

ANSIBLE



Votre partenaire formation ...

UNIX - LINUX - WINDOWS - ORACLE - VIRTUALISATION



www.spherius.fr

SOMMAIRE

PRÉSENTATION D'ANSIBLE.....	5
Introduction et concepts.....	7
INSTALLATION D'ANSIBLE.....	10
Pré-requis.....	12
Installation sous RedHat.....	13
Échange de clefs SSH.....	14
CONFIGURATION ET UTILISATION D'ANSIBLE.....	16
Le répertoire /etc/ansible.....	18
Les modules Ansible.....	21
Test de la connectivité.....	24
Le fichier d'inventaire.....	26
LES COMMANDES AD-HOC ET LES MODULES ANSIBLE.....	33
Les modules command et shell.....	35
Le transfert de fichiers.....	37
La gestion des packages.....	41
La gestion des utilisateurs.....	44
La gestion des services.....	46
Le module setup.....	48
LES PLAYBOOKS.....	51
Description d'un playbook.....	53
Les variables et les tableaux.....	61
La priorité et la portée des variables.....	65
Les templates.....	70
La boucle for.....	72
Le module debug et le mot clef register.....	74
Les Handlers.....	77
Les boucles.....	79
La condition when.....	84
Les include et les import.....	90
Les filtres.....	98
Les opérations arithmétiques.....	101
LES RÔLES.....	104
Présentation.....	106
Structure d'un rôle.....	107
Exécution d'un rôle.....	109
Un exemple de rôle.....	111
Un exemple de rôle avec des inclusions.....	113
Ansible Galaxy.....	116
FONCTIONNALITÉS AVANCÉES.....	121
Les tags.....	123
La visualisation d'un playbook.....	125
Gather_facts.....	126
La délégation par delegate_to.....	128
Les pré et post tasks.....	132
Le mot clef run_once.....	134
Le parallélisme.....	135

Le traitement avec serial.....	136
any_errors_fatal.....	138
Les blocks.....	140
La connexion avec un autre compte.....	143
Le prompt.....	145
Le fichier d'inventaire dynamique et temporaire.....	146
lookup.....	150
set_fact.....	155
Test de cohérence : assert et fail.....	157
dry-run, step-by-step et diff.....	161
COMPLÉMENTS.....	164
La création d'un module.....	166
Ansible Vault et l'encryptage.....	171
FIN DU SUPPORT DE COURS.....	179

Ce document est sous Copyright :

Toute reproduction ou diffusion, même partielle, à un tiers est interdite sans autorisation écrite de Sphérius. Pour nous contacter, veuillez consulter le site web <http://www.spherius.fr>.

Les logos, marques et marques déposées sont la propriété de leurs détenteurs.

Les auteurs de ce document sont :

- Monsieur Baranger Jean-Marc,
- Monsieur Schomaker Theo.

La version d'Ansible utilisée pour les commandes de ce support de cours est :

Ansible 2.7

Les références sont : les documents disponible sur le site web d'Ansible et de RedHat.

Présentation d'Ansible

Dans ce chapitre, nous allons présenter les principes et les concepts d'Ansible.

Présentation d'Ansible

- Introduction et concepts

Présentation d'Ansible

Introduction et concepts

- Simple Puissant Sans agent
- Automatise le déploiement, la configuration, la gestion, la maintenance de toute votre infrastructure
- Opérations en parallèle et simultanément sur toutes les machines de votre parc
- YAML Module Playbook



Introduction et concepts

Ansible est un outil de gestion de parc de machines sous licence GNU GPL.

Il automatise le déploiement, la configuration, la gestion et la maintenance de toute votre infrastructure. Ces opérations peuvent se faire en parallèle et simultanément sur toutes les machines de votre parc.

Ansible est une solution complète et robuste qui reste assez simple à prendre « en main ».

Ci-dessous une présentation issue du site d'Ansible :

Ansible est une solution d'automatisation informatique que vous pouvez apprendre rapidement. Il est assez simple pour tous les membres de votre équipe informatique, mais suffisamment puissant pour automatiser les déploiements les plus complexes. Ansible gère les tâches répétitives, donnant à votre équipe plus de temps pour se concentrer sur l'innovation.

Avec Ansible vous pouvez commencer à faire du vrai travail en quelques minutes grâce à son langage simple et lisible. Ses puissantes fonctionnalités permettent l'orchestration de l'ensemble du cycle de vie de votre application, quel que soit l'emplacement du déploiement. L'architecture sans agent d'Ansible signifie que c'est une chose de moins à gérer en terme de sécurité.

Ansible est un moteur d'automatisation informatique radicalement simple qui automatise l'approvisionnement en cloud, la gestion de la configuration, le déploiement d'applications, l'orchestration intra-service et de nombreux autres besoins informatiques.

Conçu pour les déploiements à plusieurs niveaux depuis le premier jour, Ansible modélise votre infrastructure informatique en décrivant comment tous vos systèmes interagissent, plutôt que de gérer un seul système à la fois.

Il n'utilise aucun agent et aucune infrastructure de sécurité personnalisée supplémentaire, il est donc facile à déployer - et surtout, il utilise un langage très simple (YAML, sous la forme d'Ansible Playbooks) qui vous permet de décrire vos tâches d'automatisation.

Ansible fonctionne en se connectant à vos nœuds et en poussant de petits programmes, appelés "Modules Ansible". Ces programmes sont écrits pour être des modèles de ressources de l'état souhaité du système. Ansible exécute ensuite ces modules (via SSH par défaut) et les supprime une fois terminé.

Votre bibliothèque de modules peut résider sur n'importe quel ordinateur et aucun serveur, démon ou base de données n'est requis. En règle générale, vous travaillez avec votre système d'exploitation préféré, un éditeur de texte et probablement un système de contrôle de versions pour suivre les modifications apportées à votre contenu.

Notes

Installation d'Ansible

Dans ce chapitre, nous allons installer Ansible sur Linux.

Installation d'Ansible

- Pré-requis
- Installation sous Redhat
- Échange de clefs SSH

Installation d'Ansible

Pré-requis

- version de python ≥ 2.6
- accès aux dépôts

Pré-requis

Ansible nécessite une version de python supérieure à 2.6.

Version de python

```
# python --version  
Python 2.7.5
```

L'installation se fait soit à partir de dépôts logiciels soit à partir de sources.

Installation d'Ansible

Installation sous RedHat

- Installer le Dépôt EPEL (Extra Package Enterprise Linux)
- Installer Ansible avec yum
- Vérification

Installation sous RedHat

Les packages qui sont nécessaires sont localisés sur le dépôt EPEL. L'une des méthodes ci-dessous permet de l'installer.

```
# wget http://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
# rpm -ivh epel-release-latest-7.noarch.rpm
```

Ou

```
# yum install epel-release
```

L'installation d'Ansible se fait via la commande suivante.

```
# yum install -y ansible
```

La configuration de base d'Ansible se trouve dans le fichier /etc/ansible.

```
# ls /etc/ansible/
ansible.cfg  hosts  roles
```

Pour tester l'installation ou connaître la version d'Ansible.

```
# ansible --version
```

Installation d'Ansible

Échange de clefs SSH

- Génération de clefs : ssh-keygen

```
# ssh-keygen -t rsa
```

- Envoi de la clef publique sur les machines distantes.

```
# ssh-copy-id -i /root/.ssh/id_rsa.pub 192.168.1.10
```

Échange de clefs SSH

Ansible utilise des clefs SSH pour communiquer avec les autres machines. Il faut d'abord les générer puis les envoyer sur les serveurs à administrer.

Création de la paire de clefs RSA :

```
# ssh-keygen -t rsa
```

Les clefs sont créées et disponibles au sein du répertoire /root/.ssh.

La clef privée est le fichier id_rsa, à ne pas diffuser.

La clef publique est le fichier id_rsa.pub. C'est ce fichier que l'on diffuse et qui est stocké dans le fichier authorized_keys de l'utilisateur du poste distant (\$HOME/.ssh/authorized_keys).

Envoi de la clef publique sur les serveurs distants :

```
# ssh-copy-id -i /root/.ssh/id_rsa.pub 192.168.1.10
# ssh-copy-id -i /root/.ssh/id_rsa.pub debian1
# ssh-copy-id centos2 # si positionné dans le home directory
```

Envoi de la clef publique sur un serveur distant pour l'utilisateur user1 :

```
# ssh-copy-id -i /root/.ssh/id_rsa.pub user1@192.168.1.10
```

Notes

Configuration et utilisation d'Ansible

Dans ce chapitre, nous allons effectuer nos premiers pas avec Ansible.

Configuration et utilisation d'Ansible

- Le répertoire /etc/ansible
- Les modules Ansible
- Test de la connectivité
- Le fichier d'inventaire

Configuration et utilisation d'Ansible

Le répertoire /etc/ansible

```
# tree /etc/ansible
/etc/ansible
├── ansible.cfg      fichier de configuration
├── hosts            fichier d'inventaire par défaut
└── roles            répertoires pour les rôles
```

Le répertoire /etc/ansible

Suite à l'installation d'Ansible, le répertoire /etc/ansible contient deux fichiers de configuration et le répertoire roles.

```
# tree /etc/ansible
/etc/ansible
├── ansible.cfg      fichier de configuration
├── hosts            fichier d'inventaire par défaut
└── roles            répertoires pour les rôle
```

La configuration d'Ansible est stockée dans /etc/ansible/ansible.cfg.

Le fichier hosts contient les serveurs à administrer. Il contient des exemples commentés de déclaration de serveurs ou de groupes de serveurs.

Le répertoire roles est vide pour l'instant.

Il contiendra les rôles qui permettront d'inclure des dépendances de tâches.

Le fichier de configuration :

```
# more /etc/ansible/ansible.cfg
# config file for ansible -- https://ansible.com/
# =====

# nearly all parameters can be overridden in ansible-playbook
# or with command line flags.  ansible will read ANSIBLE_CONFIG,
# ansible.cfg in the current working directory, .ansible.cfg in
# the home directory or /etc/ansible/ansible.cfg, whichever it
# finds first

[defaults]

# some basic default values...

#inventory           = /etc/ansible/hosts
#library             = /usr/share/my_modules/
#module_utils        = /usr/share/my_module_utils/
#remote_tmp          = ~/.ansible/tmp
#local_tmp           = ~/.ansible/tmp
#forks               = 5
#sudo_user           = root
#ask_sudo_pass       = True
...
```

Le fichier `/etc/ansible/ansible.cfg` contient la configuration principale d'Ansible. Il définit le comportement par défaut. Par exemple, il indique quel fichier (`inventory=`) va contenir la liste des hôtes à contrôler avec Ansible.

Dans l'ordre de sollicitation, le fichier de configuration utilisé est défini par :

- la variable d'environnement `ANSIBLE_CONFIG`,
- le fichier `ansible.cfg` dans le répertoire courant,
- le fichier `.ansible.cfg` du répertoire de connexion de l'utilisateur,
- le fichier `/etc/ansible/ansible.cfg`.

```
# ansible-config view
# config file for ansible -- https://ansible.com/
# =====

# nearly all parameters can be overridden in ansible-playbook
# or with command line flags.  ansible will read ANSIBLE_CONFIG,
# ansible.cfg in the current working directory, .ansible.cfg in
# the home directory or /etc/ansible/ansible.cfg, whichever it
# finds first

[defaults]

# some basic default values...

#inventory           = /etc/ansible/hosts
#library             = /usr/share/my_modules/
#module_utils        = /usr/share/my_module_utils/
#remote_tmp          = ~/.ansible/tmp
#local_tmp           = ~/.ansible/tmp
#forks               = 5
#poll_interval       = 15
#sudo_user           = root
#remote_port         = 22
#module_lang         = C
...
```

Le fichier d'inventaire :

Le fichier `/etc/ansible/hosts` est le fichier d'inventaire d'Ansible, il contient la liste des machines sous le contrôle d'Ansible. Elles peuvent être rassemblées par groupes.

```
# more /etc/ansible/hosts
[sphერიus_servers]          # nom du groupe de serveurs
deb_server                # machines constituant le groupe
CentOS6.5
CentOS7.1

[centos_servers]
CentOS6.5
CentOS7.1

[tests_machines]
hote1
hote2
```

Le fichier d'inventaire peut-être au format INI (exemple ci-dessus) ou au format YAML.

Le répertoire roles :

Le répertoire roles est vide pour l'instant. Il contiendra les rôles qui permettront d'inclure des dépendances de tâches. Ce point est développé plus tard dans ce support.

Configuration et utilisation d'Ansible

Les modules Ansible

- <http://docs.ansible.com/ansible/latest/modules.html>
- http://docs.ansible.com/ansible/latest/list_of_all_modules.html

```
# ansible-doc -l
```

```
# ansible-doc nom_module
```

Les modules Ansible

Un module Ansible est écrit en Python.

