CRÉATION D'UN PLAYBOOK MULTI DISTRIBUTIONS

Introduction

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

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

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

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

Préparation des nouvelles machines

Jusqu'ici, nous avions travaillé avec des machines Ubuntu et nous les avions créées avec l'outil vagrant. Si vous le souhaitez, vous pouvez les supprimer pour en créer des nouvelles avec l'image Fedora dans le but de tester notre nouveau Playbook.

Pour ce faire, il suffit d'utiliser une nouvelle image dans notre fichier Vagrantfile, voici à quoi ressemblera ce nouveau fichier :

```
# ######################################################################
# #################### CONFIGURATION VARIABLES ########################
# ######################################################################
IMAGE_NAME = "fedora/30-cloud-base"   # Image to use
MEM = 2048                            # Amount of RAM
CPU = 1                               # Number of processors
SLAVE_NBR = 2                         # Number of slaves node
NETWORK_ADAPTER="wlp1s0"              # Bridge network adapter


Vagrant.configure("2") do |config|
    config.ssh.insert_key = false

    # RAM and CPU config
    config.vm.provider "virtualbox" do |v|
        v.memory = MEM
        v.cpus = CPU
    end

    # Slave node config
    (1..SLAVE_NBR).each do |i|
        config.vm.define "slave-#{i}" do |slave|
            # OS and Hostname
            slave.vm.box = IMAGE_NAME
            slave.vm.hostname = "slave-#{i}"
            slave.vm.network "public_network", bridge: NETWORK_ADAPTER, ip: "192.168
        end
    end
end
```

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

```
vagrant up
```

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

```
ssh-copy-id vagrant@[IP_MACHINE_1]
ssh-copy-id vagrant@[IP_MACHINE_2]
```

Rappel

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

Amélioration du Playbook

Les Facts

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

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

```
ansible slave-1 -m setup
```

Résultat :

```
slave-1 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.0.2.15",
            "192.168.0.21"
  ]
...
```

```
...
  "ansible_virtualization_role": "guest",
        "ansible_virtualization_type": "virtualbox",
        "discovered_interpreter_python": "/usr/bin/python3",
        "gather_subset": [
            "all"
        ],
        "module_setup": true
    },
    "changed": false
}
```

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

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

Résultat :

```
slave-1 | SUCCESS => {
    "ansible_facts": {
        "ansible_bios_date": "12/01/2006",
        "ansible_bios_version": "VirtualBox",
        "ansible_hostname": "slave-1",
        "ansible_hostnqn": "",
        "ansible_os_family": "RedHat",
        "ansible_ssh_host_key_ecdsa_public": "AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbm
        "ansible_ssh_host_key_ed25519_public": "AAAAC3NzaC1lZDI1NTE5AAAAIIs2pEEEGym6
        "ansible_ssh_host_key_rsa_public": "AAAAB3NzaC1yc2EAAAADAQABAAABAQCrtN15m8q3
        "ansible_user_gecos": "",
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false
}
```

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

Les conditions

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

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

```
---

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

  tasks:

  - name: "Juste pour le debug"
    debug: var=ansible_os_family

  - fail:
 when: 1 == 1
...
...
```

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

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

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

Exécutons notre playbook :

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

Résultat :

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

TASK [Gathering Facts] ***
ok: [slave-1]

TASK [Juste pour le debug] ************
ok: [slave-1] => {
    "ansible_os_family": "RedHat"
}

TASK [fail] **************
fatal: [slave-1]: FAILED! => {"changed": false, "msg": "Failed as requested from tas
```

```
PLAY RECAP ***************
slave-1 : ok=2    changed=0    unreachable=0    failed=1
```

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

