Ansible

L'un des plus grands outils d'automatisation et de déploiement sur Linux via SSH.

- [Ansible] Installation
- [Ansible] Cheat-sheet

[Ansible] Installation

Introduction

Ansible est une plateforme logicielle libre pour la configuration et la gestion des ordinateurs via le protocole SSH.

Elle combine le déploiement de logiciell'exécution des tâches et la gestion de configuration.

Installation

Debian / Ubuntu

```
sudo apt update && sudo apt install -y ansible
```

RHEL / Fedora / CentOS / Alma Linux / Rocky Linux

```
dnf update && dnf install ansible
```

Arch Linux / Manjaro

```
sudo pacman -Sy && sudo pacman -S ansible
```

[Ansible] Cheat-sheet

Introduction

Cette fiche va décrire plusieurs commandes et procédures pour manipuler Ansible.

Cet outil permet d'effectuer des installations ou des opérations d'administration sur tout un parc ou sur un groupe de machines via le protocole SSH.

Inventaire

Tout d'abord il faut établir un inventaire dans le fichier **inventory.ini** :

```
[servers]
192.168.2.3
192.168.2.4
```

> Une paire de clé SSH doit être configuré entre le serveur de contrôle et le serveur cible pour qu'Ansible puisse fonctionner.

Pour vérifier la liaison entre les machines de votre inventaire et ansible, vous pouvez effectuer un ping :

```
ansible all -i inventory.ini -m ping
```

> Vous devriez avoir une réponse en json avec le mot-clé **SUCCESS** .

Playbooks

Le playbook est une liste de tâches pour effectuer une opération sur les serveurs cibles (ex: MAJ, installation logiciel etc).

Créez un fichier pour votre playbook comme **my_playbook.yml** :

```
- hosts: servers
  become: yes
  tasks:
    - name: Ensure Apache is installed
      apt:
        name: apache2
        state: present
```

> L'instruction **become** vous permet d'exécuter les tâches en tant que root lorsqu'elle est définie sur **true**.

Exécution du playbook

```
ansible-playbook -i inventory.ini my_playbook.yml
```

Tasks

Les tâches (tasks) sont exécutées dans le playbook et permettent d'effectuer diverses opérations sur le système cible.

Par exemple, voici une tâche pour mettre à jour des systèmes Debian/Ubuntu :

```
- hosts: servers
  become: yes
  tasks:
    - name: Update all packages
      apt:
        update_cache: yes
        upgrade: dist
```

Vous pouvez aussi créer un dossier par exemple :

```
- hosts: servers
  become: yes
  tasks:
    - name: Create a directory
      file:
        path: /my/directory
        state: directory
```

Ou exécuter des commandes :

```
- hosts: servers
  become: yes
  tasks:
    - name: Execute command
      shell: 'sleep 5'
```

Variables

Un des gros avantages d'Ansible est l'utilisation de variables. Pour cela, créez un fichier **vars.yaml** :

```
http_port: 80
max_clients: 200
```

Dans le playbook, vous pouvez importer votre fichier de variables :

```
- hosts: servers
  become: yes
  vars_files:
```

```
    - vars.yaml
  tasks:
    - name: Ensure Apache is installed
      apt:
        name: apache2
        state: present
```

Vous pouvez aussi passer des variables en paramètre lors de l'exécution du playbook plutôt que d'utiliser un fichier à part :

```
ansible-playbook my_playbook.yml -i inventory.ini -e "http_port=8080"
```

Facts

Les facts sont des variables automatiquement générées lors de la connexion avec l'hôte.

Ces variables peuvent être utilisées pour déployer certaines tâche selon certaines conditions telles que le système d'exploitation cible par exemple.

Vous pouvez afficher les facts d'un hôte spécifique grâce à la commande suivante :

```
ansible my_web_server -i inventory.ini -m setup
```

Par exemple, vous pouvez utiliser ces facts pour installer apache seulement sur les systèmes Debian :

