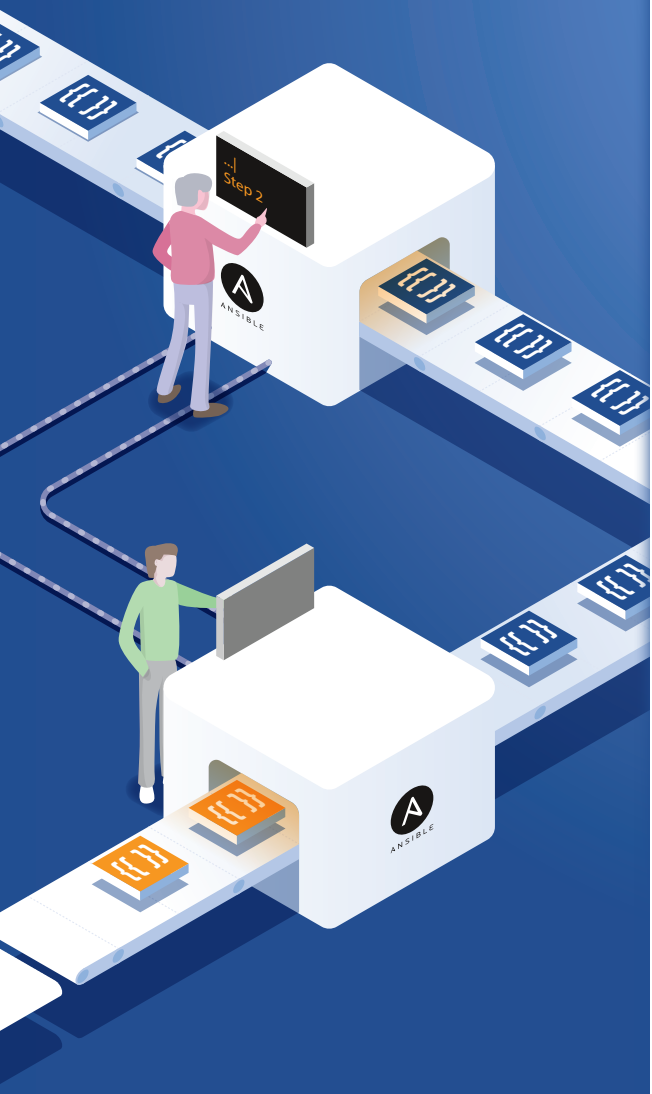


ANSIBLE

RefCard d'utilisation pour Ansible v2.7

Ecrit par
GERMAIN LEFEBVRE

MAI 2019

Ineat
NEW TECHNOLOGY, NEW LEADERS

Sommaire

Contexte

Voici la liste des versions des paquets utilisés pour réaliser la Refcard

Distribution et version : `cat /etc/redhat-release`

```
CentOS Linux release 7.5.1804 (Core)
```

Version Python : `python --version`

```
Python 2.7.5
```

Version Ansible : `ansible --version`

```
ansible 2.7.1
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/ansible/.ansible/plugins/
modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/
ansible
  executable location = /bin/ansible
  python version = 2.7.5 (default, Apr 11 2018, 07:36:10) [GCC 4.8.5
20150623 (Red Hat 4.8.5-28)]
```

Version Ansible à jour

Ansible dévoile sa Roadmap pour la v2.7 :
`https://docs.ansible.com/ansible/2.7/roadmap/ROADMAP_2_7.html`

Ansible met à disposition des guides de portage pour aider à rester à jour :

- `Ansible 2.0 Porting Guide`
- `Ansible 2.3 Porting Guide`
- `Ansible 2.4 Porting Guide`
- `Ansible 2.5 Porting Guide`
- `Ansible 2.6 Porting Guide`
- `Ansible 2.7 Porting Guide`

Définitions des objets

Facts
Les Facts sont des variables utilisées par Ansible pour persister des données entre les machines et leurs exécutions au sein d'une séquence d'un playbook. Chaque machine possède ses propres facts, comportant des données sur le système. Il est également possible d'injecter des facts.

Hosts
Les Hosts sont les serveurs joignables par le Master Ansible sur lesquels sont appliquées les actions.