Les modules permettent d'effectuer des tâches sur les serveurs.

Le site d'Ansible fournit des informations sur les modules, ainsi que les mots clefs exploitables pour un module et des exemples.

- <http://docs.ansible.com/ansible/latest/modules.html>
- http://docs.ansible.com/ansible/latest/list_of_all_modules.html

Il est fortement recommandé de se référer régulièrement aux informations de ce site.

En plus de la documentation de tous les modules, le site d'Ansible présente de manière détaillée les nouvelles fonctionnalités.

Ansible

2.7

For previous versions, see the [documentation archive](#).

INSTALLATION, UPGRADE & CONFIGURATION

- ▣ Installation Guide
- ▣ Configuring Ansible
- ▣ Ansible Porting Guides

USING ANSIBLE

▣ User Guide

- Ansible Quickstart
- ▣ Getting Started
- ▣ Working with Command Line Tools
- ▣ Introduction To Ad-Hoc Commands
- ▣ Working with Inventory
- ▣ Working With Dynamic Inventory
- ▣ Working With Playbooks
- ▣ Understanding Privilege Escalation
- ▣ Ansible Vault
- Working with Patterns

[Docs](#) » [Module Index](#)

Module Index

- [All modules](#)
- [Cloud modules](#)
- [Clustering modules](#)
- [Commands modules](#)
- [Crypto modules](#)
- [Database modules](#)
- [Files modules](#)
- [Identity modules](#)
- [Inventory modules](#)
- [Messaging modules](#)
- [Monitoring modules](#)
- [Net Tools modules](#)
- [Network modules](#)
- [Notification modules](#)
- [Packaging modules](#)
- [Remote Management modules](#)
- [Source Control modules](#)
- [Storage modules](#)
- [System modules](#)
- [Utilities modules](#)
- [Web Infrastructure modules](#)
- [Windows modules](#)

La commande **ansible-doc** fournit des informations en ligne de commande sur les modules et les plugins utilisables par Ansible.

L'option **-l** liste tous les plugins disponibles :

```
# ansible-doc -l
a10_server          Manage A10 Networks AX/SoftAX/Thunder/vThunder devices' server object.
a10_server_axapi3   Manage A10 Networks AX/SoftAX/Thunder/vThunder devices
a10_service_group   Manage A10 Networks AX/SoftAX/Thunder/vThunder devices' service groups.
```

Le nombre de plugins disponibles dépend de la version d'Ansible.

```
# ansible-doc -l | wc -l
2080
```

La documentation relative à un module se fait par un appel via le nom du module :

```
# ansible-doc acl
> ACL      (/usr/lib/python2.7/site-packages/ansible/modules/files/acl.py)

    Sets and retrieves file ACL information.

OPTIONS (= is mandatory):

- default
    if the target is a directory, setting this to yes will make it the default acl
for entities created inside the
    directory. It causes an error if path is a file.
    (Choices: yes, no) [Default: False]
    version_added: 1.5
...
```

L'option **-s** permet d'avoir des informations sur l'utilisation du module à l'intérieur d'un playbook :

```
# ansible-doc -s acl
- name: Sets and retrieves file ACL information.
  acl:
    default:          # if the target is a directory, setting this to yes will
make it the default acl for entities created inside the directory.
                    It causes an error if path is a file.
    entity:           # actual user or group that the ACL applies to when matching
entity types user or group are selected.
...
```

Configuration et utilisation d'Ansible

Test de la connectivité

- Utilisation du module ping sur tous les serveurs

```
# ansible all -m ping
```

- Spécifier une liste de machines

```
# ansible CentOS7.1_server:hotel:deb_server -m ping
```

- Exclure des machines

```
# ansible 'spherius_servers:!CentOS6.5' -m ping
```

Test de la connectivité

Exemple du module (option -m) ping sur tous les serveurs déclarés (mot clef all) dans ansible.

```
# ansible all -m ping

deb_server | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
CentOS7.1_server | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
CentOS6.5_server | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
hote2 | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: ssh: Could not resolve hostname hote2:
Name or service not known\r\n",
  "unreachable": true
}
hotel | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: ssh: Could not resolve hostname hotel:
Name or service not known\r\n",
  "unreachable": true
}
```

Les messages d'erreurs sont normaux. Les hôtes hôte1 et hôte2 ne sont pas démarrés et les clefs ssh n'ont pas été échangées. Si tout va bien la réponse est **pong**.

L'option **--one-line** permet d'avoir l'affichage sur une seule ligne.

```
# ansible all -m ping --one-line
deb_server | SUCCESS => {"changed": false, "ping": "pong"}
CentOS6.5_server | SUCCESS => {"changed": false, "ping": "pong"}
CentOS7.1_server | SUCCESS => {"changed": false, "ping": "pong"}
hotel | UNREACHABLE!: Failed to connect to the host via ssh: ssh: connect to host hotel
port 22: No route to host

hote2 | UNREACHABLE!: Failed to connect to the host via ssh: ssh: connect to host hote2
port 22: Connection timed out
```

Le mot clef **all** peut être remplacé par un nom de machine ou de groupe de machines.

```
# ansible spherius_servers -m ping
deb_server | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
CentOS6.5_server | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
CentOS7.1_server | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

```
# ansible CentOS7.1_server -m ping
CentOS7.1_server | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

Le caractère **:** permet de spécifier une liste.

```
# ansible CentOS7.1_server:hotel:deb_server -m ping
deb_server | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
CentOS7.1_server | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
hotel | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: ssh: Could not resolve hostname hotel:
Name or service not known\r\n",
  "unreachable": true
}
```

Le caractère **!** permet de signifier une exclusion (les simples quotes sont indispensables).

```
# ansible 'spherius_servers:!CentOS6.5' -m ping
deb_server | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
CentOS7.1 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

Configuration et utilisation d'Ansible

Le fichier d'inventaire

- /etc/ansible/hosts
- L'option -i

```
# ansible all -i mon_inventaire.inv -m ping
```

- Une simplification d'écriture : `poste[5:15]`
- Pour lister les machines : `--list-hosts`
- Pour lister les machines avec leur groupe : `-m debug -a "var=groups"`
`ungrouped`
- Pour afficher la valeur d'une variable : `-m debug -a "var=nom_variable"`

```
# ansible-inventory all -i mon_inventaire.inv --graph --vars
# ansible-inventory all -i mon_inventaire.inv --list
```

Le fichier d'inventaire

Le fichier d'inventaire contient la liste des machines sous le contrôle d'Ansible. Elles peuvent être rassemblées par groupes.

Le fichier d'inventaire par défaut est : /etc/ansible/hosts

```
# more /etc/ansible/hosts
[sphერიus_servers]      # nom du groupe de serveurs
deb_server             # machines constituant le groupe
CentOS6.5
CentOS7.1

[centos_servers]
CentOS6.5
CentOS7.1

[tests_machines]
hote1
hote2
```

Le module ping permet de vérifier que les machines clientes ont les pré-requis nécessaires pour fonctionner avec Ansible.

```
# ansible all -m ping
```

Un fichier d'inventaire spécifique :

L'option « -i » permet d'indiquer un autre fichier d'inventaire que celui par défaut.

```
# cat mon_inventaire.inv
client1
client2

# ansible all -i mon_inventaire.inv -m ping
```

Une simplification d'écriture :

poste[5:15] pour définir les machines poste5 à poste15

```
# cat mon_inventaire.inv
client1
client2

[domaine1]
poste1
poste2
poste3

[domaine2]
poste4
poste5
poste6

est équivalent à

client[1:2]

[domaine1]
poste[1:3]

[domaine2]
poste[4:6]
```

Pour lister les machines : --list-hosts

```
# ansible all -i mon_inventaire.inv --list-hosts
hosts (8):
  client1
  client2
  poste4
  poste5
  poste6
  poste1
  poste2
  poste3

# ansible domaine1 -i mon_inventaire.inv --list-hosts
hosts (3):
  poste1
  poste2
  poste3
```

Le module debug permet d'afficher certaines variables de l'inventaire.

Pour lister les machines avec leurs groupes : `-m debug -a "var=groups"`

- `all` représente la liste de toutes les machines au sein du fichier d'inventaire.
- `ungrouped` représente la liste des machines associées à aucun groupe.

```
# ansible localhost -i mon_inventaire.inv -m debug -a "var=groups"
localhost | SUCCESS => {
  "groups": {
    "all": [
      "client1",
      "client2",
      "poste4",
      "poste5",
      "poste6",
      "poste1",
      "poste2",
      "poste3"
    ],
    "domaine1": [
      "poste1",
      "poste2",
      "poste3"
    ],
    "domaine2": [
      "poste4",
      "poste5",
      "poste6"
    ],
    "ungrouped": [
      "client1",
      "client2"
    ]
  }
}
```

Un exemple plus complet :

```
# cat mon_inventaire.inv
postel
poste2

[all:vars]
ansible_user=root

[deb_servers]
deb_server1
deb_server2

[domaine1]
apache1  apache_url=intra.domaine http_port=80 https_port=443
mysql1
centos_6.5  ansible_user=user1
centos_7.1

[linux:children]
domaine1
deb_servers

[linux:vars]
ntp_server=0.fr.pool.ntp.org

[windows]
serveur1
basededonnee1

[windows:vars]
ansible_connection=winrm
ansible_user=Administrator
```

Afficher les machines de l'inventaire.

```
# ansible all -i mon_inventaire.inv --list-hosts
hosts (10):
  postel
  poste2
  serveur1
  basededonnee1
  apache1
  mysql1
  centos_6.5
  centos_7.1
  deb_server1
  deb_server2
```

Afficher les groupes de machines de l'inventaire.

```
# ansible localhost -i mon_inventaire.inv -m debug -a "var=groups"
localhost | SUCCESS => {
  "groups": {
    "all": [
      "postel",
      "poste2",
      "serveur1",
      "basededonnee1",
      "apache1",
      "mysql1",
      "centos_6.5",
      "centos_7.1",
      "deb_server1",
      "deb_server2"
    ],
    "deb_servers": [
      "deb_server1",
```

```
        "deb_server2"
    ],
    "domaine1": [
        "apache1",
        "mysql1",
        "centos_6.5",
        "centos_7.1"
    ],
    "linux": [
        "apache1",
        "mysql1",
        "centos_6.5",
        "centos_7.1",
        "deb_server1",
        "deb_server2"
    ],
    "ungrouped": [
        "poste1",
        "poste2"
    ],
    "windows": [
        "serveur1",
        "basededonnee1"
    ]
}
}
```

Pour afficher la valeur d'une variable spécifique : `-m debug -a "var=nom_variable"`

```
# ansible all -i inventaire.inv -m debug -a "var=ansible_user" --one-line
poste1 | SUCCESS => {  "ansible_user": "root",      "changed": false}
serveur1 | SUCCESS => {  "ansible_user": "Administrator",  "changed": false}
poste2 | SUCCESS => {  "ansible_user": "root",      "changed": false}
apache1 | SUCCESS => {  "ansible_user": "root",      "changed": false}
basededonnee1 | SUCCESS => {  "ansible_user": "Administrator",  "changed": false}
mysql1 | SUCCESS => {  "ansible_user": "root",      "changed": false}
centos_7.1 | SUCCESS => {  "ansible_user": "root",      "changed": false}
centos_6.5 | SUCCESS => {  "ansible_user": "user1",      "changed": false}
deb_server1 | SUCCESS => {  "ansible_user": "root",      "changed": false}
deb_server2 | SUCCESS => {  "ansible_user": "root",      "changed": false}
```

La commande ansible-inventory

```
# ansible-inventory all -i inventaire.inv --graph
@all:
  |--@domaine1:
  |   |--basededonnee1
  |   |--deb_client1
  |   |--deb_client2
  |   |--serveur1
  |--@linux:
  |   |--@debian:
  |   |   |--deb_client1
  |   |   |--deb_client2
  |   |--@domaine2:
  |   |   |--apache1
  |   |   |--centos_6.5
  |   |   |--centos_7.1
  |   |   |--mysql1
  |--@ungrouped:
  |--@windows:
  |   |--basededonnee1
  |   |--serveur1
```

Pour un affichage avec les variables :

```
# ansible-inventory all -i inventaire.inv --graph --vars
@all:
  |--@domaine1:
  |   |--basededonnee1
  |   |--deb_client1
  |   |--deb_client2
  |   |--serveur1
  |--@linux:
  |   |--@debian:
  |   |   |--deb_client1
  |   |   |--deb_client2
  |   |--@domaine2:
  |   |   |--apache1
  |   |   |--{apache_url = intra.domaine}
  |   |   |--{http_port = 80}
  |   |   |--{https_port = 443}
  |   |   |--centos_6.5
  |   |   |--{ansible_user = user1}
  |   |   |--centos_7.1
  |   |   |--mysql1
  |--@ungrouped:
  |--@windows:
  |   |--basededonnee1
  |   |--serveur1
  |   |--{ansible_connection = winrm}
  |   |--{ansible_user = Administrator}
  |--{ansible_user = root}
```

Il existe évidemment d'autres options, telle que « --list » pour un affichage détaillé.