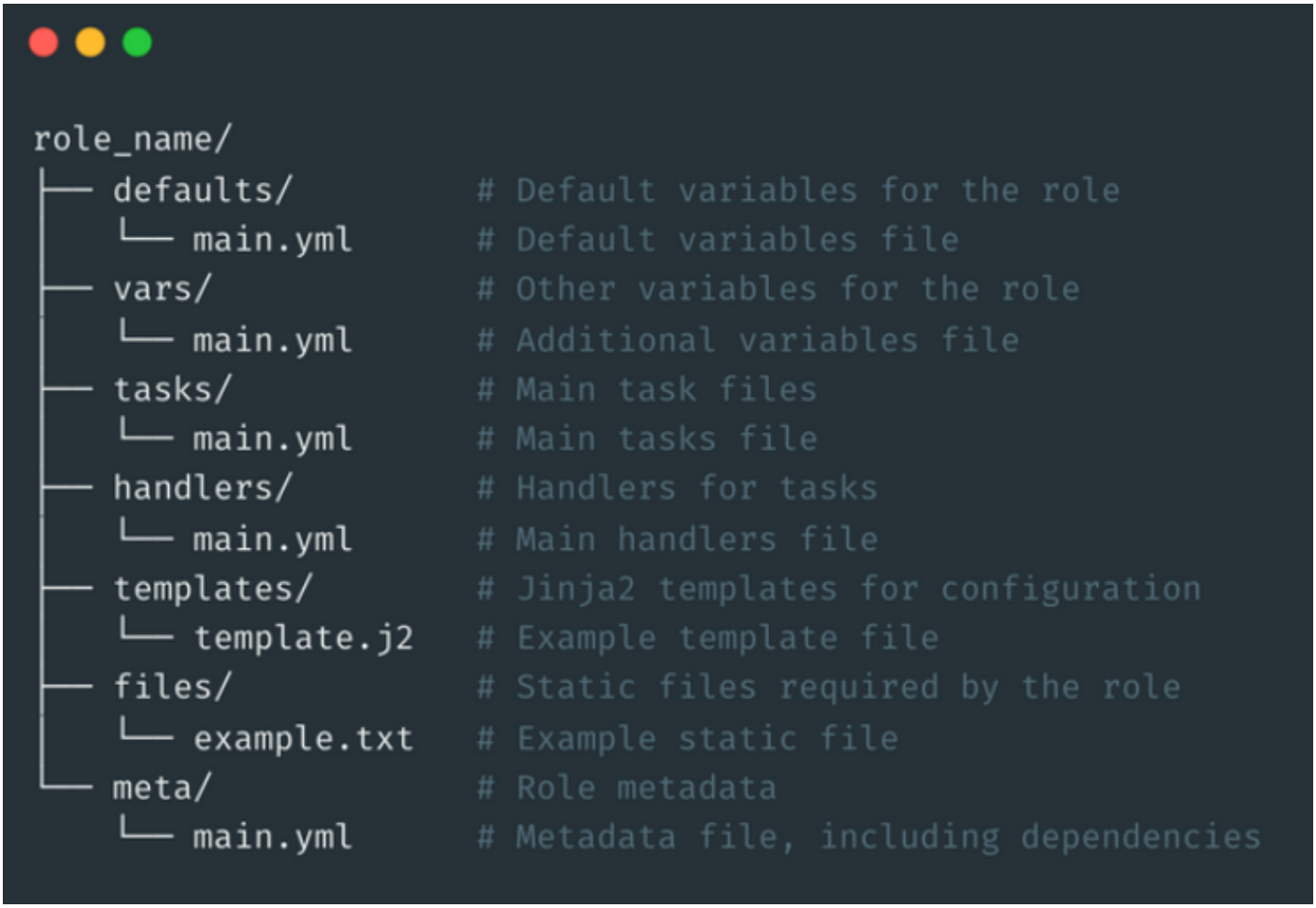
```
- hosts: servers
  become: yes
  tasks:
    - name: Ensure Apache is installed
      apt:
        name: apache2
        state: present
      when: ansible_os_family == "Debian"
```

Rôles

Les rôles sont des briques logiques de votre application qui servent à structurer votre projet en découpant le déploiement de votre application sous-tâches.

Par exemple, vous pourriez avoir besoin d'un serveur web et d'une base de donnée dans notre application, nous devrions donc créer un rôle respectif pour le déploiement de ces services.

Chacun des rôles a donc ces propres fichiers et sa propre arborescence comme le décrit le schéma ci-dessous :



On peut créer un rôle avec **ansible-galaxy** :

```
ansible-galaxy init <ROLE_NAME>
```

Vault

Il s'agit d'un plugin présent dans Ansible qui vous permet de sécuriser vos secrets.