Inventories
Les Inventories comportent la liste des serveurs identifiés par IP/FQDN et organisés dans des groupes. Les groupes peuvent être constitués de serveurs ou de groupes de serveurs. Les serveurs peuvent être aliasés pour faciliter la lisibilité globale.

Tasks
Les Tasks sont des actions exécutées sur les serveurs distants. Les tâches sont écrites en YAML. La structure descriptive des actions permet de faciliter la lecture, d'unifier et d'homogénéiser l'écriture.

Variables
Les Variables apportent la possibilité de modifier les valeurs au sein des tâches. Elles peuvent avoir une portée locale à une séquence ou une portée globale à tous les playbooks.

Plays
Les Plays sont des séquences d'actions, ils comportent des tâches et des inventaires et permettent d'appliquer une liste de tâches sur un ensemble de serveurs.

Playbooks
Les Playbooks regroupent des ensembles de Plays pour arriver à un but. Les Playbooks sont des fichiers lancés avec la commande ansible-playbook.

Rôles
Les Rôles sont des regroupements de tâches servant dans un même but. Ils sont appelés par les playbooks et permettent une meilleure lisibilité et facilité d'écriture. Les Rôles ont pour vocation de devenir génériques, réutilisables et personnalisables grâce aux variables.

Handlers
Les Handlers sont des actions appelées lorsqu'elles sont déclenchées par des tâches. Le déclenchement se fait à l'exécution d'une tâche, mais son exécution s'effectue à la fin de la séquence d'actions.

Modules
Les Modules sont des scripts écrits en Python qui constituent les tâches. Une tâche appelle un module à l'aide du YAML qui sera ensuite exécuté sur le serveur distant. Les modules permettent d'uniformiser les actions à appliquer.

Configuration de Ansible

Ansible peut être configuré à l'aide de fichiers ou de variables d'environnement système. La configuration suivra l'ordre de priorités suivantes (la plus forte valeur en premier) :

· Variables d'environnements
· ansible.cfg (répertoire courant)
· ~/.ansible.cfg (utilisateur courant)
· /etc/ansible/ansible.cfg

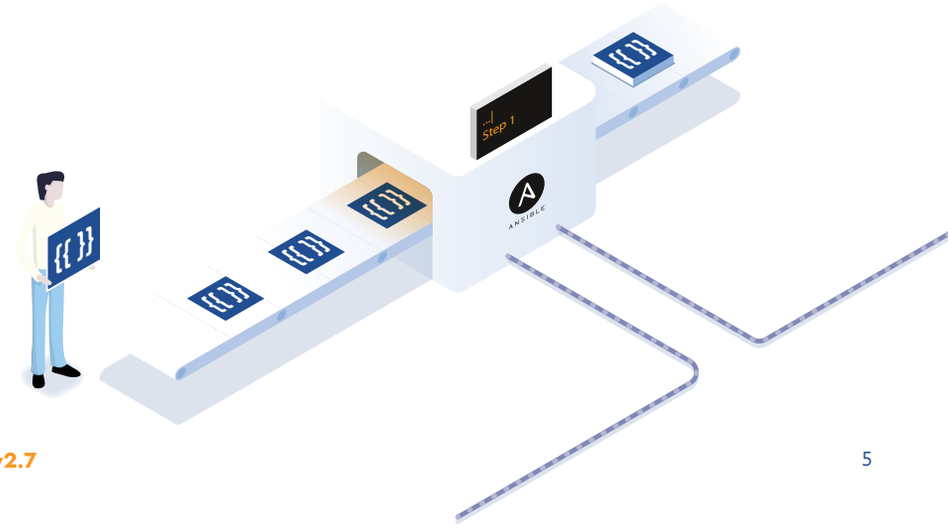
Ansible AdHoc Commands

Les commandes AdHoc sont lancées à la volée sans travail ni configuration préalable. Elles permettent de lancer des actions simples sur l'ensemble des serveurs. Il est nécessaire de lui passer en argument les serveurs (ou groupe de serveurs), le module à lancer (-m) et ses arguments (-a).

```
ansible localhost -m ping
ansible localhost -m setup
ansible all -i inventories/servers -m ping
```

Il est également possible de les joindre à des inventaires.

```
ansible all -i inventories/hosts -m debug -a 'myVar'
ansible redhat -i inventories/hosts -m shell -a 'uptime'
```