```
# ansible-inventory all -i inventaire.inv --list
```

Notes

Les commandes Ad-Hoc et les modules Ansible

Dans ce chapitre, nous allons étudier l'utilisation des modules Ansible.

Les commandes Ad-Hoc et les modules Ansible

- Les modules command et shell
- Le transfert de fichiers
- La gestion des packages
- La gestion des utilisateurs
- La gestion des services
- Le module setup

Les commandes Ad-Hoc et les modules Ansible

Les modules command et shell

- Le module command Le module shell
- http://docs.ansible.com/ansible/latest/command_module.html
- http://docs.ansible.com/ansible/latest/shell_module.html

```
# ansible-doc command
```

```
# ansible-doc shell
```

Les modules command et shell

Ansible permet d'exécuter directement des commandes sur les hôtes.

Le module command :

C'est le module par défaut.

C'est une manière d'exécuter ponctuellement des commandes sur un groupe de machines.

Pour une exécution récurrente, les commandes seront exécutées via un Playbook.

```
# ansible spherius_servers -m command -a "/usr/bin/uptime"
ou
# ansible spherius_servers -a "/usr/bin/uptime"
deb_server | CHANGED | rc=0 >>
 15:33:45 up 0 min,  1 user,  load average: 0,58, 0,20, 0,07

CentOS6.5 | CHANGED | rc=0 >>
 15:33:28 up 2 min,  1 user,  load average: 0.52, 0.36, 0.14

CentOS7.1 | CHANGED | rc=0 >>
 15:33:33 up 2 min,  2 users, load average: 0,50, 0,40, 0,16
```

Pour obtenir la documentation sur le module command :

```
# ansible-doc command
```

Le module shell :

Le module **command** n'intègre pas les syntaxes spécifiques du shell comme les redirections, les pipes et le point virgule. Pour utiliser ces fonctionnalités, il faut appeler le module **shell**.

```
# ansible all -m shell -a "/bin/echo test ansible > /tmp/ans_test"
deb_server | CHANGED | rc=0 >>

CentOS6.5 | CHANGED | rc=0 >>

CentOS7.1 | CHANGED | rc=0 >>
```

Le module shell permet aussi d'utiliser les caractères spéciaux.

```
# ansible all -m shell -a "date ; cd /tmp ; touch fic ; ls fic*"
# ansible all -m shell -a "who | wc -l > /tmp/solution"
```

Pour obtenir la documentation sur le module shell :

```
# ansible-doc shell
```

Le transfert de fichiers

- Le module copy
- Le module file
- Utilise scp pour transférer les fichiers
- http://docs.ansible.com/ansible/latest/copy_module.html
- http://docs.ansible.com/ansible/latest/file_module.html

```
# ansible-doc copy
```

```
# ansible-doc file
```

Le module copy :

Le module **copy** permet de transférer des fichiers à plusieurs hôtes. Les fichiers sont copiés à l'identique sur la destination.

```
# ansible-doc copy
```

Le mot clef src indique le fichier source à copier.
Le mot clef dest indique la destination sur le poste client.

Copie du fichier /etc/passwd :

```
# ansible all -m copy -a "src=/etc/passwd dest=/tmp/password"
deb_server | CHANGED => {
  "changed": true,
  "checksum": "43ebe41a57d0dbe47727d6434b4a783a9bf3f67e",
  "dest": "/tmp/password",
  "gid": 0,
  "group": "root",
  "md5sum": "56a44958597ad6b61bc0c748f1d17d4b",
  "mode": "0644",
  "owner": "root",
  "size": 2157,
  "src": "/root/.ansible/tmp/ansible-tmp-1517930992.29-201300349940166/source",
  "state": "file",
  "uid": 0
}
CentOS6.5 | CHANGED => {
  "changed": true,
```

```

"checksum": "43ebe41a57d0dbe47727d6434b4a783a9bf3f67e",
"dest": "/tmp/password",
"gid": 0,
"group": "root",
"md5sum": "56a44958597ad6b61bc0c748f1d17d4b",
"mode": "0644",
"owner": "root",
"secontext": "unconfined_u:object_r:admin_home_t:s0",
"size": 2157,
"src": "/root/.ansible/tmp/ansible-tmp-1517930992.28-138131138374404/source",
"state": "file",
"uid": 0
}

```

Remarques :

- si "src=/tmp/rep dest=/tmp" copie du répertoire rep et de son contenu.
- si "src=/tmp/**rep/** dest=/tmp" copie que le contenu du répertoire rep.
- si "src=/tmp/rep dest=/tmp directory mode=777 group=1001" :
copie du répertoire rep et de son contenu, les répertoires auront comme droits 777 (pas les fichiers) et le groupe propriétaire est 1001.

Le module file :

Le module file permet de modifier le propriétaire et les permissions du fichier. Les mêmes options peuvent-être utilisées pour le module copy.

```
# ansible-doc file
```

Modification des droits et du propriétaire d'un fichier :

```

# ansible all -m file \
    -a "dest=/tmp/password mode=600 owner=theo group=users"
deb_server | CHANGED => (
  "changed": true,
  "gid": 100,
  "group": "users",
  "mode": "0600",
  "owner": "theo",
  "path": "/tmp/password",
  "size": 2157,
  "state": "file",
  "uid": 1000
)
CentOS6.5 | CHANGED => (
  "changed": true,
  "gid": 100,
  "group": "users",
  "mode": "0600",
  "owner": "theo",
  "path": "/tmp/password",
  "secontext": "unconfined_u:object_r:admin_home_t:s0",
  "size": 2157,
  "state": "file",
  "uid": 500
)

```

Copie d'un fichier en modifiant le propriétaire et les permissions :

```
# ansible spherius_servers -m copy \
  -a "src=/etc/passwd dest=/tmp/mypass mode=600 owner=theo group=users"
deb_server | CHANGED => {
  "changed": true,
  "checksum": "43ebe41a57d0dbe47727d6434b4a783a9bf3f67e",
  "dest": "/tmp/mypass",
  "gid": 100,
  "group": "users",
  "md5sum": "56a44958597ad6b61bc0c748f1d17d4b",
  "mode": "0600",
  "owner": "theo",
  "size": 2157,
  "src": "/root/.ansible/tmp/ansible-tmp-1517932488.9-256092188516732/source",
  "state": "file",
  "uid": 1000
}
...
```

Un code couleur permet de savoir si une action a été effectué ou non sur les machines clientes.

```
# ansible spherius_servers -m copy \
  -a "src=/etc/passwd dest=/tmp/mypass mode=600 owner=theo group=users"
deb_server | SUCCESS => {
  "changed": false,
  "checksum": "43ebe41a57d0dbe47727d6434b4a783a9bf3f67e",
  "dest": "/tmp/mypass",
  "gid": 100,
  "group": "users",
  "md5sum": "56a44958597ad6b61bc0c748f1d17d4b",
  "mode": "0600",
  "owner": "theo",
  "size": 2157,
  "src": "/root/.ansible/tmp/ansible-tmp-1517932488.9-256092188516732/source",
  "state": "file",
  "uid": 1000
}
...
```

Le module file peut aussi créer une structure arborescente comme **mkdir -p** à l'aide de l'option **state=directory**.

```
# ansible CentOS7.1 -m file -a "dest=/tmp/rep1/rep2/rep3 mode=755
owner=theo group=users state=directory"
CentOS7.1 | CHANGED => {
  "changed": true,
  "gid": 100,
  "group": "users",
  "mode": "0755",
  "owner": "theo",
  "path": "/tmp/rep1/rep2/rep3",
  "secontext": "unconfined_u:object_r:user_tmp_t:s0",
  "size": 0,
  "state": "directory",
  "uid": 1000
}
```

Le module file permet aussi de supprimer une arborescence de fichiers.

Suppression du répertoire /tmp/rep1/rep2 avec tout son contenu :

```
# ansible CentOS7.1 -m file -a "dest=/tmp/rep1/rep2 state=absent"
CentOS7.1 | CHANGED => {
  "changed": true,
  "path": "/tmp/rep1/rep2",
  "state": "absent"
}
```


Les commandes Ad-Hoc et les modules Ansible

La gestion des packages

- La gestion avec yum
 - La gestion avec apt
 - La gestion avec package
-
- http://docs.ansible.com/ansible/latest/yum_module.html
 - http://docs.ansible.com/ansible/latest/apt_module.html
 - http://docs.ansible.com/ansible/latest/package_module.html

```
# ansible-doc yum
# ansible-doc apt
# ansible-doc package
```

La gestion des packages

Vérifier qu'un package est présent. Ne pas le mettre à jour s'il est présent. L'installer s'il est absent.

```
# ansible centos_servers -m yum -a "name=nmap state=present"
```

```
CentOS7.1 | SUCCESS => {
  "ansible_facts": {
    "pkg_mgr": "yum"
  },
  "changed": false,
  "msg": "",
  "rc": 0,
  "results": [
    "2:nmap-6.40-7.el7.x86_64 providing nmap is already installed"
  ]
}
CentOS6.5 | CHANGED => {
  "ansible_facts": {
    "pkg_mgr": "yum"
  },
  "changed": true,
  "msg": "",
  "rc": 0,
  "results": [
    "Loaded plugins: fastestmirror, refresh-packagekit, security\nLoading mirror
speeds from cached hostfile\n * base: centos.mirror.fr.planethoster.net\n * extras:
centos.mirrors.ovh.net\n * updates: centos.mirror.fr.planethoster.net\nSetting up Install
Process\nResolving Dependencies\n--> Running transaction check\n--> Package nmap.x86_64
2:5.51-6.el6 will be installed\n--> Finished Dependency Resolution\n\nDependencies
Resolved\n\n.....
...
ansaction Test Succeeded\nRunning Transaction\n\nr Installing : 2:nmap-5.51-6.el6.x86_64
1/1 \n\nr Verifying : 2:nmap-5.51-6.el6.x86_64
1/1 \n\nInstalled:\n nmap.x86_64 2:5.51-6.el6
\n\nComplete!\n"
  ]
}
```

Pour mettre à jour vers une version spécifique du package si c'est possible :

```
# ansible centos_servers -m yum -a "name=nmap-6.40 state=present"
CentOS7.1 | SUCCESS => {
  "ansible_facts": {
    "pkg_mgr": "yum"
  },
  "changed": false,
  "msg": "",
  "rc": 0,
  "results": [
    "2:nmap-6.40-7.el7.x86_64 providing nmap-6.40 is already installed"
  ]
}
CentOS6.5 | FAILED! => {
  "ansible_facts": {
    "pkg_mgr": "yum"
  },
  "changed": false,
  "msg": "No package matching 'nmap-6.40' found available, installed or updated",
  "rc": 126,
  "results": [
    "No package matching 'nmap-6.40' found available, installed or updated"
  ]
}
```

Pour installer la dernière version disponible d'un package :

```
# ansible centos_servers -m yum -a "name=nmap state=latest"
CentOS6.5 | SUCCESS => {
  "ansible_facts": {
    "pkg_mgr": "yum"
  },
  "changed": false,
  "msg": "",
  "rc": 0,
  "results": [
    "All packages providing nmap are up to date",
    ""
  ]
}
CentOS7.1 | SUCCESS => {
  "ansible_facts": {
    "pkg_mgr": "yum"
  },
  "changed": false,
  "msg": "",
  "rc": 0,
  "results": [
    "All packages providing nmap are up to date",
    ""
  ]
}
```

Pour désinstaller un package :

```
# ansible centos_servers -m yum -a "name=ksh state=absent"
CentOS6.5 | SUCCESS => {
  "ansible_facts": {
    "pkg_mgr": "yum"
  },
  "changed": false,
  "msg": "",
  "rc": 0,
  "results": [
```

```
    "ksh is not installed"
  ]
}
CentOS7.1 | SUCCESS => {
  "ansible_facts": {
    "pkg_mgr": "yum"
  },
  "changed": false,
  "msg": "",
  "rc": 0,
  "results": [
    "ksh is not installed"
  ]
}
```

Pour lister les packages installés :

```
# ansible client1 -m yum -a "list=installed"
```

Pour savoir si un package donné est installé :

```
# ansible client1 -m yum -a "list=nmap"
```

- avec la variable yumstate à 'available' : il s'agit d'un package au sein d'un repository, donc pouvant être installé.
- avec la variable yumstate à 'installed' : il s'agit d'un package installé sur le poste client.

Le module apt

Sur les serveurs à base de Débian, il suffit d'utiliser le module **apt**. Les différentes options sont identiques entre le module yum et apt.

```
# ansible deb_server -m apt -a "name=ksh state=present"
```

Le module package

Un module générique existe mais il supporte moins d'options que les modules spécifiques.