```
- name: "Juste pour le debug"
  debug: var=ansible_facts['os_family']
```

Utilisation des conditions dans la partie Web

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

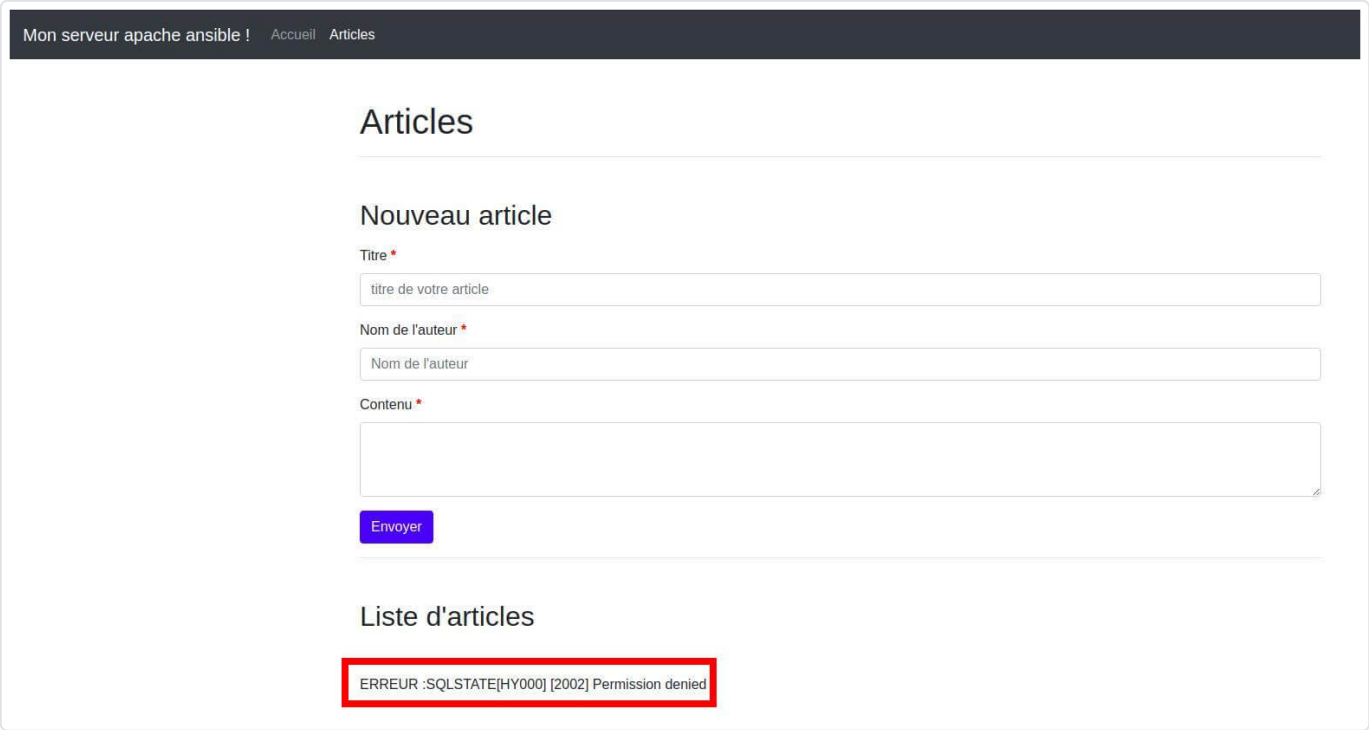
```
---

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

  tasks:
  - name: install apache and php last version for Debian os family
    apt:
      name: ['apache2', 'php', 'php-mysql']
      state: present
      update_cache: yes
    when: ansible_facts['os_family'] == "Debian"

  - name: install apache and php last version for RedHat os family
    yum:
      name: ['httpd', 'php', 'php-mysqlnd']
      state: present
      update_cache: yes
    when: ansible_facts['os_family'] == "RedHat"

  - fail:
 when: 1 == 1
```

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

suivantes :

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

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

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

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

- name: ensure apache service is start (Debian os family)
  service:
    name: apache2
    state: started
    enabled: yes
  when: ansible_facts['os_family'] == "Debian"

- name: ensure apache service is start (RedHat os family)
  service:
    name: httpd
    state: started
    enabled: yes
  when: ansible_facts['os_family'] == "RedHat"

- name: enable connection with remote database (RedHat os family)
  shell: setsebool -P httpd_can_network_connect_db 1
  when: ansible_facts['os_family'] == "RedHat"
```

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



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

Utilisation des conditions dans la partie base de données

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

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

```
- name: install mysql repo (Fedora)
  yum:
    name: "http://repo.mysql.com/mysql80-community-release-fc{{ ansible_facts['distr
    state: present
```

```
    update_cache: yes
  when: ansible_facts['distribution'] == "Fedora"

- name: install mysql repo (CENTOS or RedHat)
  yum:
    name: "http://repo.mysql.com/mysql80-community-release-el{{ ansible_facts['distri
    state: present
    update_cache: yes
  when: ansible_facts['os_family'] == "RedHat" and ansible_facts['distribution'] != "

- name: install mysql package (RedHat os family)
  yum:
    name: mysql-community-server
    state: present
    disablerepo: mysql80-community
    enablerepo: mysql57-community
  when: ansible_facts['os_family'] == "RedHat"

- name: install PyMySQL from pip (RedHat os family)
  pip:
    name: PyMySQL # for mysql_db and mysql_user modules
  when: ansible_facts['os_family'] == "RedHat"
```

