - name: Restart mysql
      service:
        name: mysql
        state: restarted
```

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

Serveur web

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

Partie hosts

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

```
---

- hosts: web
```

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

élévation de privilèges

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

```
become: true
```

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

```
tasks:
- service:
    name: nginx
    state: started
  become: yes
```

Variables

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

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

Voici le contenu du fichier de variables :

```
---
mysql_user: "admin"
mysql_password: "secret"
mysql_dbname: "blog"
db_host: "192.168.0.22"
webserver_host: "192.168.0.21"
```

- `mysql_user` : l'utilisateur de notre base de données mysql qui exécutera nos requêtes SQL depuis notre application web.

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

Les tâches

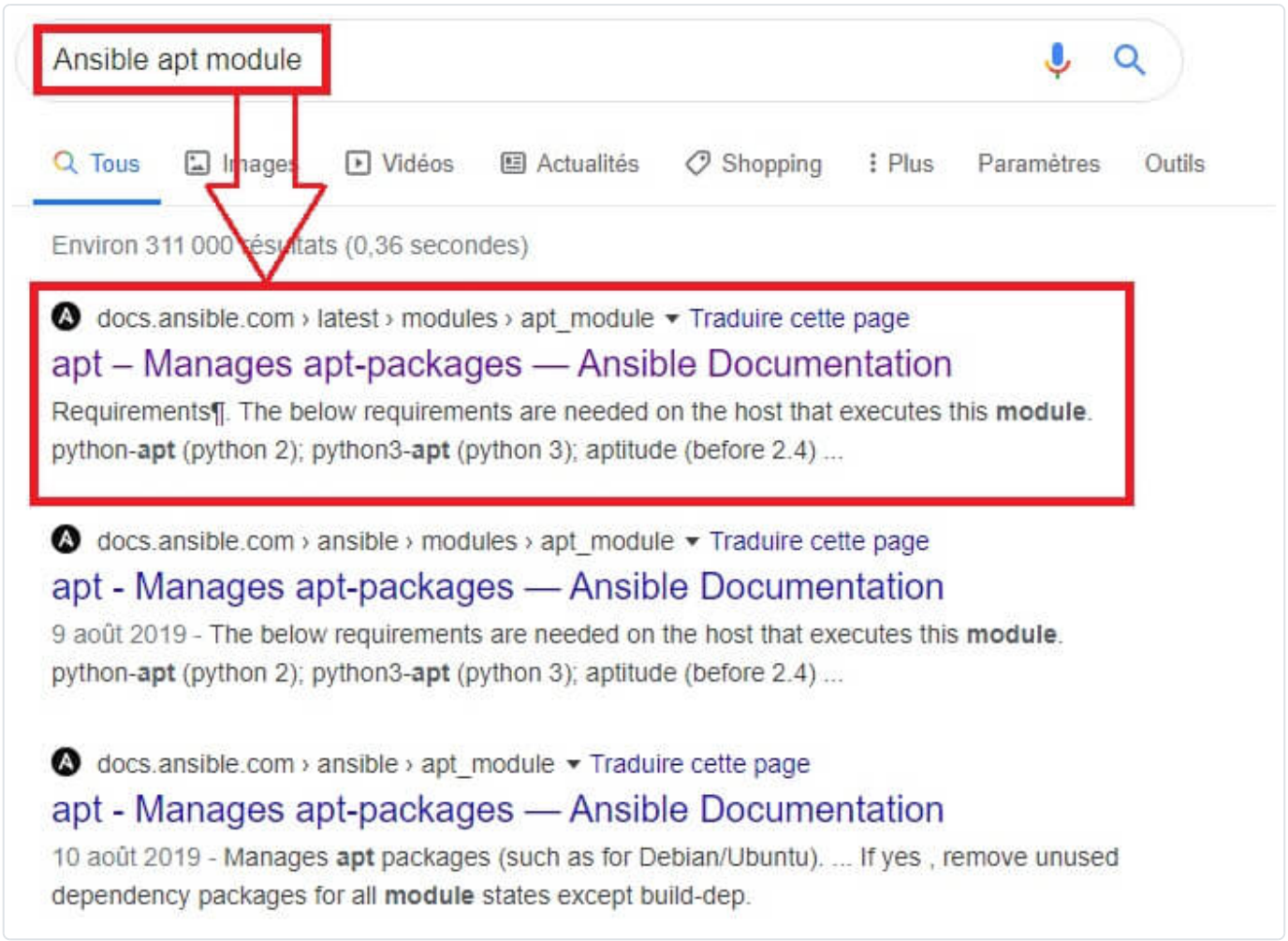
Chaque hôte contient une liste de tâches au-dessous du mot-clé `tasks`. Les tâches sont exécutées dans l'ordre, une à la fois, sur toutes les machines correspondant au modèle d'hôte avant de passer à la tâche suivante.

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

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

Dans notre projet, notre premier besoin consiste à installer les packages apache2, php et php-mysql avec le gestionnaire de paquêts apt. Et peut-être que vous vous

demandez comment j'ai faits pour trouver le module adéquat ? La réponse est "Google !", en effet Google est votre meilleur ami (ou Bing, Ecosia, Qwant, DuckDuckGo, etc ...) ! J'ai tapé sur le moteur de recherche les mots-clés suivants "Ansible apt module" et j'ai cliqué sur le premier lien fourni par Google (celui-ci).



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

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

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

- `name` (type: liste) : liste de noms de packages (on peut aussi spécifier la version du package ex curl=1.6 ou curl=1.0*).
- `state` (type: string) : indique l'état du package, voici un exemple des valeurs possibles :
  - `latests` : assure que c'est toujours la dernière version qui est installée.
  - `present` : vérifie si le package est déjà installé, si c'est le cas il ne fait rien, sinon il l'installe.
  - `absent` : supprime le package s'il est déjà installé.
- `update_cache` (type: booléen) : exécute l'équivalent de la commande `apt-get update` avant l'installation des paquets.

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