```
# ansible all -m package -a "name=ksh state=present"
```

Pour plus d'informations sur les options supportées par un module :

```
# ansible-doc apt
# ansible-doc yum
# ansible-doc package
```

Les commandes Ad-Hoc et les modules Ansible

La gestion des utilisateurs

- Le module user le module group
- http://docs.ansible.com/ansible/latest/user_module.html
- http://docs.ansible.com/ansible/latest/group_module.html

```
# ansible-doc user
```

```
# ansible-doc group
```

La gestion des utilisateurs

Le module user gère les comptes utilisateurs.

Création d'un utilisateur avec quelques options :

```
# ansible spherius_servers -m user -a "name=user1 password='\6\$\21kkZVVna
934bICwoojtJ6oIZ3IcmVffOnyd2FxEaUW8JP8wvYLEy1OXR5DRHzMcCv4rRI.NIKG5K1tUAY
n.N51/dJ9V.a1' comment='compte utilisateur' uid=1111 group=users"
deb_server | CHANGED => {
  "append": false,
  "changed": true,
  "comment": "compte utilisateur",
  "group": 100,
  "home": "/home/user1",
  "move_home": false,
  "name": "user1",
  "password": "NOT_LOGGING_PASSWORD",
  "shell": "/bin/bash",
  "state": "present",
  "uid": 1111
}
CentOS7.1 | CHANGED => {
  "append": false,
  "changed": true,
  "comment": "compte utilisateur",
  "group": 100,
  "home": "/home/user1",
  "move_home": false,
  "name": "user1",
  "password": "NOT_LOGGING_PASSWORD",
  "shell": "/bin/bash",
  "state": "present",
  "uid": 1111
}
```

Création d'un utilisateur sans option :

```
# ansible spherius_servers -m user -a "name=user1 state=present"
```

Modification d'un compte existant :

```
# ansible all -m user -a "name=eve uid=1111 group=2222"
```

Le groupe avec le 2222 doit exister au préalable. Le compte est créé sur les postes n'ayant pas cet utilisateur, sinon le compte est modifié. Le répertoire de connexion, ainsi que son contenu est affecté à l'uid=1111 et au gid=2222.

Suppression d'un compte utilisateur, sans suppression du répertoire de connexion :

```
# ansible client1 -m user -a "name=user1 state=absent"
```

Suppression d'un compte utilisateur, avec suppression du répertoire de connexion :

```
# ansible client1 -m user -a "name=user1 state=absent remove=yes"
```

Le module group gère les groupes utilisateurs.

Création d'un groupe (remarque 'state=present' est la valeur par défaut) :

```
# ansible spherius_servers -m group -a "name=compta gid=2000"
deb_server | CHANGED => {
  "changed": true,
  "gid": 2000,
  "name": "compta",
  "state": "present",
  "system": false
}
CentOS6.5 | CHANGED => {
  "changed": true,
  "gid": 2000,
  "name": "compta",
  "state": "present",
  "system": false
}
```

L'aide

Pour le détail des arguments utilisables :

```
# ansible-doc user
# ansible-doc group
```

Les commandes Ad-Hoc et les modules Ansible

La gestion des services

- Le module service
- http://docs.ansible.com/ansible/latest/service_module.html

```
# ansible-doc service
```

La gestion des services

Le module service gère la gestion d'un service : start, stop, restart, etc.

Pour le détail des arguments utilisables :

```
# ansible-doc service
```

Démarrer le service httpd sur les serveurs CentOS :

```
# ansible centos_servers -m service -a "name=httpd state=started"
CentOS6.5 | SUCCESS => {
  "changed": false,
  ...
}
CentOS7.1 | SUCCESS => {
  "changed": false,
  "name": "httpd",
  "state": "started",
  "status": {
    "ActiveEnterTimestamp": "mer. 2018-02-07 09:52:52 CET",
    ...
  }
}
```

Démarrer le service apache sur le serveur Débian :

```
# ansible deb_server -m service -a "name=apache2 state=started"
deb_server | SUCCESS => {
  "changed": false,
  "name": "apache2",
  "state": "started",
  "status": {
    "ActiveEnterTimestamp": "mer. 2018-02-07 09:58:05 CET",
    ...
  }
}
```

Redémarrer un service.

```
# ansible CentOS6.5 -m service -a "name=httpd state=restarted"
CentOS6.5 | CHANGED => {
  "changed": true,
  "name": "httpd",
  "state": "started"
...
}
```

Arrêter un service.

```
# ansible centos_servers -m service -a "name=httpd state=stopped"
CentOS6.5 | CHANGED => {
  "changed": true,
  ...
}
CentOS7.1 | CHANGED => {
  "changed": true,
  "name": "httpd",
  "state": "stopped",
  "status": {
  ...
}
```

Les commandes Ad-Hoc et les modules Ansible

Le module setup

- Le module `setup` Liste des variables d'un hôte.
- http://docs.ansible.com/ansible/latest/modules/setup_module.html

```
# ansible-doc setup
```

Le module setup

Le module **setup** permet de récupérer des informations d'un hôte (sous forme de variables). Cela peut être des données sur les caractéristiques matérielles (type de processeurs, sur la mémoire, la swap, les disques et leur partitionnement, le détail des lvm, des cartes réseaux, etc), des variables systèmes (nom de la machine, etc) ou autres. Ces variables sont exploitables au sein de différents playbooks, rôles et templates d'Ansible. Ces variables sont appelées des « facts ».

Pour le détail des options utilisables :

```
# ansible-doc setup
```

Exemple de quelques variables :

```
# ansible client1 -m setup

client1 | SUCCESS => {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "192.168.122.1",
      "192.168.0.30"
    ],
    "ansible_date_time": {
      "date": "2018-04-22",
      "day": "22",
      "hour": "16",
      "tz": "CEST",
      ...
    },
  },
}
```



```

"ansible_default_ipv4": {
  "address": "192.168.0.30",
  "alias": "enp0s3",
  "gateway": "192.168.0.254",
  "interface": "enp0s3",
  "macaddress": "08:00:27:ad:c7:72",
  "netmask": "255.255.255.0",
  "network": "192.168.0.0",
  "type": "ether"
},

"ansible_devices": {
  "sda": {
    ... informations sur le disque et son partitionnement
  },
  ...
},
"ansible_env": {
  ... informations sur les variables systèmes du hôte
  "HOME": "/root",
  "HOSTNAME": "mars",
},
"ansible_hostname": "mars",
"ansible_os_family": "RedHat",
"ansible_pkg_mgr": "yum",
"ansible_user_id": "root",
...
},
"changed": false
}

```

Création d'un répertoire /tmp/facts avec des fichiers portant le nom de chaque hôte. Chaque fichier contient le résultat du « setup » de son hôte.

```

# ansible all -m setup --tree /tmp/facts
# cat /tmp/facts/client1 | tr ',' '\n'      visualisation du fichier

```

Pour filtrer sur quelques variables, l'utilisation de caractères spéciaux est préconisée.

```

# ansible all -m setup -a 'filter=*eth[0-2]*'
# ansible all -m setup -a 'filter=*mb*'
client1 | SUCCESS => {
  "ansible_facts": {
    "ansible_memfree_mb": 238,
    "ansible_memory_mb": {
      "nocache": {
        "free": 975,
        "used": 864
      },
      "real": {
        "free": 238,
        "total": 1839,
        "used": 1601
      },
      "swap": {
        "cached": 0,
        "free": 2047,
        "total": 2047,
        "used": 0
      }
    },
    "ansible_memtotal_mb": 1839,
    "ansible_swapfree_mb": 2047,
    "ansible_swaptotal_mb": 2047
  },
  "changed": false
}

```

Notes

Les playbooks

Dans ce chapitre, nous allons étudier la création et le fonctionnement des playbooks.

Les playbooks

- Description d'un playbook
 - Les variables et les tableaux
 - La priorité et la portée des variables
 - Les templates
 - La boucle for
 - Les Handlers
 - Le module debug et le mot clef register
- Les boucles
 - La condition when
 - Les filtres
 - Les opérations arithmétiques

Les playbooks

Description d'un playbook

- <https://github.com/ansible/ansible-examples>
- Le langage yaml
- Exécution et débogage

```
# ansible-playbook --syntax-check playbook_exemple.yml
# ansible-playbook --check playbook_exemple.yml
# ansible-playbook playbook_exemple.yml
```

Description d'un playbook

Un playbook permet d'orchestrer l'ensemble des actions à effectuer sur un parc de machines en tenant compte de contraintes (ordre de démarrage, etc). Des exemples de playbooks sont consultables sur le site suivant :

<https://github.com/ansible/ansible-examples>

Les playbooks sont au format YAML qui a une syntaxe qui est rapidement assimilable. Chaque playbook est constitué d'un ou plusieurs plays.

Un play pourrait être traduit par tâche et playbook par liste de tâches. Une tâche Ansible est basiquement un appel à un module Ansible.

En composant son propre playbook il est possible de contrôler le déploiement de plusieurs machines et de contrôler les opérations à effectuer dessus.

La syntaxe d'un playbook reste relativement simple. Il faut définir les hôtes, les variables et indiquer les tâches à effectuer. Chaque tâche a un nom et appelle des modules. Le nom des modules est identique que ceux en ligne de commandes.

L'option **--syntax-check** de la commande `playbook` permet de vérifier la syntaxe. L'option **--check** permet de simuler l'action sans l'appliquer réellement.

Les documents écrits en YAML commencent par trois tirets (---), ils peuvent se terminer par trois points (...) mais cela n'est pas obligatoire. L'indentation est obligatoire dans le fichier.

Un exemple de base :

```
# more exemple1.yml
---
- name: Premier exemple de base
  hosts: centos_servers
  tasks:
    - name: Execution du module ping
      ping:
    - name: Execution de la commande date
      command:
        date
    - name: copie d'un fichier
      copy:
        src: httpd.conf
        dest: /etc/httpd/conf/httpd.conf
    - name: redemarrage du service httpd
      service:
        name: httpd
        state: restarted
...
```

```
[root@server ~]# ansible-playbook exemple1.yml

PLAY [Premier exemple de base] *****

TASK [Gathering Facts] *****
ok: [centos2]
ok: [centos1]

TASK [Execution du module ping] *****
ok: [centos2]
ok: [centos1]

TASK [Execution de la commande date] *****
changed: [centos1]
changed: [centos2]

TASK [copie d'un fichier] *****
changed: [centos2]
changed: [centos1]

TASK [redemarrage du service httpd] *****
changed: [centos1]
changed: [centos2]

PLAY RECAP *****
centos1           : ok=5    changed=3    unreachable=0    failed=0
centos2           : ok=5    changed=3    unreachable=0    failed=0
```

Un autre exemple :

```
# cat connection.yml
---
- name: Test de la copie d'un fichier
  hosts: localhost, client1, client2
  tasks:
    - name: Copie d un fichier vers /rep
      copy:
        src: /etc/passwd
        dest: /rep/fic10
...
# ansible-playbook connection.yml

PLAY [Test de la copie d'un fichier] *****

TASK [Gathering Facts] *****
ok: [localhost]
ok: [client1]
fatal: [client2]: UNREACHABLE! => {"changed": false, "msg": "Failed to connect to the
host via ssh: ssh: connect to host client2 port 22: No route to host\r\n", "unreachable":
true}

TASK [Copie d un fichier vers /rep] *****
changed: [localhost]
fatal: [client1]: FAILED! => {"changed": false, "checksum":
"358234378830070365cd637c0191ffe8899c99bd", "msg": "Destination directory /rep does not
exist"}
    to retry, use: --limit @/root/Ansible_Playbooks/connection.retry

PLAY RECAP *****
client1                : ok=1    changed=0    unreachable=0    failed=1
client2                : ok=0    changed=0    unreachable=1    failed=0
localhost              : ok=2    changed=1    unreachable=0    failed=0
```

Pour localhost : la connexion a fonctionné, les facts ont été récupéré et le fichier a été copié.

Pour client1: la connexion a fonctionné, les facts ont été récupéré. Mais le fichier n'a pas été copié, une erreur sur la task est apparue (la raison : absence du répertoire /the/suite sur client1).

Pour client2 : la connexion a échoué. On constate qu'il n'y a pas par la suite de tentative d'exécution de tasks sur ce hôte.

La commande a généré un fichier du nom du playbook avec l'extension .retry contenant la liste des machines sur lesquelles une erreur s'est produite.

```
# more connection.retry
client1
```

Il est possible de paramétrer ce comportement en modifiant la variable adéquate dans le fichier de configuration d'Ansible.

```
# grep retry_files_save_path /etc/ansible/ansible.cfg
retry_files_save_path=/tmp/.ansible-retry
```

Le fichier créé sera /tmp/.ansible-retry au lieu de celui par défaut.

En relançant le playbook, on obtient :

```
# ansible-playbook connection.yml

. . .

