COMPRENDRE ET CRÉER DES PLAYBOOKS

STRUCTURE D'UN PLAYBOOK

FICHIERS YAML

SYNTAXE

Les **playbooks** sont écrits en **YAML** (Ain't Markup Language). La syntaxe suit la structure suivante :

- key: value another_key:
  - element1
  - element2 ```

UTILISATION (ESPACEMENT, INDENTATIONS)

- Utiliser **2 espaces** pour les indentations.
- Ne pas utiliser de **tabulation**.
- Les blocs sont délimités par des tirets – et les clés sont suivies de `:`.
- Les listes sont délimitées par des tirets – et les éléments sont **indentés**.

HÔTES CIBLES

DÉFINITION

On définit les **hôtes** sur lesquels les tâches du **playbook Ansible** seront exécutées en utilisant une liste d'hôtes sous la clé `hosts`.

GESTION DES GROUPES D'HÔTES

Dans l'inventaire, on peut définir des **groupes d'hôtes** pour les cibler facilement dans les **playbooks**.

- hosts: groupe1:groupe2 ```

VARIABLES

DÉCLARATION ET UTILISATION

Les **variables** permettent d'injecter des **valeurs dynamiques** dans le playbook.

- hosts: all vars: var1: "valeur1" var2: "valeur2" tasks:
    - name: Utiliser les variables command: echo {{ var1 }} {{ var2 }} ```

PORTÉE DES VARIABLES

Les variables peuvent avoir différentes **portées**, comme :

- **Globale**
- **Playbook**
- **Rôle**
- etc.

PARAMÈTRES

PASSER DES ARGUMENTS AU PLAYBOOK

On peut passer des **arguments** à un playbook lors de l'exécution de `ansible-playbook`:

```
ansible-playbook playbook.yml --extra-vars "arg1=12 arg2=42"
```

UTILISER LES VALEURS DES ARGUMENTS DANS LE PLAYBOOK

Utiliser les arguments passés dans le playbook :

- hosts: all tasks:
    - name: Utiliser les arguments command: echo {{ arg1 }} {{ arg2 }} ```

COMPRENDRE ET CRÉER DES PLAYBOOKS

UTILISATION DES TÂCHES

TÂCHES ET ACTIONS

Les **tâches** sont des unités de travail exécutées par **Ansible** sur les hôtes cibles. Chaque tâche est une action réalisée par un **module**.

Tâche	Module utilisé	Description
Installer un package	apt/yum	Installe ou met à jour un paquet sur une machine
Copier un fichier	copy	Copie un fichier local vers un hôte distant
Exécuter une commande	command	Exécute une commande shell sur un hôte distant

SYNTAXE TÂCHES

```
- name: Tâche exemple
  module: argument=value
```

Tableau des modules courants :

Module	Description
apt	Gestion des paquets pour les distributions Debian
yum	Gestion des paquets pour les distributions Red Hat
service	Gérer les services avec le système d'init
copy	Copier des fichiers sur les hôtes distants
file	Définir les attributs de fichier et de répertoire
command	Exécuter une commande sur les hôtes distants

EXEMPLE D'ACTION

```
- name: Installer un paquet
  ansible.builtin.yum:
    name: httpd
    state: present
```

Ansible utilise des **actions** pour effectuer des tâches spécifiques sur les hôtes cibles.

LISTE DES ACTIONS INTÉGRÉES

Ansible intègre de nombreuses actions pour gérer divers services et systèmes. Consultez la documentation pour la liste complète Liste des modules Ansible.

Module	Description
file	Gère les fichiers et les répertoires
copy	Copie des fichiers sur l'hôte distant
template	Génère des fichiers à partir de modèles
command	Exécute une commande sur l'hôte distant
service	Gère les services sur l'hôte distant
user	Gère les comptes utilisateur
group	Gère les groupes d'utilisateurs

GESTION DES MODULES

UTILISATION DES MODULES ANSIBLE

Les modules Ansible sont inclus dans la **distribution standard**. Utilisez un module en spécifiant son nom dans la tâche :

```
- name: Copier un fichier
  ansible.builtin.copy:
    src: /etc/hosts
    dest: /tmp/hosts
```

CRÉATION DE MODULES PERSONNALISÉS

Vous pouvez créer des **modules personnalisés** dans un langage de programmation de votre choix. Un module personnalisé doit être ajouté à la **librairie de modules** de Ansible.