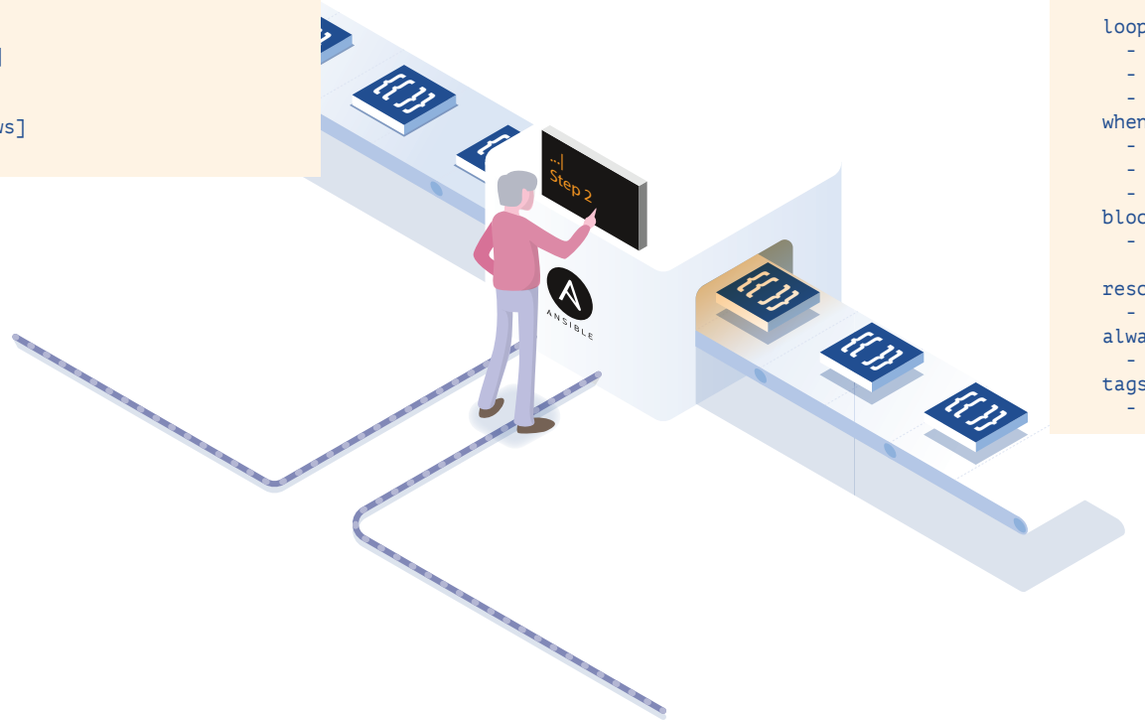


Ansible Inventories

Les inventaires rassemblent les tâches et rôles à appliquer sur des serveurs ou groupes de serveurs. Ils sont généralement organisés par environnement pour utiliser toute la puissance des variables et notamment des group_vars.

Définition d'un inventaire :

```
server.domain.fr
server[01-09].domain.fr

rhel01 rhel01.domain.fr
deb01 deb01.domain.fr
arch01 arch01.domain.fr
win01 win01.domain.fr

[redhat]
rhel01

[debian]
deb01

[linux:children]
redhat
debian

[linux]
arch01

[windows]
win01
```



Ansible Tasks

Une tâche Ansible est définie par une structure descriptive en YAML. L'objet suivant liste les options d'une tâche, seule l'utilisation des modules est indispensable.

Définition de Task :

```
- name: The task name to make things clearly
  become: yes
  become_method: sudo
  become_user: root
  check_mode: yes
  diff: no
  remote_user: ansible
  ignore_errors: True
  import_tasks: more_handlers
  include_tasks: other-tasks.yml
  notify: restart apache
  register: my_register
  changed_when: False
  failed_when: False
  vars:
    - myvar: toto
    myfiles:
      -  default.conf
  loop:
    - item1
    - ["item2", "item3"]
    - { name: "item4", description: "Desc Item 4" }
  when:
    - my_register is defined
    - ansible_distribution == "CentOS"
    - ansible_distribution_major_version == "7"
  block:
    - name: Task to run in block
      ...
  rescue:
    - name: Task when block failed
  always:
    - name: Task always run before/after block
  tags:
    - stuff
```

Ansible Playbooks

Un Playbook permet d'appliquer des tâches sur les serveurs de l'inventaire.