```
- name: install apache and php last version
  apt:
    name:
      - apache2
      - php
      - php-mysql
    state: present
 update_cache: yes
```

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

Les types en Yaml :

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

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

```
- name: install apache and php last version
  apt:
 name: ['apache2', 'php', 'php-mysql']
 state: present
 update_cache: yes
```

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

```
update_cache: yes
update_cache: no
update_cache: True
update_cache: TRUE
update_cache: false
```

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

```
tasks:
  - name: deploy test.cfg file
    copy:
      src: /tmp/test.cfg
      dest: /tmp/test.cfg
      owner: root
      group: root
      mode: 0644
```

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

```
tasks:
  - name: deploy test.cfg file
    copy: src=/tmp/test.cfg dest=/tmp/test.cfg
    owner=root group=root mode=0644
```

Idempotence

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

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

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

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

Suite des tâches

- ~~Installer les packages apache2, php et php-mysql~~
- Déployer les sources de notre application dans notre serveur web distant

- S'assurer que le service apache est bien démarré

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

```
- name: Give writable mode to http folder
  file:
    path: /var/www/html
    state: directory
 mode: '0755'
```

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

```
- name: remove default index.html
  file:
    path: /var/www/html/index.html
    state: absent
```

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

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

```
- name: upload web app source
  copy:
    src: app/
 dest: /var/www/html/
```

Peut-être que vous l'avez remarqué, mais je n'ai pas eu besoin de fournir le dossier `files` dans le chemin du paramètre `src`, car ce dossier est spécialement conçu

pour que le module copy recherche dedans automatiquement nos différents fichiers ou dossiers à envoyer (si vous déposez vos fichiers dans un autre emplacement, il faut dans ce cas que vous insériez le chemin relatif ou absolu complet)

Fichier de configuration dynamique (Jinja2)

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

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

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

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

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

```
<?php
const DB_DSN = 'mysql:host={{ db_host }};dbname={{ mysql_dbname }}';
const DB_USER = "{{ mysql_user }}";
const DB_PASS = "{{ mysql_password }}";



$options = array(
    PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8", // encodage utf-8
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION, // gérer les erreurs en tant qu'excep
    PDO::ATTR_EMULATE_PREPARES => false // faire des vrais requêtes préparées et non u
);
```

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

Ce qui nous donne la tâche suivante :

```
- name: deploy php database config
  template:
    src: "db-config.php.j2"
    dest: "/var/www/html/db-config.php"
```

Information

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

Le module service

- ~~Installer les packages apache2, php et php-mysql~~

- ~~Déployer les sources de notre application dans notre serveur web distant~~
- S'assurer que le service apache est bien démarré

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

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

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

```
- name: ensure apache service is start
  service:
    name: apache2
    state: started
 enabled: yes
```

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

Serveur de base de données

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

Préparation des tâches

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

```
- hosts: db
  become: true
  vars_files: vars/main.yml
  vars:
    root_password: "my_secret_password"
```

Installation des paquets

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

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

```
- name: install mysql
  apt:
    name:
      - mysql-server
      - python-mysqldb # for mysql_db and mysql_user modules
    state: present
    update_cache: yes
```

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

Modification du mot de passe root

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

```
- name: Create MySQL client config
  copy:
    dest: "/root/.my.cnf"
    content: |
      [client]
      user=root
   password={{ root_password }}
```

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

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

Autorisation des connexions externes

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

mysql distant.

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

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

```
- name: Allow external MySQL connexions (1/2)
  lineinfile:
    path: /etc/mysql/mysql.conf.d/mysqld.cnf
    regexp: '^skip-external-locking'
    line: "# skip-external-locking"
  notify: Restart mysql

- name: Allow external MySQL connexions (2/2)
  lineinfile:
    path: /etc/mysql/mysql.conf.d/mysqld.cnf
    regexp: '^bind-address'
    line: "# bind-address"
  notify: Restart mysql
```

notify et handlers

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

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

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