Créer un secret

```
ansible-vault create secret.yml
```

Éditer le fichier secret

```
ansible-vault edit secret.yml
```

> Le format est le même que pour déclarer des variables.

Voir un secret

```
ansible-vault view secret.yml
```

Chiffrer un secret

```
ansible-vault encrypt secret.yml
```

Déchiffrer un secret

```
ansible-vault decrypt secret.yml
```

Créer un secret

```
ansible-vault create secret.yml
```

Utiliser un secret dans un playbook

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

> Le mot de passe du coffre vous sera demandé.

Vous pouvez aussi spécifier un fichier qui contient le mot de passe du coffre :

```
ansible-playbook playbook.yml --vault-password-file /path/to/creds
```

Inclure le fichier secret dans le fichier playbook

```
- hosts: all
  vars_file:
    - secrets.yml
```

```
  tasks:
    - name: Display the API key
      debug: "The API key is {{ api_key }}"
```

> Le secret contenu dans le fichier **secrets.yml** sera affiché.

Ansible Galaxy

Il s'agit d'une bibliothèque publique de rôle mis à disposition par la communauté qui peuvent être utilisés.

Installer un rôle

Vous pouvez installer un rôle pour **Nginx** :

```
ansible-galaxy install geerlingguy.nginx
```

Ou encore **MySQL** :

```
ansible-galaxy install geerlingguy.mysql
```

Inclure un rôle dans un playbook

```
- hosts: web_servers
  become: yes
  roles:
    - geerlingguy.nginx

- hosts: db_servers
  become: yes
  roles:
    - geerlingguy.mysql
```

Gestion d'erreurs

Ignorer les erreurs

```
- hosts: servers
  become: yes
  tasks:
    - name: Task that might fail
      command: /a/command/that/might/fail
      ignore_errors: yes
```

Contrôle d'erreurs

```
- hosts: servers
  become: yes
  tasks:
    - name: Execution of a script that always returns 0
      command: /my/script.sh
      register: script_result
      failed_when: "'ERROR' in script_result.stdout"
      changed_when: script.result.rc != 0
```

> Si le mot clé **ERROR** apparaît, la tâche va planter et se mettre en erreur.

Gestion fine des erreurs

```
- hosts: servers
  become: yes
  tasks:
    - name: Task block
      block:
        - debug:
            msg: 'I am about to run a task'
        - command: /a/command/that/might/fail
      rescue:
        - debug:
            msg: 'An error occurred while executing the command'
      always:
        - debug:
            msg: 'This message always displays after the bloc, success or failure'
```

Retries / Until

Cette instruction permet de spécifie de réessayer une tâche un certains nombre de fois, jusqu'à l'arrivé d'un certains événement :

```
- hosts: servers
  become: yes
  tasks:
    - name: Retry a task until it succeeds
      command: /a/temporary/unstable/command
      register: result
      until: result.rc == 0
      retries: 5
      delay: 10
```

> Ici, la tâche sera exécutée en boucle jusqu'à ce que le fact **result.rc** atteigne la valeur **0** ou que **5** tentatives soit effectuée.

> Un délai de 10 secondes sera mis entre chaque tentative.

Assertions

Les assertions sont des vérifications pour être sûr que la tâche se déroule normalement :

```
- hosts: servers
  become: yes
  tasks:
    - name: Verify that the variable 'my parameter' is defined
      assert:
        that:
          - my_parameter is defined
        fail_msg: "'my_parameter' is not defined"
        success_msg: "'my_parameter' is defined"
```

Handlers

Les handlers sont des tâches spéciales qui se déclenchent seulement lorsqu'elle est notifié par une autre tâche ce qui ajoute des fonctionnalités de dépendance car par défaut, les tâches sont exécutées de manière séquentielle :

```
- hosts: servers
  become: yes
  tasks:
    - name: Install a package
      apt:
        name: my_package
        state: latest
      notify: restart_my_service
  handlers:
    - name: restart_my_service
      service:
        name: my_service
        state: restarted
```

Debug

Le module de debug vous permet d'afficher les valeurs des variables lors de l'exécution des tâches :

```
- hosts: servers
  become: yes
  tasks:
    - name: Display the value of a variable
      debug:
        var: my_variable
```

Vous pouvez aussi utiliser le mode verbose de la commande ansible :

```
ansible-playbook my_playbook.yml -i inventory.ini -v
```

> Vous avez trois niveaux de verbosité (-v, -vv et -vvv).