Installer un paquet sur les serveurs Redhat

```
- hosts: [redhat]
  become: true
  tasks
  - yum:
      name: httpd
      state: installed
```

Lancer le rôle Docker sur les serveurs Docker excepté le Master

```
- hosts: docker,!docker-master
  become: true
  roles:
  - docker
  - kubernetes
```

Ansible Variables

Variable Definition

Une variable Ansible peut être définie à plusieurs endroits, notamment dans les *group_vars, host_vars, role vars, CLI vars* et est appelée à l'aide du Templating Jinja : `{{ my_variable }}`. Les variables peuvent être appelées dans tous les objets Ansible (tasks, variables, template, ...).

Variable Typology

Une variable est de 3 types (python) : String (chaîne de caractères), Liste (Tableau d'objets), Dictionnaire (Tableau associatif d'objets).

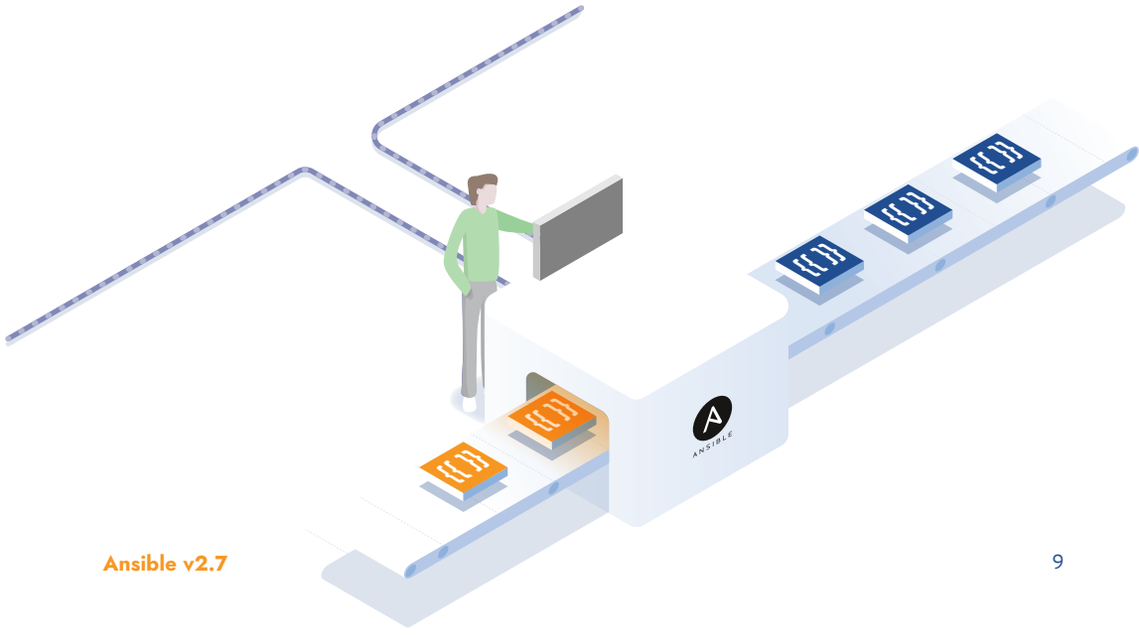
Une variable est définie par un couple Clé/Valeur où la Clé est normalisée ([a-z][A-Z][0-9][_-]) et la Valeur est une structure comme définie ci-dessus.

```
my_string: "value"
my_list: ["bob", "alice"]
my_list:
    - "bob"
    - "alice"
my_dict:
    item1: value1
    item2: value2
```