TASK [Copie d un fichier vers /rep] *****
fatal: [client1]: FAILED! => {"changed": false, "checksum":
"358234378830070365cd637c0191ffe8899c99bd", "msg": "Destination directory /rep does not
exist"}
ok: [localhost]
    to retry, use: --limit @/root/Ansible_Playbooks/connection.retry

PLAY RECAP *****
client1          : ok=1    changed=0    unreachable=0    failed=1
client2          : ok=0    changed=0    unreachable=1    failed=0
localhost        : ok=2    changed=0    unreachable=0    failed=0
```

Pas de changement sur le hôte localhost car le fichier étant présent la copie ne s'est pas faite.

Exemple d'un fichier playbook:

```
# cat  playbook.yml
---
- name: Playbook Exemple
  hosts: centos_servers
  remote_user: root
  vars:
    bind_port: 53
    domain: mondomaine.lan
  tasks:
    - name: Installation des outils de developpement
      yum:
        name: "@Development Tools"
        state: present
    - name: Installation du serveur DNS
      yum:
        name: bind,bind-utils
        state: present
    - name: copie du fichier de configuration named.conf
      copy:
        src: dns/named_source.conf
        dest: /etc/named.conf
    - name: copie du fichier de configuration named.rfc1912.zones
      copy:
        src: dns/rfc_source.zones
        dest: /etc/named.rfc1912.zones
    - name: copie du fichier de configuration de la zone
      copy:
        src: dns/zone_source.conf
        dest: /var/named/{{domain}}
    - name: Redemarrage du serveur DNS
      service:
        name: named
        state: restarted
...
```

Vérification des erreurs dans le playbook:

option: --syntax-check

Permet la vérification de la syntaxe du fichier playbook.

```
# ansible-playbook  --syntax-check  playbook.yml

playbook: playbook.yml
```

vérification syntaxique YAML : yamllint

```
# yamllint  playbook.yml
```

vérification syntaxique et logique Ansible : ansible-lint

```
# ansible-lint  playbook.yml
```

Simulation de l'application du playbook: --check

Permet de simuler les actions sans les appliquer réellement.

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

PLAY [Playbook Exemple] *****

TASK [Gathering Facts] *****
ok: [centos1]
ok: [centos2]

TASK [Installation des outils de developpement] *****
ok: [centos1]
ok: [centos2]

TASK [Installation du serveur DNS] *****
changed: [centos1]
changed: [centos2]

TASK [copie du fichier de configuration named.conf] *****
changed: [centos2]
changed: [centos1]

TASK [copie du fichier de configuration named.rfc1912.zones] *****
changed: [centos1]
changed: [centos2]

TASK [copie du fichier de configuration de la zone] *****
changed: [centos1]
changed: [centos2]

TASK [Redemarrage du serveur DNS] *****
changed: [centos1]
changed: [centos2]

PLAY RECAP *****
centos1      : ok=7    changed=5    unreachable=0    failed=0
centos2      : ok=7    changed=5    unreachable=0    failed=0
```

Exécution du playbook :

```
# ansible-playbook  playbook.yml

PLAY [Playbook Exemple] *****

TASK [Gathering Facts] *****
ok: [centos1]
ok: [centos2]

TASK [Installation des outils de developpement] *****
ok: [centos1]
ok: [centos2]

TASK [Installation du serveur DNS] *****
changed: [centos1]
changed: [centos2]

TASK [copie du fichier de configuration named.conf] *****
changed: [centos2]
changed: [centos1]

TASK [copie du fichier de configuration named.rfc1912.zones] *****
changed: [centos1]
changed: [centos2]

TASK [copie du fichier de configuration de la zone] *****
changed: [centos1]
changed: [centos2]

TASK [Redemarrage du serveur DNS] *****
changed: [centos1]
changed: [centos2]

PLAY RECAP *****
centos1      : ok=7    changed=5    unreachable=0    failed=0
centos2      : ok=7    changed=5    unreachable=0    failed=0
```

Un code couleur permet de savoir rapidement et facilement ce qui a été appliqué sur nos différentes machines.

La reexécution du playbook est plus rapide car les packages et les fichiers sont déjà présents.

```
# ansible-playbook playbook.yml

PLAY [Playbook Exemple] *****

TASK [Gathering Facts] *****
ok: [centos1]
ok: [centos2]

TASK [Installation des outils de developpement] *****
ok: [centos1]
ok: [centos2]

TASK [Installation du serveur DNS] *****
ok: [centos1]
ok: [centos2]

TASK [copie du fichier de configuration named.conf] *****
ok: [centos2]
ok: [centos1]

TASK [copie du fichier de configuration named.rfc1912.zones] *****
ok: [centos1]
ok: [centos2]

TASK [copie du fichier de configuration de la zone] *****
ok: [centos1]
ok: [centos2]

TASK [Redemarrage du serveur DNS] *****
changed: [centos1]
changed: [centos2]

PLAY RECAP *****
centos1      : ok=7    changed=1    unreachable=0    failed=0
centos2      : ok=7    changed=1    unreachable=0    failed=0
```

Les playbooks

Les variables et les tableaux

- Règles de nommage des variables
- Les variables de l'inventaire - Les variables internes Ansible - Les variables déclarées
- Utilisation de variables : `{{ ansible_all_ipv4_addresses[0] }}`

```
- hosts: spherius_servers
  vars:
    appli_path: "{{ ip_addr }}/22"
```

```
# more fichier_variables
variable1: valeur_1
equipe: [ {nom: jean, uid: 1001, gid: 2020}, {nom: marc, uid: 1002, gid: 2020} ]
```

```
- hosts: spherius_servers
  var_files:
    - fichier_variables
```

Les variables et les tableaux

Les variables dans Ansible sont constituées de lettres, de chiffres et d'underscores. Les variables doivent toujours commencer par une lettre.

Les noms suivants sont des noms de variables valides : var, var_10, var_ip

Les noms suivants sont noms de variables invalides : var-1, 10_var, var.ip

La syntaxe pour définir les variables :

```
vars:
  variable1: valeur_1
  variable2: valeur_2
```

La déclaration de tableaux :

```
tab:
  - 1
  - "deux"
  - "valeur trois"
est équivalent à tab: [ 1, "deux", "valeur trois" ]
```

Une déclaration de variables plus complexe (table de hachage) :

```
equipe:
  - nom: jean
    uid: 1001
    gid: 2020
  - nom: marc
    uid: 1002
    gid: 2020
est équivalent à
equipe:
  - { nom: jean, uid: 1001, gid: 2020 }
  - { nom: marc, uid: 1002, gid: 2020 }
est équivalent à
equipe: [ {nom: jean, uid: 1001, gid: 2020}, {nom: marc, uid: 1002, gid: 2020} ]
```

Un autre exemple:

```
ami1:
  nom: dupond
  prenom: jean
  adresse:
    rue: "1 chemin de la paix"
    code: 75000
    ville: Paris
```

Les variables de l'inventaire

Les variables de l'inventaire sont constituées des variables du fichier `/etc/ansible/hosts` et du fichier `/etc/ansible/ansible.cfg`. Le fichier `hosts` contient des variables qui référencent les noms des hôtes par machine, par groupe de machines ou par groupe de groupes de machines.

```
# more /etc/ansible/hosts
postel
poste2

[deb_servers]                # groupe
deb_server1  http_port=80 https_port=443
deb_server2

[centos_servers]
CentOS6.5
CentOS7.1

[centos_servers:vars]        # les variables du groupe centos_servers
ntp_server=0.fr.pool.ntp.org

[servers:children]           # groupe de groupes
deb_servers
centos_servers
```

La documentation complète :

http://docs.ansible.com/ansible/latest/intro_inventory.html

Les variables personnalisées

Les variables sont définissables directement dans un playbook :

```
- hosts: spherius_servers
  vars:
    http_port: 80
```

ou :

```
- hosts: spherius_servers
  vars: http_port=80
```

Les variables récupérées par le module setup sont également utilisables :

```
# ansible CentOS7.1 -m setup | more
CentOS7.1 | SUCCESS => {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "192.168.122.1",
      "192.168.1.14"
    ],
    "ansible_all_ipv6_addresses": [
      "fe80::a00:27ff:fe76:1606"
    ],
    "ansible_architecture": "x86_64",
    "ansible_bios_date": "12/01/2006",
    ...
  }
```

Utilisation des variables

L'utilisation de ces variables respecte la syntaxe suivante:

- pour la valeur d'une variable :

```
{{ ansible_bios_version }}          {{ ansible_architecture }}
```

- pour la valeur d'une propriété d'une variable :

```
{{ ansible_date_time.epoch }}       {{ ansible_date_time.month }}
```

- pour la valeur d'une variable indexée dans un tableau. L'index commence à 0 pour la première valeur du tableau. Un tableau est entouré de crochets ([.....]):

```
{{ ansible_all_ipv4_addresses[0] }}
{{ ansible_all_ipv4_addresses[1] }}
```

Attention : La syntaxe YAML nécessite que si vous commencer une valeur avec {{ var }}, il faut entourer toute la ligne de quotes.

Exemple de mauvaise syntaxe:

```
- hosts: spherius_servers
  vars:
    appli_path: {{ ip_addr }}/24
```

Exemple de bonne syntaxe:

```
- hosts: spherius_servers
  vars:
    appli_path: "{{ ip_addr }}/24"
```

Un fichier de variables:

```
# more fichier_variables
variable1: valeur_1
variable2: valeur_2
equipe: [ {nom: jean, uid: 1001, gid: 2020}, {nom: marc, uid: 1002, gid: 2020} ]
```

Utilisation dans un playbook:

```
- hosts: spherius_servers
  var_files:
    - fichier_variables
```

Voici un exemple de mise en œuvre pour un playbook:

```
# cat variables_base.yml
---
- name: Manipulation de variables Ansible
  hosts: all
  vars:
    ip_addr: "{{ ansible_all_ipv4_addresses[0] }}"
    appli_path: "{{ ip_addr }}/22"
  tasks:
    - name: Affichage de la variable appli_path
      debug:
        msg: "La variable appli_path = {{ appli_path }}"
...
```

```
# ansible-playbook variables_base.yml
PLAY [Manipulation de variables Ansible] *****

TASK [Gathering Facts] *****
ok: [debian1]
ok: [centos1]
ok: [centos2]

TASK [Affichage de la variable appli_path] *****
ok: [centos1] => {
  "msg": "La variable appli_path = 192.168.1.49/22"
}
ok: [centos2] => {
  "msg": "La variable appli_path = 192.168.1.88/22"
}
ok: [debian1] => {
  "msg": "La variable appli_path = 192.168.1.69/22"
}

PLAY RECAP *****
centos1           : ok=2    changed=0    unreachable=0    failed=0
centos2           : ok=2    changed=0    unreachable=0    failed=0
debian1          : ok=2    changed=0    unreachable=0    failed=0
```

Un exemple utilisant un tableau :

```
# cat variables_tableau.yml
---
- name: Les variables Ansible dans un tableau
  hosts: localhost
  vars:
    equipe: [{nom: jean, uid: 1001}, {nom: marc, uid: 1002}]
  tasks:
    - name: Affichage des variables à l'index 1 de mon tableau
      debug:
        msg: "Bonjour {{equipe[1].nom}}, ton uid est {{equipe[1].uid}}"
...
```

```
# ansible-playbook variables_tableau.yml
PLAY [Les variables Ansible dans un tableau] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Affichage des variables à l'index 1 de mon tableau] *****
ok: [localhost] => {
  "msg": "Bonjour marc, ton uid est 1002"
}

PLAY RECAP *****
localhost        : ok=2    changed=0    unreachable=0    failed=0
```


Les playbooks

La priorité et la portée des variables

role defaults	role (et include_role) params
inventory file ou script group vars	include params
inventory group_vars/all	include_vars
playbook group_vars/all	set_facts / registered vars
inventory group_vars/*	extra vars (toujours « gagnante »)
playbook group_vars/*	
inventory file ou script host vars	
inventory host_vars/*	
playbook host_vars/*	
host facts	
play vars	
play vars_prompt	
play vars_files	
role vars (définies dans role/vars/main.yml)	
block vars (seulement pour une tasks dans un bloc)	
task vars (seulement pour la task)	

La priorité et la portée des variables

Ordre de priorité d'une variable

Les variables ont une priorité en fonction de l'endroit où elles sont déclarées.

Ordre de priorité des variables dans ansible 2.x de la moins prioritaire à la plus prioritaire :

- role defaults
- inventory file ou script group vars
- inventory group_vars/all
- playbook group_vars/all
- inventory group_vars/*
- playbook group_vars/*
- inventory file ou script host vars
- inventory host_vars/*
- playbook host_vars/*
- host facts
- play vars
- play vars_prompt
- play vars_files
- role vars (définies dans role/vars/main.yml)
- block vars (seulement pour une tasks dans un bloc)
- task vars (seulement pour la task)
- role (et include_role) params
- include params
- include_vars
- set_facts / registered vars
- extra vars (toujours « gagnante »)

De manière basique, les variables définies dans le rôle par défaut sont le plus facilement écrasées. Chaque variable définie dans le répertoire vars du rôle écrase les versions précédentes de la variable définie dans l'espace de noms. L'idée étant que plus la variable est déclarée explicitement, plus elle est prioritaire. C'est pour cela que les variables déclarées en ligne de commande avec l'option -e sont toujours les « gagnantes ».