INCLUSION

INCLURE DES FICHIERS PLAYBOOKS

```
- name: Inclure un playbook
  import_playbook: other_playbook.yml
```

Note : Inclure un playbook permet de séparer les tâches en différents fichiers, améliorant l'organisation et la modularité du code.

INCLURE DES FICHIERS DE TÂCHES

```
- name: Inclure des tâches
  ansible.builtin.include_tasks: tasks_file.yml
```

SÉQUENTIALITÉ ET PARALLÉLISME

EXÉCUTION SÉQUENTIELLE DES TÂCHES

Les tâches sont exécutées dans l'**ordre défini** dans le **playbook**, pour chaque **hôte cible**.

EXÉCUTION PARALLÈLE DES TÂCHES

Utilisez les **stratégies** (`strategy`) pour contrôler l'ordre d'exécution des tâches sur les hôtes.

```
- hosts: all
  strategy: free
  tasks: ...
```

SYNCHRONISATION ENTRE TÂCHES

Utilisez `wait_for`, `wait_for_connection`, `wait_for_active_connection` pour synchroniser l'exécution des **tâches**.

Fonction	Description
wait_for	Attend qu'une condition spécifiée soit remplie
wait_for_connection	Attend qu'une connexion soit établie avec le noeud cible
wait_for_active_connection	Attend qu'une connexion soit active avant de continuer l'exécution

GESTION DES ERREURS

GÉRER L'ÉCHEC D'UNE TÂCHE

IGNORE_ERRORS

Permet d'**ignorer les erreurs** d'une tâche et de continuer l'exécution des tâches suivantes.

```yaml
- name: Exemple de tâche avec ignore_errors
  command: /usr/bin/somecommand
  ignore_errors: yes
```

RÉESSAYER EN CAS D'ÉCHEC

Utiliser les paramètres `retries` et `delay` pour réessayer une tâche en cas d'échec.

```
- name: Exemple de tâche avec réessai
  command: /usr/bin/somecommand
  register: result
  until: result.stdout != ""
  retries: 3
  delay: 2
```

UTILISER DES BLOCS POUR GÉRER L'ÉCHEC EN GROUPE

Les **blocs** permettent de regrouper plusieurs tâches et de définir des **stratégies de gestion d'erreurs** pour l'ensemble du groupe.

```
- block:
    - debug: msg='Executant les taches dans le bloc'
  rescue:
    - debug: msg='Une erreur est survenue dans le bloc'
```

GÉRER LES CHANGEMENTS

DÉTECTER LES CHANGEMENTS

Utiliser le paramètre `register` pour enregistrer l'état d'une tâche et `changed_when` pour déterminer si des changements ont été effectués.

```
- name: Exemple de détection de changements
  command: /usr/bin/somecommand
  register: result
  changed_when: "'modification detectée' in result.stdout"
```

EMPÊCHER LES CHANGEMENTS (CHECK MODE)

Le mode `check` permet d'exécuter un playbook sans appliquer de **modifications**.

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

TESTS DE VÉRIFICATION

ASSERTION ET VÉRIFICATION

Utiliser le module `assert` pour vérifier si une condition est respectée, sinon arrêter l'exécution du playbook.

```
- name: Exemple d'assertion
  assert:
    that:
      - 'result.stdout == "valeur attendue"'
    fail_msg: "La vérification a échoué"
```

LISTE DES TESTS INTÉGRÉS

Ansible fournit plusieurs tests pour vérifier les conditions. Exemples : `**equalto**`, `**truthy**`, `**match**`.

```
- name: Exemple de test intégré
  assert:
    that: "{{ item }} est match('regex')"
  with_items:
    - 'a'
    - 'b'
    - 'c'
```