Variable precedence

Les variables Ansible peuvent être définies à plusieurs endroits comme `group_vars`, `playbooks`, `roles`, etc... et sont évaluées par priorité. Voici la liste des priorités en commençant par la plus faible :

```
Valeur en ligne de commande
Valeur par default du rôle
Fichier d'inventaire
Inventaire global group_vars/all
Playbook group_vars/all
Inventaires group_vars/*
Playbook group_vars/*
Fichier d'inventaire ou script host vars
Inventaire host_vars/*
Playbook host_vars/*
Facts serveur / set_facts en cache
Variables du Play
Variables vars_prompt du Play
Variables vars_files du Play
Variables du rôle (dans role/vars/main.yml)
Bloc de variables (seulement pour les blocs de tâches)
Variables des tâches (seulement pour les tâches)
Import de variables include_vars
Facts set_facts / Variables register
Paramètres de rôle and inclusion include_role
Import des paramètres
Extra vars argument des CLI (-e)
```



Ansible Plays

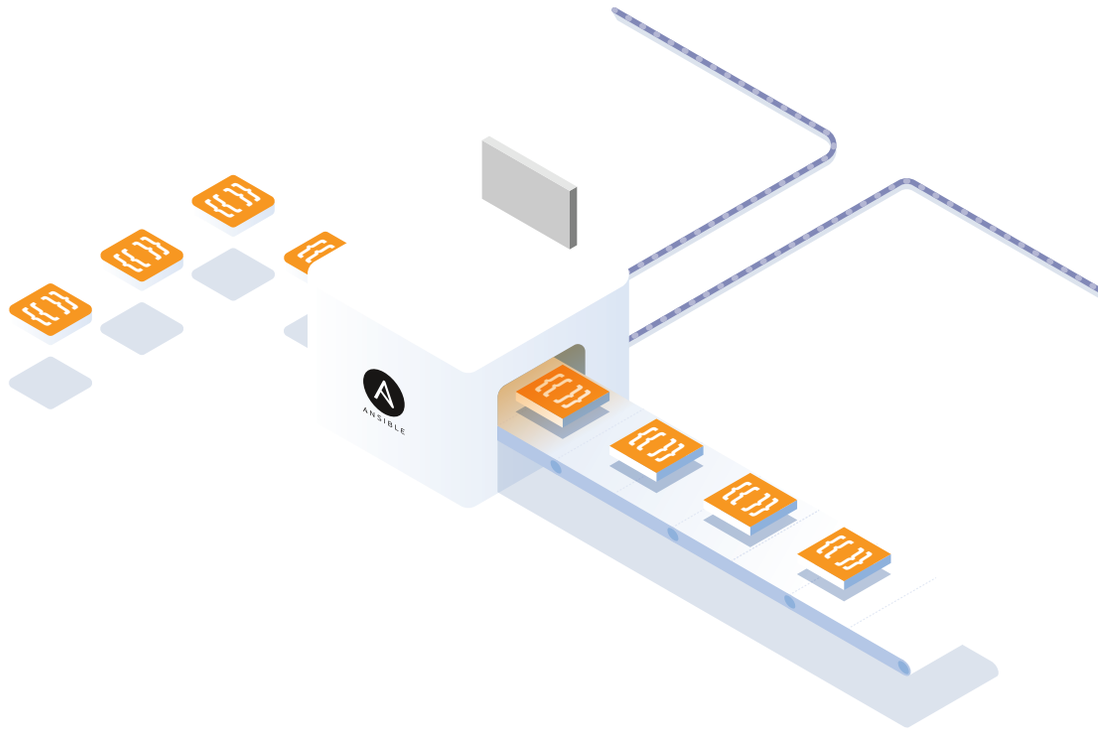
Une liste conséquente d'attributs d'un Play Ansible au sein d'un Playbook :

```
- hosts: webservers
  accelerate: no
  accelerate_port: 5099
  ansible_connection: local
  any_errors_fatal: True
  become: yes
  become_method: su
  become_user: postgress
  become_flags: True
  debugger: on_failed
  gather_facts: no
  max_fail_percentage: 30
  order: sorted
  remote_user: root
  serial: 5
  strategy: debug
  vars:
    http_port: 80
  vars_files:
    - "vars.yml"
  vars_prompt:
    - name: "my_password2"
      prompt: "Enter password2"
      default: "secret"
      private: yes
      encrypt: "md5_crypt"
      confirm: yes
      salt: 1234
      salt_size: 8
  tags:
    - stuff
  pre_tasks:
    - <task>
  roles:
    - common
    - common
      vars:
        port: 5000
      when: "bar == 'Baz'"
      tags : [one, two]
    - common
    - { role: common, port: 5000, when: "bar == 'Baz'",
      tags :[one, two] }
```

...