Les répertoires group_vars et host_vars peuvent-être créés au niveau du fichier d'inventaire ou au niveau du playbook. Ils contiennent des fichiers portant respectivement le nom des groupes et des hôtes avec leur variables spécifiques.

Exemple:

Pour un site (tous les hôtes), les variables peuvent-être déclarées dans le répertoire group_vars/all

```
# more /etc/ansible/group_vars/all
---
ntp_server: 0.fr.ntp.pool.org
...
```

Pour une région (un groupe de hôtes) , les variables peuvent-être déclarées dans le fichier group_vars/nom_du_groupe_de_hotes. La valeur de ntp_server va écraser la valeur définie au niveau du site. Le fichier ci-dessous concerne le groupe de hôtes « paris ».

```
# more /etc/ansible/group_vars/paris
---
ntp_server: paris.ntp.pool.org
...
```

Si pour une raison quelconque, il faut indiquer un serveur ntp spécifique pour un hôte, la valeur de la variable au niveau du groupe sera écrasée par celle de l'hôte. Le fichier ci-dessous concerne la machine « mail.paris.mydomain.lan » du groupe de hôtes « paris ».

```
# more /etc/ansible/host_vars/mail.paris.mydomain.lan
---
ntp_server: interne.ntp.mydomain.lan
...
```

Lors de la création des rôles avec des valeurs par défaut classiques, indiquez les dans le fichier **roles/nom_du_role/defaults/main.yml**. Cela permet d'avoir la valeur par défaut pour les variables mais elles sont écrasées par n'importe quel paramétrage spécifique dans Ansible.

Un développement pour mettre en évidence certaines priorités

```
# tree rep_vars
├── hosts -----> Le fichier d'inventaire
├── group_vars
│   ├── all
│   └── societe
├── host_vars
│   └── client1
└── Playbook
    ├── variables.yml -----> Le fichier playbook
    ├── autre_var.yml
    ├── group_vars
    │   ├── all
    │   └── host_vars
    │       └── client1
```

Le fichier d'inventaire.

```
# cat hosts
[societe]
client1
client2
[all:vars]
prenom=Jean
```

Le playbook.

```
# cat Playbook/variables.yml
---
- hosts: all
  tasks:
    - name: Affichage de la valeur de prenom
      debug:
        msg: "La variable prenom contient {{ prenom }}"
...
```

La commande ci-dessous est exécutée à chaque fois. Les fichiers sont créés ou modifiés au fur à mesure.

```
# ansible-playbook -i hosts Playbook/variables.yml

Pour client1 et client2:  "msg": "La variable prenom contient Jean"

# cat group_vars/all                                     group_vars/all inventaire
---
prenom: Marc
...

Pour client1 et client2:  "msg": "La variable prenom contient Marc"

# cat Playbook/group_vars/all                             group_vars/all playbook
---
prenom: Theo
...

Pour client1 et client2:  "msg": "La variable prenom contient Theo"

# cat group_vars/societe                                 group_vars/societe inventaire
---
prenom: Eve
...

Pour client1 et client2:  "msg": "La variable prenom contient Eve"
```

```
# cat hosts fichier inventaire sur un host
[societe]
client1      prenom=Celine
client2
[all:vars]
prenom=Jean

Pour client1:  "msg": "La variable prenom contient Celine"          client2 toujours Eve

# cat host_vars/client1 host_vars/nom_hote inventaire
---
prenom: Valerie
...

Pour client1:  "msg": "La variable prenom contient Valerie"        client2 toujours Eve

# cat Playbook/host_vars/client1 host_vars/nom_hote playbook
---
prenom: Antoine
...

Pour client1:  "msg": "La variable prenom contient Antoine"        client2 toujours Eve

# cat Playbook/variables.yml play vars
---
- hosts: all
  vars:
    prenom: Noelle
  tasks:
    - name: Affichage de la valeur de prenom
      debug:
        msg: "La variable prenom contient {{ prenom }}"
...

Pour client1 et client2:  "msg": "La variable prenom contient Noelle"

# cat variables.yml # cat autre_var.yml include_vars
---
- hosts: all
  vars:
    prenom: Noelle
  tasks:
    - name: inclusion d'un fichier de variables
      include_vars: autre_var.yml
    - name: Affichage de la valeur de prenom
      debug:
        msg: "La variable prenom contient {{ prenom }}"
...

Pour client1 et client2:  "msg": "La variable prenom contient Olivier"
```

L'option -e est toujours gagnante.

```
# ansible-playbook -i hosts Playbook/variables.yml -e prenom=Ansible

Pour client1 et client2:  "msg": "La variable prenom contient Ansible"
```

Autre exemple :

```
# cat inventaire.inv
poste[1:2]

[all:vars]
ansible_user=root

[deb_servers]
deb_server[1:2]

[domaine1]
apache1      apache_url=intra.domaine http_port=80 https_port=443
mysql1
centos_6.5   ansible_user=user1
centos_7.1

[linux:children]
domaine1
deb_servers

[linux:vars]
ntp_server=0.fr.pool.ntp.org

[windows]
serveur1
basededonnee1

[windows:vars]
ansible_connection=winrm
ansible_user=Administrator
```

Afficher la valeur de la variable `ansible_user`.

```
# ansible all -i inventaire.inv -m debug -a "var=ansible_user" --one-line
poste1 | SUCCESS => { "ansible_user": "root", "changed": false}
serveur1 | SUCCESS => { "ansible_user": "Administrator", "changed": false}
poste2 | SUCCESS => { "ansible_user": "root", "changed": false}
apache1 | SUCCESS => { "ansible_user": "root", "changed": false}
basededonnee1 | SUCCESS => { "ansible_user": "Administrator", "changed": false}
mysql1 | SUCCESS => { "ansible_user": "root", "changed": false}
centos_7.1 | SUCCESS => { "ansible_user": "root", "changed": false}
centos_6.5 | SUCCESS => { "ansible_user": "user1", "changed": false}
deb_server1 | SUCCESS => { "ansible_user": "root", "changed": false}
deb_server2 | SUCCESS => { "ansible_user": "root", "changed": false}
```

Afficher la valeur de plusieurs variables . L'affichage a été modifié pour la rendre plus lisible.

```
# ansible all -i inventaire.inv -m debug \
-a "var=ansible_connection,ansible_user,ntp_server,apache_url" --one-line

machine      ansible_connection  ansible_user      ntp_server      apache_url
poste1       'ssh'               'root'            Undefined       Undefined
poste2       'ssh'               'root'            Undefined       Undefined
basededonnee1 'winrm'             'Administrator'   Undefined       Undefined
serveur1     'winrm'             'Administrator'   Undefined       Undefined
mysql1       'ssh'               'root'            '0.fr.pool.ntp.org' Undefined
apache1      'ssh'               'root'            '0.fr.pool.ntp.org' 'intra.domaine'
centos_6.5   'ssh'               'user1'           '0.fr.pool.ntp.org' Undefined
deb_server1  'ssh'               'root'            '0.fr.pool.ntp.org' Undefined
centos_7.1   'ssh'               'root'            '0.fr.pool.ntp.org' Undefined
deb_server2  'ssh'               'root'            '0.fr.pool.ntp.org' Undefined
```

Les playbooks

Les templates

- Fichier paramétrable Utilise les variables Ansible
- Exemple de fichier Template

```
# cat exemple_template.j2
# Ceci est un fichier de parametrage

user={{ mavar1 }}
repLog=/opt/appli/{{ ansible_distribution }}/log

l'adresse du poste est {{ mavar2 }}
ou encore {{ ansible_all_ipv4_addresses[0] }}
```

- Le module template

```
- name: copie du template
  template:
    src: /root/Ansible/templates/exemple_template.template
    dest: /opt/appli/etc/appli.conf
```

Les templates

Un fichier template permet de customiser un fichier en exploitant le résultat des variables Ansible. Ainsi, le contenu du fichier transmis au poste client est personnalisé en fonction de spécificités du poste client. Le module à utiliser est : template.

Exemple:

Le template:

```
# cat exemple_template.j2
# Ceci est un fichier de parametrage

user={{ mavar1 }}
repLog=/opt/appli/{{ ansible_distribution }}/log

l'adresse du poste est {{ mavar2 }}
ou encore {{ ansible_all_ipv4_addresses[0] }}
```

Le playbook:

```
# cat exemple_playbook_template.yml
---
- name: Utilisation d'un template
  hosts: client1
  vars:
    mavar1: Paul
    mavar2: "{{ ansible_all_ipv4_addresses[0] }}"
  tasks:
    - name: copie du template
      template:
        src: exemple_template.j2
        dest: /opt/appli/etc/appli.conf
...
```

Après exécution du playbook, sur le poste CLIENT:

```
Poste_CLIENT# cat /opt/appli/etc/appli.conf
# Ceci est un fichier de parametrage

user=Paul
repLog=/opt/appli/CentOS/log

l'adresse du poste est 192.168.0.20
ou encore 192.168.0.20
```

Autre exemple:

Extrait d'un fichier template pour un serveur apache:

```
# grep '{{' /ansible/template/apache/httpd.conf
Listen {{ http_port }}
ServerAdmin root@{{ domain }}
DocumentRoot "/var/www/{{ domain }}"
<Directory "/var/www/{{ domain }}">
```

Exemple de playbook utilisant un template:

```
# cat playbook_exemple_template.yml
---
- name: Configuration sur serveur Apache
  hosts: centos_servers
  vars:
    http_port: 80
    domain: mondomaine.lan
  tasks:
    - name: Installer apache
      yum:
        name: httpd
        state: present
    - name: copie du fichier template d'apache
      template:
        src: httpd.conf
        dest: /etc/httpd/conf/httpd.conf
    - name: Redemarrage du serveur Apache
      service:
        name: httpd
        state: restarted
...
```

Résultat sur un poste de centos_servers:

```
# more /etc/httpd/conf/httpd.conf
...
Listen 80
ServerAdmin root@mondomaine.lan
DocumentRoot "/var/www/mondomaine.lan"
<Directory "/var/www/mondomaine.lan">
...
```

Les playbooks

La boucle for

```
- hosts: all
vars:
  liste: [ "Jean", "Marc", "Theo" ]
tasks:
  - name: "test de la boucle for"
    template:
      src: boucle_for.j2
      dest: /tmp/{{inventory_hostname}}.res
    connection: local
```

Le template boucle_for.j2

```
Traitement sur le poste {{inventory_hostname}}
Une liste de personne :
  {% for personne in liste %}
    Nom : {{personne}}
  {% endfor %}
La liste des interfaces est :
  {% for element in ansible_interfaces %}
    interface : {{element}}
  {% endfor %}
```

La boucle for

La boucle for est utilisée pour les templates. La syntaxe est :

```
{% for variable in liste_des_elements %}
    boucler sur la variable en utilisant {{ variable }}
{% endfor %}
```

Exemple :

```
# cat boucle_for.yml
---
- name: test d'une boucle for
  hosts: all
  vars:
    liste: [Jean, Marc, Theo]
  tasks:
    - name: copie du fichier template
      template:
        src: template_boucle_for.j2
        dest: "/tmp/{{inventory_hostname}}.res"
      connection: local
...

```

```
# cat template_boucle_for.j2
Traitement sur le poste {{inventory_hostname}}
Exemple avec le tableau liste
Une liste de personnes :
  {% for personne in liste %}
    Prenom : {{personne}}
  {% endfor %}
Autre exemple avec le tableau ansible_interfaces:
La liste des interfaces est:
  {% for element in ansible_interfaces %}
    interface : {{element}}
  {% endfor %}
```


Exécution:

```
# ansible-playbook boucle_for.yml
```

Remarque:

La ligne '[connection: local](#)' permet de récupérer le résultat sur le poste local et non sur les machines clientes.