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

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

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

Opérateur	Signification	Description
`or`	OU	Vrai si au moins une des comparaisons est vraie
`and`	ET	Vrai si toutes les comparaisons sont vraies
`not`	NON	Retourne faux si la comparaison est vraie et vraie si la comparaison est fausse

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

```
- name: ensure mysql service is start (Debian os family)
  service:
    name: mysql
    state: started
    enabled: yes
  when: ansible_facts['os_family'] == "Debian"

- name: ensure mysqld service is start (RedHat os family)
  service:
    name: mysqld
    state: started
    enabled: yes
  when: ansible_facts['os_family'] == "RedHat"
```

Enregistrer les sorties dans une variable

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

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

```
- name: Register temporary password (RedHat os family)
  shell: "grep 'temporary password' /var/log/mysqld.log  | awk '{print $(NF)}'"
  register: password_tmp
  when: ansible_facts['os_family'] == "RedHat"
```

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

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

Voici le résultat lors du debug de cette variable enregistrée :

```
- name: Register temporary password (RedHat os family)
  shell: "grep 'temporary password' /var/log/mysqld.log  | awk '{print $(NF)}'"
  register: password_tmp
  when: ansible_facts['os_family'] == "RedHat"

- debug: var=password_tmp

- fail:
  when: 1 == 1
```

Résultat :

```
TASK [debug] ********
ok: [slave-2] => {
    "password_tmp": {
        "changed": true,
        "cmd": "grep 'temporary password' /var/log/mysqld.log  | awk '{print $(NF)}'"
        "delta": "0:00:00.007624",
        "end": "2020-02-05 18:35:21.521570",
        "failed": false,
        "rc": 0,
        "start": "2020-02-05 18:35:21.513946",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "d.to29&Rque&",
        "stdout_lines": [
            "d.to29&Rque&"
        ]
    }
}
```

Surcharger la valeur d'une variable

Lors de mes tests, j'ai remarqué que le mot de passe du compte root mysql par défaut pouvait être utilisé sur des machines de la famille Debian, ce qui n'est pas le cas des machines de type RedHat où il faut utiliser le mot de passe temporaire

récupérée précédemment. Le mot de passe peut donc varier d'une famille de distribution à l'autre.

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

```
- name: Set default root user password (RedHat os family)
  set_fact:
    default_root_password: '{{ password_tmp.stdout }}'
  when: ansible_facts['os_family'] == "RedHat"
```

Forcer l'ignorance d'une erreur

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

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

```
- name: check if mysql config is correct (RedHat os only)
  shell: 'grep "^bind-address" /etc/my.cnf'
  register: test_grep
  when: ansible_facts['os_family'] == "RedHat"
  ignore_errors: yes # dont exit if it doesn't found something
```

```
- name: change mysql config (RedHat os only)
  blockinfile:
    path: /etc/my.cnf
    insertafter: EOF
    block: |
      default_authentication_plugin=mysql_native_password
      bind-address=0.0.0.0
      default_password_lifetime=0
      validate_password_policy=LOW
      validate_password_length=6
      validate_password_number_count=0
  when: ansible_facts['os_family'] == "RedHat" and test_grep.rc != 0
  notify: Restart mysqld
```

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

```
- name: Change root SQL password and GRANT root privileges (RedHat os family)
  command: "mysql --user=root --password={{ default_root_password }} --connect-expire
  ignore_errors: yes # ignore errors because we only change mysql root password once
  when: ansible_facts['os_family'] == "RedHat"

- name: Create MySQL client config (Debian os family)
  copy:
    dest: "/root/.my.cnf"
    content: |
      [client]
      user=root
      password={{ root_password }}
    mode: 0400
  when: ansible_facts['os_family'] == "Debian"


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

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


- name: "Create {{ mysql_user }} with all {{ mysql_dbname }} privileges (Debian os fa
  mysql_user:
```

```
    name: "{{ mysql_user }}"
    password: "{{ mysql_password }}"
    priv: "{{ mysql_dbname }}.*:ALL"
    host: "{{ webserver_host }}"
    state: present
    login_user: root
    login_password: '{{ root_password }}'
    login_unix_socket: /var/run/mysqld/mysqld.sock
  when: ansible_facts['os_family'] == "Debian"
  notify: Restart mysql


- name: "Create {{ mysql_user }} with all {{ mysql_dbname }} privileges (RedHat os fa
  mysql_user:
    name: "{{ mysql_user }}"
    password: "{{ mysql_password }}"
    priv: "{{ mysql_dbname }}.*:ALL"
    host: "{{ webserver_host }}"
    state: present
    login_user: root
    login_password: '{{ root_password }}'
    login_unix_socket: /var/lib/mysql/mysql.sock
  when: ansible_facts['os_family'] == "RedHat"
  notify: Restart mysqld
```