...

```
tasks:
    - include: tasks.yaml
    - include: tasks.yaml
      when: day == 'Thursday'
      vars:
        foo: aaa
        baz:
          - z
          - y
    - { include: tasks.yaml, foo: zzz, baz: [a,b]}
    - <task>
 post_tasks:
    - <task>
```



Ansible Rôles

Structure d'un rôle

Arborescence de répertoire d'un rôle Ansible :

```
roles/
└── my-role
    ├── defaults
    │   └── main.yml
    ├── files
    │   └── file
    ├── handlers
    │   └── main.yml
    ├── tasks
    │   └── main.yml
    ├── templates
    │   └── template.j2
    └── vars
        └── main.yml
```

Liste des fichiers potentiels et leurs fonctions au sein du rôle :

· `my-role/defaults/main.yml` définit les variables par défaut du rôle,
· `my-role/files/file` est un fichier (sans variables Jinja) à copier sur le serveur distant,
· `my-role/handlers/main.yml` définit les Handlers déclenchables,
· `my-role/tasks/main.yml` est le fichier de tâches par défaut appelé à l'appel du rôle,
· `my-role/templates/template.yml.j2` est un fichier (avec variables Jinja) à copier,
· `my-role/vars/main.yml` définit les variables à surcharger.

Génération et copie d'un Template

Fichier Template `roles/example/templates/my-template.sh.j2`

```
#!/bin/bash
echo "{{ timezone }}"
```

Fichier Task `roles/example/tasks/main.yml`

```
- template:
    src: my-template.sh.j2
    dest: /tmp/my-template.sh
```

Déclenchement d'un Handler

Fichier Handler `roles/example/handlers/main.yml.`

```
- name: Restart Apache
  systemd:
    name: httpd
    state: restart
```

Fichier Task `roles/example/tasks/main.yml.`

```
- copy:
    src: httpd.conf
    dest: /etc/httpd/conf/httpd.conf
  notify: Restart Apache
```

Le Handler `Restart Apache` sera déclenché à l'exécution de la Task `copy` (état `changed`).

Variables par défaut du rôle

Fichier Variables `roles/example/defaults/main.yml`

```
apache_version: '2.4.2'
```

Fichier Task `roles/example/tasks/main.yml`

```
- yum:
    name: httpd-{{ apache_version }}
    state: present
```

Import de Playbooks, Tasks et Rôles

Inclusion d'un playbook

Seul l'appel `import_playbook` permet d'inclure un Playbook entier dans un autre.

```
- import_playbook: install_apache.yml
```

Inclusion de tâches

Les appels `import_task` et `include_task` permettent d'inclure des tâches.

```
- hosts: [redhat]
  tasks:
  - import_tasks: roles/example/tasks/main.yml
  - include_tasks: roles/example/tasks/main.yml
```

Les 2 méthodes ont des particularités.

Inclusion de tâches en filtrant par tag

Seule l'appel `import_tasks` permet d'inclure des tâches et de les filtrer à l'aide de Tags.
Important : Les appels `include_tasks` nécessitent de comporter les tags à propager, sinon les tâches ne s'exécuteront pas.

Inclusion d'un rôle

Les appels `import_role` et `include_role` permettent d'inclure des tâches d'un rôle.

```
- hosts: [redhat]
  tasks:
  - import_role:
      name: example
  - include_role:
      name: example
```

Inclusion d'une partie d'un rôle

L'appel `include_role` permet d'inclure une partie d'un rôle en embarquant son écosystème : variables, handlers. L'attribut `tasks_from` permet de cibler le fichier de tâches à appeler à la place du `main.yml`. Il devient facile de personnaliser les rôles et leur donner plusieurs sens fonctionnels.

```
- hosts: [redhat]
  tasks:
  - include_role:
      name: example
      tasks_from: install
```

Ceci appelle le rôle `example` au travers du fichier de tâche `install.yml` inclus dans le rôle.