Résultat:

```
# cat /tmp/client1.res
Traitement sur le poste client1
Exemple avec le tableau liste
Une liste de personnes :
    Prenom: Jean
    Prenom: Marc
    Prenom: Theo
Autre exemple avec le tableau ansible_interfaces :
La liste des interfaces est :
    interface : lo
    interface : enp0s3
```

La déclaration d'un tableau peut aussi se faire avec la syntaxe ci-dessous.

```
# cat boucle_for2.yml
---
- name: test d'une boucle for
  hosts: all
  vars:
    liste:
      - Jean
      - Marc
      - Theo
  tasks:
    - name: copie du fichier template
      template:
        src: template_boucle_for.j2
        dest: "/tmp/{{inventory_hostname}}.res"
      connection: local
...
```

Les playbooks

Le module debug et le mot clef register

```
- debug: msg="le port http est {{ http_port }}"
```

```
- tasks:
  - shell: "echo commande 1; echo commande 2"
    register: resultat
  - debug:
      var=resultat

      var=resultat.stdout_lines

      var=resultat.stdout_lines[-1]
```

```
- debug: msg={{resultat.stdout_lines}}
```

Le module debug et le mot clef register

Le module **debug** permet, entre autres, d'afficher des informations à la suite de l'exécution du playbook. Elles portent en particulier sur les caractéristiques d'exécution d'une tâche.

Ce module permet de récupérer la valeur d'une variable: `msg`

```
# tail -3 playbook_template.yml
- debug:
  msg: "le port http est {{ http_port }}"
- debug: msg="le nom de domaine est {{ domain }}"
...
```

```
# ansible-playbook playbook_template.yml | grep -A 3 "debug"

TASK [debug] *****
ok: [CentOS7.1] => {
  "msg": "le port http est 80"
}

TASK [debug] *****
ok: [CentOS7.1] => {
  "msg": "le nom de domaine est mondomaine.lan"
}
```

Les variables ont bien été remplacé par leur valeur.

Le mot clef **register** permet de récupérer l'état d'exécution d'une tâche via une variable.

```
# cat debug.yml
---
- name: Test debug
  hosts: centos1
  tasks:
    - name: Commande 1 et 2
      shell:
        echo "commande 1" ; echo "commande 2"
      register: resultat
    - name: Commande 3
      command:
        echo "commande 3"
    - name: Affichage de la variable resultat
      debug:
        var: resultat
...
```

Affichage de la totalité de la « sortie » du contenu de la variable resultat:

var=resultat

ou

var: resultat

Exécution du playbook:

```
# ansible-playbook debug.yml

PLAY [Test debug] *****

TASK [Gathering Facts]*****
ok: [centos1]

TASK [Commande 1 et 2] *****
changed: [centos1]

TASK [Commande 3] *****
changed: [centos1]

TASK [Affichage de la variable resultat] *****
ok: [centos1] => {
  "resultat": {
    "changed": true,
    "cmd": "echo \"commande 1\" ; echo \"commande 2\"",
    "delta": "0:00:00.009313",
    "end": "2018-12-28 14:56:54.505716",
    "failed": false,
    "rc": 0,
    "start": "2018-12-28 14:56:54.496403",
    "stderr": "",
    "stderr_lines": [],
    "stdout": "commande 1\ncommande 2",
    "stdout_lines": [
      "commande 1",
      "commande 2"
    ]
  }
}

PLAY RECAP *****
centos1 : ok=4 changed=2 unreachable=0 failed=0
```

Autres possibilités d'exploitation de la « sortie » du 'register' :

var=resultat.stdout_lines

```
# tail -5 debug2.yml
- name: Affichage de la variable resultat.stdout_lines
  debug:
    var: resultat.stdout_lines
...

TASK [Affichage de la variable resultat.stdout_lines] *****
ok: [centos1] => {
  "resultat.stdout_lines": [
    "commande 1",
    "commande 2"
  ]
}
```

var=resultat.stdout_lines[index]

```
# tail -5 debug3.yml
- name: Affichage du dernier element du tableau resultat.stdout_lines
  debug:
    var: resultat.stdout_lines[-1]
...

TASK [Affichage du dernier element du tableau resultat.stdout_lines] *****
ok: [centos1] => {
  "resultat.stdout_lines[-1]": "commande 2"
}
```

Autre exemple avec stdout_lines pour récupérer le résultat d'une commande :

```
# cat register.yml
---
- name: Enregistrement des fichiers /etc/hosts/*
  hosts: centos1
  tasks:
    - name: stocker le resultat d'execution dans la variable fic
      shell: "ls /etc/host*"
      register: solution
    - name: afficher solution.stdout_lines
      debug:
        msg: "{{solution.stdout_lines}}"
...

# ansible-playbook register.yml
TASK [afficher solution.stdout_lines] *****
ok: [centos1] => {
  "msg": [
    "/etc/host.conf",
    "/etc/hostname",
    "/etc/hosts",
    "/etc/hosts.allow",
    "/etc/hosts.deny"
  ]
}
```

Les playbooks

Les Handlers

```
- name: Test handler
hosts: client1
handlers:
  - name: Redemarrage d un service
    service:
      name: crond
      state: restarted
tasks:
  - name: Fichier de conf d'un service
    copy:
      src: modele.conf
      dest: /tmp/crond.conf
      register: resultat
    notify: Redemarrage d un service
```

Les Handlers

Une tâche associée à un handler n'est exécutée que si nécessaire. Un handler n'est exécuté que s'il est appelé. Son exécution se fera après le traitement de toutes les tâches. S'il est appelé par plusieurs tâches, il ne sera exécuté qu'une seule fois.

Quelque soit l'ordre dans lequel des handlers sont appelés, ils ne s'exécuteront que dans l'ordre dans lequel ils ont été défini au sein de la section handlers.

Déclaration d'un handler par le mot clef : handlers

Appel d'un handler via le mot clef : notify

Exemple:

```
# cat handler.yml
---
- name: Test handler
  hosts: centos1
  handlers:
    - name: Redemarrage du service crond
      service:
        name: crond
        state: restarted
  tasks:
    - name: Copie du fichier de configuration du service
      copy:
        src: modele.conf
        dest: /tmp/crond.conf
        register: resultat
      notify: Redemarrage du service crond
    - name: Recuperation du PID de crond
      shell:
        ps -ef | grep crond > /tmp/pid_crond
    - name: Affichage de la variable resultat.changed
      debug:
        var: resultat.changed
...

```

Exemple:

Arret du service crond:

```
# ansible centos1 -m service -a 'name=crond state=stopped'
```

Execution du playbook:

```
# ansible-playbook handler.yml

PLAY [Test handler] *****

TASK [Gathering Facts] *****
ok: [centos1]

TASK [Copie du fichier de configuration du service] *****
changed: [centos1]

TASK [Recuperation du PID de crond] *****
changed: [centos1]

TASK [Affichage de la variable resultat.changed] *****
ok: [centos1] => {
    "resultat.changed": true
}

RUNNING HANDLER [Redemarrage du service crond] *****
changed: [centos1]

PLAY RECAP *****
centos1 : ok=5 changed=3 unreachable=0 failed=0
```

La variable resultat.changed est à true. L'handler est bien sollicité juste après la tâche qui l'a appelé via le mot clef notify.

Deuxième exécution. Le service crond est à nouveau arrêté au préalable.

```
# ansible centos1 -m service -a 'name=crond state=stopped'
```

```
# ansible-playbook handler.yml

PLAY [Test handler] *****

TASK [Gathering Facts] *****
ok: [centos1]

TASK [Copie du fichier de configuration du service] *****
ok: [centos1]

TASK [Recuperation du PID de crond] *****
changed: [centos1]

TASK [Affichage de la variable resultat.changed] *****
ok: [centos1] => {
    "resultat.changed": false
}

PLAY RECAP *****
centos1 : ok=4 changed=1 unreachable=0 failed=0
```

La variable resultat.changed est à false. Le fichier de configuration n'ayant pas été copié, l'handler n'a pas été sollicité.

Les playbooks

Les boucles

```
- shell: echo "{{item}}" >>/tmp/fic
  loop:
    - element1
    - element2
    - element4
    - element5

    equipe: [ {nom: jean, uid: 1001}, {nom: marc, uid: 1002} ]
- debug:
  msg:
    "xx {{item.nom}} xx {{item.uid}}"
  loop:
    "{{equipe}}"

- command: echo "{{ item }}"
  loop: [ 0, 2, 4, 6, 8, 10 ]
```

Les boucles

Les boucles utilisent les mots clefs **loop**

Syntaxe :

```
- shell: echo "{{item}}" >>/tmp/fic
  loop:
    - element1
    - element2
    - element3
    - element4
    - element5
```

item est le nom de la variable dans une boucle loop

Syntaxe avec un tableau :

equipe: [{nom: jean, uid: 1001}, {nom: marc, uid: 1002}]

```
- debug:
  msg:
    "xx {{item.nom}} xx {{item.uid}}"
  loop: "{{equipe}}"

- command: echo "{{ item }}"
  loop: [ 0, 2, 4, 6, 8, 10 ]
```

Remarque : le mot clé with_items est deprecated pour les boucles.

Exemple:

```
# cat boucle_loop.yml
---
- name: Test de la boucle loop
  hosts: centos1
  vars:
    equipe:
      - nom: jean
        uid: 1001
      - nom: marc
        uid: 1002
      - nom: eve
        uid: 1003
  tasks:
    - name: Stocker les fichiers /etc/host* dans la variable resultat
      shell: ls /etc/host*
      register: resultat
    - name: Boucler sur resultat.stdout_lines
      shell: echo "{{item}}" >> /tmp/boucle.res
      connection: local
      loop: "{{resultat.stdout_lines}}"
    - name: Afficher le contenu du tableau equipe
      debug:
        msg: "Bonjour {{item.nom}}, ton uid est {{item.uid}}"
      loop: "{{equipe}}"
...
```

```
# ansible-playbook boucle_loop.yml

PLAY [Test de la boucle loop] *****

TASK [Gathering Facts] *****
ok: [centos1]

TASK [Stocker les fichiers /etc/host* dans la variable resultat] *****
changed: [centos1]

TASK [Boucler sur resultat.stdout_lines] *****
changed: [centos1] => (item=/etc/host.conf)
changed: [centos1] => (item=/etc/hostname)
changed: [centos1] => (item=/etc/hosts)
changed: [centos1] => (item=/etc/hosts.allow)
changed: [centos1] => (item=/etc/hosts.deny)

TASK [Afficher le contenu du tableau equipe] *****
ok: [centos1] => (item={u'nom': u'jean', u'uid': 1001}) => {
  "msg": "Bonjour jean, ton uid est 1001"
}
ok: [centos1] => (item={u'nom': u'marc', u'uid': 1002}) => {
  "msg": "Bonjour marc, ton uid est 1002"
}
ok: [centos1] => (item={u'nom': u'eve', u'uid': 1003}) => {
  "msg": "Bonjour eve, ton uid est 1003"
}

PLAY RECAP *****
centos1 : ok=4    changed=2    unreachable=0    failed=0
```

```
# cat /tmp/boucle.res
/etc/host.conf
/etc/hostname
/etc/hosts
/etc/hosts.allow
/etc/hosts.deny
```


Autres exemples:

```
# cat ajout_users.yml
---
- name: Ajout des utilisateurs sur les machines centos
  hosts: centos_servers
  tasks:
    - name: Creation des utilisateurs user1 à user5
      user:
        name: "{{item}}"
        state: present
        groups: users
      loop:
        - user1
        - user2
        - user3
        - user4
        - user5
...

```

Extrait du résultat d'exécution.

```
TASK [Creation des utilisateurs user1 à user5] *****
changed: [centos2] => (item=user1)
changed: [centos1] => (item=user1)
changed: [centos2] => (item=user2)
changed: [centos1] => (item=user2)
changed: [centos2] => (item=user3)
changed: [centos1] => (item=user3)
changed: [centos2] => (item=user4)
changed: [centos1] => (item=user4)
changed: [centos2] => (item=user5)
changed: [centos1] => (item=user5)

```

```
# cat ajout_users_uid.yml
---
- name: Ajout des utilisateurs sur les machines centos
  hosts: centos_servers
  tasks:
    - name: Creation des utilisateurs user1 à user5 avec des uid specifiques
      user:
        name: "user{{item}}"
        state: present
        groups: users
        uid: "1000{{item}}"
      loop:
        - 1
        - 2
        - 3
        - 4
        - 5
...

```

Le mot clef loop_control:

Le mot clef **loop_control** permet d'exploiter les paramètres internes aux boucles.
On peut redéfinir la variable item par une autre variable via loop_var, ou utiliser l'index de l'élément en cours de la liste via index_var.

```
# cat user_loopcontrol.yml
---
- name: Playbook pour ajouter des utilisateurs
  hosts: all
  tasks:
    - name: Ajout d'utilisateurs
      user:
        name: "{{nom_user}}"
        state: present
        uid: "100{{index_uid}}"
        home: "/home/{{nom_user}}"
      loop:
        - jean
        - marc
        - theo
      loop_control:
        loop_var: nom_user
        index_var: index_uid
...

```

Cela permet notamment de boucler dans une boucle

Exemple:

Boucle principale.

```
# cat boucle_loopcontrol.yml
---
- name: Test d'une inclusion de boucle
  hosts: centos1
  tasks:
    - name: Inclusion du fichier boucle_inner.yml
      include_tasks: boucle_inner.yml
      loop:
        - Entree
        - Plat
        - Dessert
      loop_control:
        loop_var: var_boucle_externe
        index_var: mon_index
...