CRÉATION DE TESTS PERSONNALISÉS

Il est possible de créer des tests personnalisés pour vérifier des conditions spécifiques. Les tests personnalisés doivent être créés dans un dossier `filter_plugins`.

```python
# filters/custom_test.py
def custom_test(value, param):
    return value == param

class FilterModule(object):
    def filters(self):
        return {
            'custom_test': custom_test,
        }
```

Utilisation du test personnalisé :

```yaml
- name: Exemple de test personnalisé
  assert:
    that: "{{ item }} est custom_test('paramètre')"
  with_items:
    - 'a'
    - 'b'
    - 'c'
```

BONNES PRATIQUES ET OPTIMISATIONS

STRUCTURATION DES DOSSIERS

Il est important d'organiser les **fichiers** et **dossiers** de manière claire et structurée pour faciliter la **gestion** et la **maintenance** des **playbooks Ansible**.

Exemple de structure :

Répertoire	Description
inventory	Contient les fichiers d'inventaire
group_vars	Variables spécifiques aux groupes d'hôtes
host_vars	Variables spécifiques aux hôtes individuels
roles	Contient les rôles Ansible
playbooks	Contient les playbooks Ansible principaux

RÔLES

- Contiennent les **tâches**, **variables**, **fichiers** et **modèles** liés à une fonction spécifique
- Peuvent être **réutilisés** et **partagés** entre différents playbooks
- Dossier: `roles`

PLAYBOOKS

- Contiennent des **plays** définissant des ensembles de tâches à exécuter sur des **hôtes**
- Dossier: `playbooks`

VARIABLES

- Variables et **valeurs spécifiques** à un **rôle**, un **play** ou un **environnement**
- Dossier: `group_vars` et `host_vars`

Type	Emplacement
Variables de rôle	`roles/{nom_du_rôle}/vars`
Variables de play	`playbooks/vars`
Variables de groupe	`group_vars`
Variables d'hôte	`host_vars`

TÂCHES

- **Contiennent** les actions à réaliser sur les **hôtes**
- Dossier : `tasks`

SÉPARER LES CONFIGURATIONS PAR ENVIRONNEMENT

La **gestion des configurations** est un élément-clé pour adapter un playbook en fonction des besoins spécifiques de chaque **environnement**.

- Utiliser des **variables** pour stocker les informations spécifiques à chaque environnement
- Utiliser des **fichiers de configuration** externes pour regrouper ces variables
- Créer un **répertoire** dédié à chaque environnement avec ses fichiers de configuration correspondants

Environnement	Répertoire	Fichier de configuration
Développement	dev	dev_vars.yml
Test	test	test_vars.yml
Production	production	prod_vars.yml

GESTION DES INVENTAIRES

- Inventaires définissant les **hôtes** et **groupes d'hôtes** pour chaque environnement
- Dossier: `inventories`

Type d'inventaire	Description
Production	Environnement de production
Staging	Environnement de pré-production / recette
Development	Environnement de développement

UTILISATION DE FICHIERS DE CONFIGURATION EXTERNES

Gérer les **configurations** et les **paramètres** dans des fichiers externes facilite la maintenance et la **portabilité** des playbooks.

Avantages	Exemples d'utilisation
Facilité de maintenance	Variables de rôles
Portabilité	Variables d'environnement
Éviter la duplication	Groupe de variables

FORMATS COMPATIBLES

- Formats de fichier pris en charge : `ini`, `yaml`, `json`

Format	Extensions	Caractéristiques
INI	.ini	Simple, hiérarchie sans imbriquage
YAML	.yaml, .yml	Indentation pour la hiérarchie, synthaxe simple
JSON	.json	Intégration JavaScript, plus verbeux que YAML

CHARGER DYNAMIQUEMENT DES FICHIERS DE CONFIGURATION

- Ansible peut charger des fichiers de configuration **externes** lors de l'exécution d'un **playbook** pour adapter les paramètres en fonction de l'environnement ou des besoins spécifiques.

Include_vars et **vars_files** sont deux options pour inclure des variables externes.

Option	Utilisation
include_vars	Inclure un fichier de variables spécifique à un certain état ou rôle
vars_files	Inclure un fichier de variables commun à plusieurs états ou rôles