Inclusion de rôle en filtrant par tag

Seul l'appel `import_role` permet d'inclure un rôle tout en posant un filtre avec des tags.
Important : Les appels `include_role` nécessitent de comporter les tags à propager, sinon les tâches ne s'exécuteront pas.

```
- hosts: [redhat]
  tasks:
  - import_role:
      name: example
```

Ansible Options

Ansible offre des possibilités plus avancées qui aident à la vérification, au débuggage et poussent sur le plan non destructif des exécutions.

Filtre sur l'inventaire

L'option `--limit` ou `-l` filtre l'exécution du Playbook par serveur (host ou alias), groupe de l'inventaire et interprète parfaitement les exclusions.

```
ansible-playbook -i hosts.yml playbook.yml --limit 'linux,!debian'
```

Filtre par Tag

L'option `--tag` ou `-t` filtre l'exécution du Playbook par tag stipulé dans l'attribut `tags` : des Plays et Tasks. L'exclusion est possible sur le filtre par Tag.

```
ansible-playbook -i host.yml playbook.yml --tags 'config,service'
--skip-tags 'reload'
```

Mode Dry-Run

L'option `--check` déroule le Playbook sans effectuer la modification côté serveur et projette l'état de l'action (ok, changed, failed). Le mode Dry-Run est un bon moyen de tester la robustesse des scripts Ansible.

```
ansible-playbook -i host.yml playbook.yml --check
```

L'option Dry-Run fausse l'idempotence avec les modules `shell` et `commands` à cause du manque de visiblité d'Ansible vis-à-vis des commandes shell du script.

Mode Différences

L'option `--diff` affiche les différences constatées sur les actions liées aux fichiers (copy, template). Le mode des différences est une pratique utile pour constater les changements opérés sur les serveurs.

```
ansible-playbook -i host.yml playbook.yml --diff
```

Ansible Modules

Emplacement

Le module peut être stocké à plusieurs endroits :
· En local (répertoire courant) : Créer le Module Ansible où sont lancés les Playbooks dans le répertoire `library`.
· Coté Serveur (tous les utilisateurs) : Définir le chemin des modules dans l'attribut `library` du fichier `ansible.cfg`.

```
library = /usr/share/my_modules/
```

Instanciation de la Classe du Module

Le module à appeler correspond au nom du fichier du Module : `./library/myansiblemodule.py`.

La Classe du Module doit à minima comporter un constructeur et une méthode d'appel, ici `def process()`.

```
class MyAnsibleModule(AnsibleModule):
    def __init__(self, *args, **kwargs):
        self._output = []
        self._facts = dict()
        super(MyAnsibleModule, self).__init__(*args, **kwargs)
    def process(self):
        [...]
```

Exécution du module

```
if __name__ == '__main__':
  MyAnsibleModule().process()
```

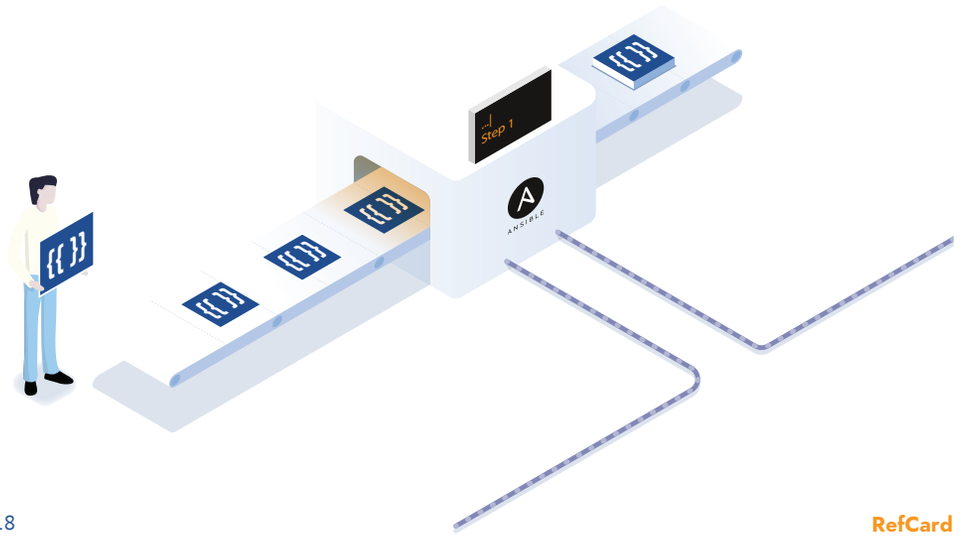
Définition des attributs

La définition des arguments du module se fait grâce à l'option `argument_spec` de la Classe. L'objet passé dans l'attribut doit être un dictionnaire.

```
MyAnsibleModule(
    argument_spec=dict(
        url=dict(type='str',
                 required=True),
        type=dict(type='str',
                  required=True,
                  choices=['indice', 'document']),
        state=dict(type='str',
                   choices=['present','absent'],
                   default='present')
    )
).process()
```