```
handlers:
  - name: Restart mysql
    service:
      name: mysql
      state: restarted
```

Création et configuration de notre base de données

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

```
- name: upload sql table config
  template:
    src: "table.sql.j2"
    dest: "/tmp/table.sql"

- name: add sql table to database
  mysql_db:
    name: "{{ mysql_dbname }}"
    state: present
    login_user: root
    login_password: '{{ root_password }}'
    state: import
 target: /tmp/table.sql
```

Information

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

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

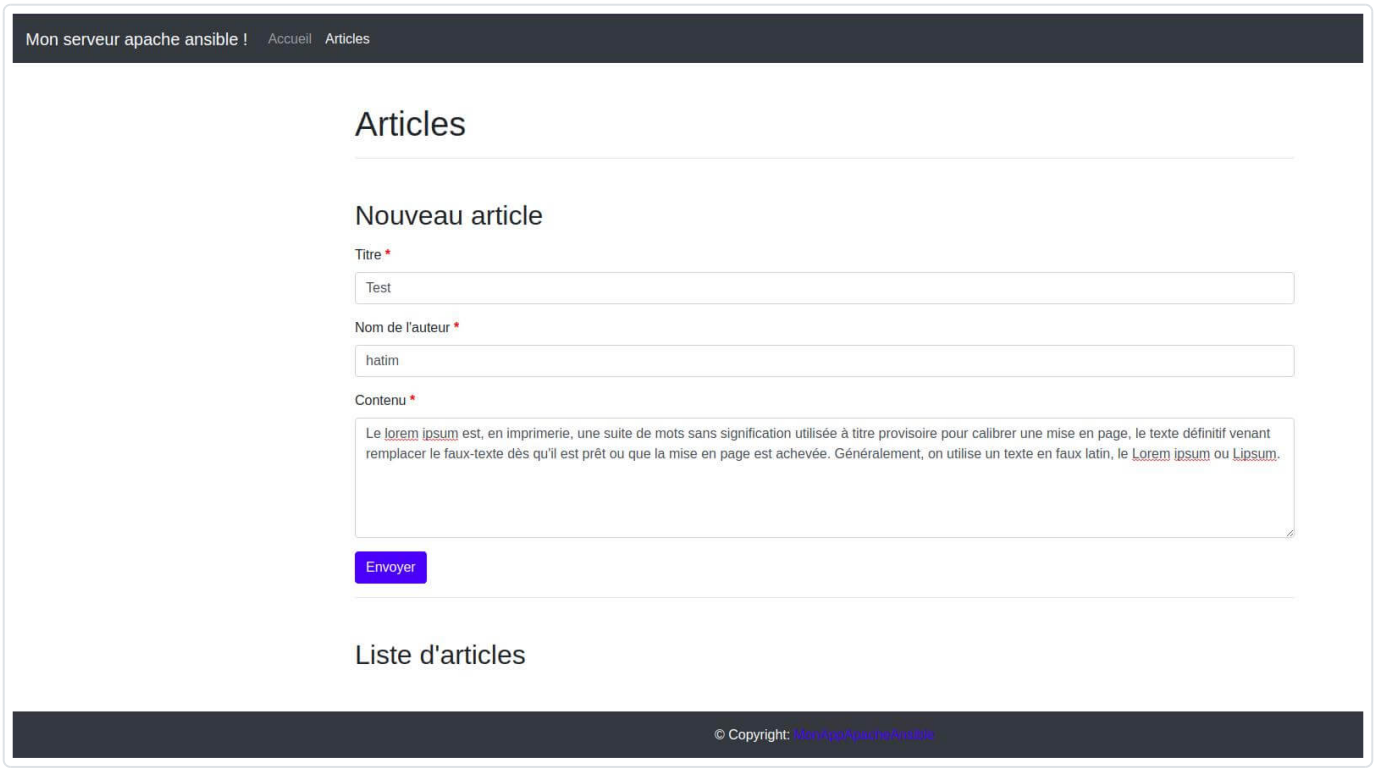
```
- name: "Create {{ mysql_user }} with all {{ mysql_dbname }} privileges"
  mysql_user:
    name: "{{ mysql_user }}"
    password: "{{ mysql_password }}"
    priv: "{{ mysql_dbname }}.*:ALL"
    host: "{{ webserver_host }}"
    state: present
    login_user: root
    login_password: '{{ root_password }}'
 login_unix_socket: /var/run/mysqld/mysqld.sock
```

Test

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

```
ansible-playbook playbook.yml
```

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



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

Conclusion

Je pense que vous l'aurez compris, le Playbook est un fichier permettant de faciliter la gestion de nos modules Ansible. Nous verrons dans le prochain chapitre

comment améliorer notre playbook avec les conditions et nous aborderons également les boucles dans les playbooks.