Les Boucles

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

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

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

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

```
- name: install extra packages (Debian os family)
  apt:
    name: "{{ extra_packages_debian }}"
    state: present
  when: ansible_facts['os_family'] == "Debian"

- name: install extra packages (RedHat os family)
  yum:
    name: "{{ extra_packages_redhat }}"
    state: present
  when: ansible_facts['os_family'] == "RedHat"
```

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

```
- name: add several users
  user:
    name: "{{ item }}"
    state: present
    groups: "dev"
    with_items:
     - user1
     - user2
```

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

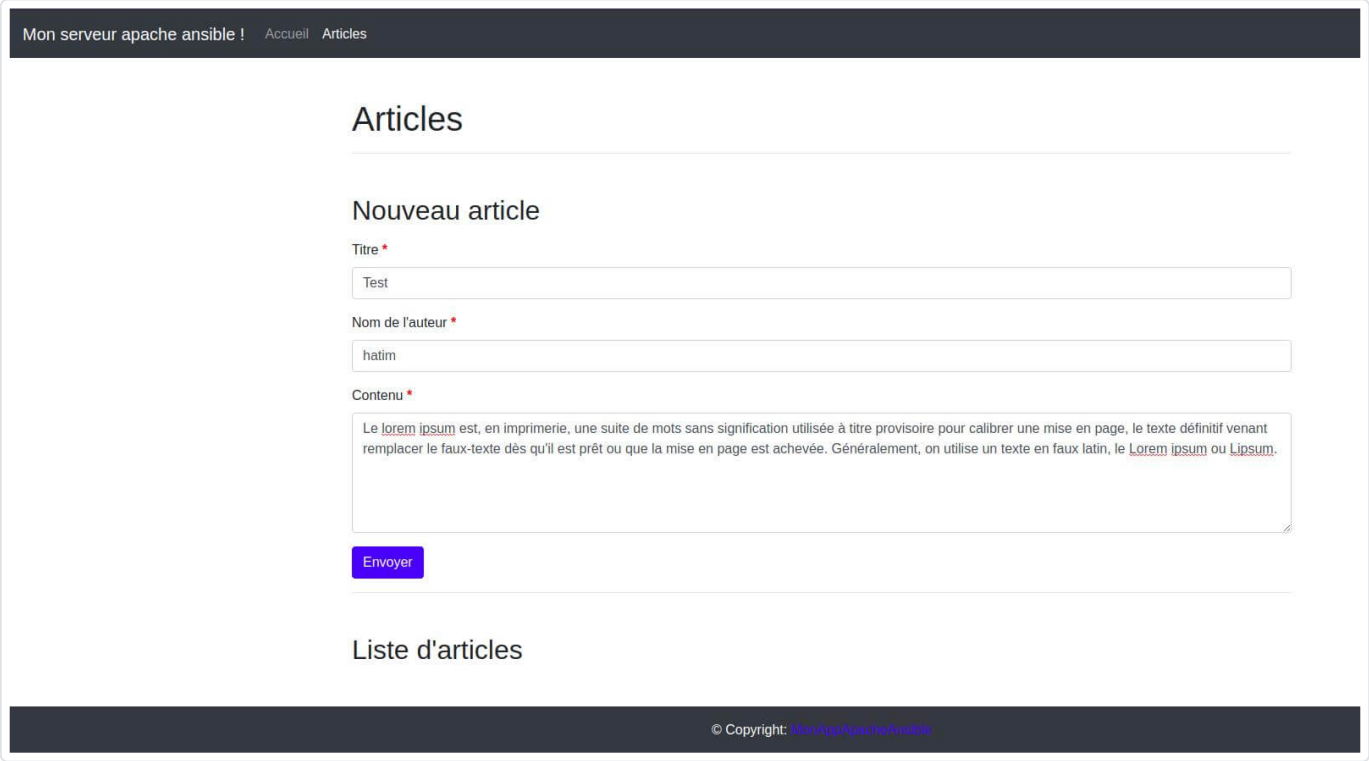
```
- name: add several users
  user:
    name: "{{ item.name }}"
    state: present
    groups: "{{ item.groups }}"
    with_items:
```

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

Conclusion et Test

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

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



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

Liste d'articles

Test

24/01/20 19:26

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

— hatim

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