L'intéraction avec les arguments dans la méthode d'appel se fait avec la variable `self.params`.

```
def process(self):
    try:
        changed = False
        # Retrieve value for attribute 'url'
        param_url = self.params['url']
        self.exit_json(changed=changed, ansible_facts={},
        output=self._output)
    except Exception as e:
        self.fail_json(msg=(e.message, self._output))
```



Activation du mode Dry-Run

Pour utiliser le mode Dry-Run, il faut activer l'option `supports_check_mode` dans la Classe.

```
def main():
    MyAnsibleModule(
        supports_check_mode=True,
        [...]
    ).process()
```

L'intéraction avec le mode Dry-Run dans la méthode se fait avec la variable `self.check_mode`.

```
def process(self):
    # Stop if check_mode is true before doing any action
    if self.check_mode:
        return True
    # Keep doing actions if check_mode is false
```

Ansible Vault

Vault Configuration

La configuration du mot de passe dans Ansible Vault se fait sous plusieurs angles :

· Attribut `vault_password_file` dans le fichier `ansible.cfg`,
· Variable environnement `ANSIBLE_VAULT_PASSWORD_FILE`,
· Argument de commande `--vault-password-file` ou `--vault-id` avec le fichier contenant le mot de passe,
· Argument de commande `--ask-vault-pass` pour demander le mot de passe.

Vault Fichier

Un fichier peut être encrypté complètement.

```
ansible-vault create foo.yml
ansible-vault encrypt foo.yml
```

Une fois encrypté, il est possible de le lire ou le décrypter.

```
ansible-vault view foo.yml
ansible-vault decrypt foo.yml
```

Vault Variables

Une variable peut être encryptée, au lieu du fichier, pour faciliter l'accès au fichier en clair sans dévoiler les valeurs sensibles.

```
ansible-vault encrypt_string --name 'mykey' 'mysecret'
```

Ansible avec Vault

La configuration du mot de passe dans le système est importante pour utiliser Ansible sans le stipuler à chaque commande.

Si ce n'est pas le cas, il est nécessaire de l'inclure dans la commande.

```
ansible-playbook site.yml --ask-vault-pass
ansible-playbook site.yml --vault-password-file ~/.vault_pass
ansible-playbook --vault-id dev@dfile-dev-password site.yml
ansible-playbook --vault-id prod@prompt site.yml
```

Il est également possible de fournir plusieurs mot de passe pour dérouler les actions sur plusieurs environnements sans changer la commande entre les plateformes.

```
ansible-playbook --vault-id dev@dev-password --vault-id prod@prompt
site.yml
```

Extension par utilisation

Ansible est en premier lieu un outil de déploiement permettant d'homogénéiser les actions côté serveur et de centraliser la configuration des applications déployées.

Vérifications

Ansible peut cependant être utilisé pour une toute autre utilisation. L'implémentation du mode Dry-Run permet de vérifier l'état des serveurs sans apporter de modifications. Il devient facile de l'utiliser comme outil de vérification dans le but de contrôler tout changement opéré sur les serveurs.

Idempotence

L'idempotence est la facultée d'une action à ne pas appliquer de changement quand il n'est pas nécessaire. La répétabilité de l'action ne doit pas être destructrice (état changed) et doit remonter quant à sa bonne configuration (état ok).

Un playbook est idempotent dès lors qu'il ne provoque plus d'état changed à son passage. L'idempotence est dite parfaite quand, à un instant t, les actions sont changeantes sur le premier lancement et ne le sont plus sur tous les suivants.

Notes

Ineat

INEAT Lille - Euratechnologies
2, allée de la Haye du Temple
59160 Lomme - France
www.ineat-group.com

Conception graphique