```

Boucle incluse.

```
# cat boucle_inner.yml
---
- name: "Execution de ma boucle incluse {{mon_index}}"
  debug:
    msg: "Index {{mon_index}} var boucle externe: {{var_boucle_externe}}
          var boucle incluse: {{item}}"
  loop:
    - choix1
    - choix2
    - choix3
...

```

Résultat d'exécution.

```
# ansible-playbook  boucle_loopcontrol.yml

PLAY [Test d'une inclusion de boucle] *****

TASK [Gathering Facts] *****
ok: [centos1]

TASK [Inclusion du fichier boucle_inner.yml] *****
included: /root/boucle_inner.yml for centos1
included: /root/boucle_inner.yml for centos1
included: /root/boucle_inner.yml for centos1

TASK [Execution de ma boucle include 0] *****
ok: [centos1] => (item=choix1) => {
    "msg": "Index 0 var boucle externe: Entree var boucle include: choix1"
}
ok: [centos1] => (item=choix2) => {
    "msg": "Index 0 var boucle externe: Entree var boucle include: choix2"
}
ok: [centos1] => (item=choix3) => {
    "msg": "Index 0 var boucle externe: Entree var boucle include: choix3"
}

TASK [Execution de ma boucle include 1] *****
ok: [centos1] => (item=choix1) => {
    "msg": "Index 1 var boucle externe: Plat var boucle include: choix1"
}
ok: [centos1] => (item=choix2) => {
    "msg": "Index 1 var boucle externe: Plat var boucle include: choix2"
}
ok: [centos1] => (item=choix3) => {
    "msg": "Index 1 var boucle externe: Plat var boucle include: choix3"
}

TASK [Execution de ma boucle include 2] *****
ok: [centos1] => (item=choix1) => {
    "msg": "Index 2 var boucle externe: Dessert var boucle include: choix1"
}
ok: [centos1] => (item=choix2) => {
    "msg": "Index 2 var boucle externe: Dessert var boucle include: choix2"
}
ok: [centos1] => (item=choix3) => {
    "msg": "Index 2 var boucle externe: Dessert var boucle include: choix3"
}

PLAY RECAP *****
centos1                : ok=7    changed=0    unreachable=0    failed=0    failed=0
```

Les playbooks

La condition when

```
when: resultat.changed          resultat.changed est un booléen
when: resultat.changed == True  idem que précédemment
when: resultat.changed == False
when: resultat.dest == "/tmp/crond.conf"
when: resultat.dest != "/tmp/crond.conf"
```

```
- name: Fichier de conf d'un service
  copy:
    src: modele.conf
    dest: /tmp/crond.conf
  register: resultat
- name: Redemarrage d'un service
  service:
    name: crond
    state: restarted
  when: resultat.changed
```

La condition when

La condition when permet l'exécution de la tâche si le test associé est vrai.
Il est donc possible d'activer une tâche à partir d'une valeur d'une variable.

Syntaxe:

```
when: resultat.changed          resultat.changed est un booléen
when: resultat.changed == True  idem que précédemment
when: resultat.changed == False
when: resultat.dest == "/tmp/crond.conf"
when: resultat.dest != "/tmp/crond.conf"
```

Exemple:

```
# cat when.yml
---
- name: Test when
  hosts: centos1
  tasks:
    - name: Copie du fichier de configuration du service
      copy:
        src: modele.conf
        dest: /tmp/crond.conf
      register: resultat
    - name: Redemarrage du service crond
      service:
        name: crond
        state: restarted
      when: resultat.changed
    - name: Afficher la valeur de resultat.changed
      debug: var=resultat.changed
...

```

Première exécution:

La tâche « Redémarrage d'un service » s'exécute car la copie s'est réalisée.

```
# ansible-playbook when.yml

PLAY [Test when] *****

TASK [Gathering Facts] *****
ok: [centos1]

TASK [Copie du fichier de configuration du service] *****
changed: [centos1]

TASK [Redémarrage du service crond] *****
changed: [centos1]

TASK [Afficher la valeur de resultat.changed] *****
ok: [centos1] => {
  "resultat.changed": true
}

PLAY RECAP *****
centos1 : ok=4    changed=2    unreachable=0    failed=0
```

Deuxième exécution:

La tâche « Redémarrage d'un service » ne s'exécute pas car la copie ne s'est pas faite ce qui est confirmé par la valeur de la variable resultat.changed.

```
# ansible-playbook when.yml

PLAY [Test when] *****

TASK [Gathering Facts] *****
ok: [centos1]

TASK [Copie du fichier de configuration du service] *****
ok: [centos1]

TASK [Redémarrage du service crond] *****
skipping: [centos1]

TASK [Afficher la valeur de resultat.changed] *****
ok: [centos1] => {
  "resultat.changed": false
}

PLAY RECAP *****
centos1 : ok=3    changed=0    unreachable=0    failed=0
```

D'autres possibilités de tests avec la syntaxe 'when : condition'

Exemple 2: test sur un nombre

```
# cat when2.yml
---
- name: Test de when
  hosts: localhost
  tasks:
    - name: Affichage des variables supérieures à cinq
      shell: echo "{{item}}" >> /tmp/resultat
      loop: [0, 2, 4, 6, 8, 10]
      when: item > 5
...
```

```
TASK [Affichage des variables supérieures à cinq] *****
skipping: [localhost] => (item=0)
skipping: [localhost] => (item=2)
skipping: [localhost] => (item=4)
changed: [localhost] => (item=6)
changed: [localhost] => (item=8)
changed: [localhost] => (item=10)
```

Les opérateurs suivants sont disponibles:

```
<, lt, <=, le, >, gt, >=, ge, ==, eq, !=, <>, ne
```

```
# cat when2bis.yml
---
- name: Afficher les machines dont la version est supérieure à 6.5
  hosts: all
  tasks:
    - name: Afficher les distributions supérieure à 6.5
      debug:
        msg: La version est supérieure a 6.5
      when: ansible_distribution_version >= '6.5'
...
```

Extrait du résultat d'exécution.

```
TASK [Afficher les distributions supérieure à 6.5] *****
ok: [centos1] => {
  "msg": "La version est supérieure a 6.5"
}
ok: [centos2] => {
  "msg": "La version est supérieure a 6.5"
}
ok: [debian1] => {
  "msg": "La version est supérieure a 6.5"
}
```

Exemple 3: match et search.

```
# cat when3.yml
---
- name: Tests de when
  hosts: localhost
  vars:
    url: http://example.com/users/foo/resources/bar
  tasks:
    - name: Utilisation de url is match
      debug:
        msg: L'url correspond
      when: url is match("http://example.com/users/*/resources/*")
    - name: Utilisation de url is search
      debug:
        msg: L'url contient le motif
      when: url is search("/users/*/resources/*")
    - name: Utilisation de url is search
      debug:
        msg: L'url contient /users/
      when: url is search("/users/")
...
```

```
# ansible-playbook when3.yml
PLAY [Tests de when] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Utilisation de url is match] *****
ok: [localhost] => {
  "msg": "L'url correspond"
}

TASK [Utilisation de url is search] *****
ok: [localhost] => {
  "msg": "L'url contient le motif"
}

TASK [Utilisation de url is search] *****
ok: [localhost] => {
  "msg": "L'url contient /users/"
}
```

Remarques :

- Pour les tests, on peut utiliser les mots clés `is` et `is not`.
- On peut combiner plusieurs tests avec les mots clés `or` et `and`.

Exemple 4: Les mots clefs pour des tests sur les fichiers sont les suivants:

`directory, file, link, exists, same_file(fichier2), mount`

```
# cat when4.yml
---
- name: Tests de when
  hosts: localhost
  vars:
    liste:
      - /etc
      - /etc/passwd
      - /xxx
  tasks:
    - name: test si c'est un repertoire
      debug:
        msg: "{{item}} est un repertoire"
      when: item is directory
      loop: "{{liste}}"
    - name: test existence
      debug:
        msg: "{{item}} existe"
      when: item is exists
      loop: "{{liste}}"
...
```

```
# ansible-playbook when4.yml
PLAY [Tests de when] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [test si c'est un repertoire] *****
ok: [localhost] => (item=/etc) => {
  "msg": "/etc est un repertoire"
}
skipping: [localhost] => (item=/etc/passwd)
skipping: [localhost] => (item=/xxx)

TASK [test existence] *****
ok: [localhost] => (item=/etc) => {
  "msg": "/etc existe"
}
ok: [localhost] => (item=/etc/passwd) => {
  "msg": "/etc/passwd existe"
}
skipping: [localhost] => (item=/xxx)
```


Exemple 5:

```
# cat when5.yml
---
- name: Nouveau test when
  hosts: all
  tasks:
    - name: Execution d'une commande inexistante
      shell: /usr/bin/foo
      register: result
      ignore_errors: true
    - name: Action sur commande en echec
      debug:
        msg: Action si c'est un echec
      when: result is failed
...
```

Les mots clefs suivants sont disponibles :

accept, failed, changed, succeeded, success, skipped

ignore_errors à true permet de continuer l'exécution du playbook malgré une erreur.

Exécution du playbook:

```
# ansible-playbook when5.yml

PLAY [Nouveau test when] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Execution d'une commande inexistante] *****
fatal: [localhost]: FAILED! => {"changed": true, "cmd": "/usr/bin/foo", "delta":
"0:00:00.013890", "end": "2018-12-29 23:55:37.913900", "msg": "non-zero return code",
"rc": 127, "start": "2018-12-29 23:55:37.900010", "stderr": "/bin/sh: /usr/bin/foo: Aucun
fichier ou dossier de ce type", "stderr_lines": ["/bin/sh: /usr/bin/foo: Aucun fichier ou
dossier de ce type"], "stdout": "", "stdout_lines": []}
...ignoring

TASK [Action sur commande en echec] *****
ok: [localhost] => {
  "msg": "Action si c'est un echec"
}

PLAY RECAP *****
localhost : ok=3    changed=1    unreachable=0    failed=0
```

Les playbooks

Les include et les import

```
include_tasks: fichier_de_taches.yml
include_tasks: fichier_de_taches.yml  var1=val1  var2=val2
import_tasks: fichier_de_taches.yml
```

```
include_vars: fichier_variables.yml
```

```
tasks:
  - include_vars: variables/variables.yml
  - include_tasks: tasks/deploiement_baseDeDonnees.yml  bdd=base1.sql
  - include_tasks: tasks/creation_du_site.yml
```

Les include et les import

On a la possibilité d'intégrer les tasks d'un autre fichier au sein d'un playbook.

Le mot clef `include` est obsolète, on utilise **`include_tasks`** (comportement dynamique) ou **`import_tasks`** (comportement statique).

Syntaxe:

```
include_tasks: autre_fichier_de_taches.yml
include_tasks: autre_fichier_de_taches.yml  var1=val1  var2=val2
import_tasks: autre_fichier_de_taches.yml
```

Ansible pré-traite toutes les importations statiques au cours du temps d'analyse du Playbook. Les inclusions dynamiques sont traitées pendant l'exécution au moment où cette tâche est rencontrée. Ainsi, on utilise `include_tasks` lorsqu'il y a des mots-clefs, boucles et conditions.

Le principal avantage de l'utilisation des instructions `include` est la mise en boucle. Lorsqu'une boucle est utilisée avec un `include`, les tâches ou le rôle inclus seront exécutés une fois pour chaque élément de la boucle.

Il existe également **`include_vars`** pour intégrer un fichier de variables au sein d'une tâche.

Exemple 1 pour include tasks:

Le playbook principal

```
# cat include1.yml
---
- name: Test d'include_tasks
  hosts: centos_servers
  tasks:
    - name: Tache 1
      debug:
        msg: "Traitement UN machine {{ inventory_hostname }}"
    - name: Inclusion d'une autre tache
      include_tasks: include_autre.yml
    - name: Tache 2
      debug:
        msg: "Traitement DEUX machine {{ inventory_hostname }}"
...
```

La tâche à inclure.

```
# cat include_autre.yml
---
- name: Fichier d'inclusion
  debug:
    msg: autre tache
...
```

Exécution du playbook.

```
# ansible-playbook include1.yml

PLAY [Test d'include_tasks] *****

TASK [Gathering Facts] *****
ok: [centos1]
ok: [centos2]

TASK [Tache 1] *****
ok: [centos1] => {
  "msg": "Traitement UN machine centos1"
}
ok: [centos2] => {
  "msg": "Traitement UN machine centos2"
}

TASK [Inclusion d'une autre tache] *****
included: /root/include_autre.yml for centos1, centos2

TASK [Fichier d'inclusion] *****
ok: [centos1] => {
  "msg": "autre tache"
}
ok: [centos2] => {
  "msg": "autre tache"
}

TASK [Tache 2] *****
ok: [centos1] => {
  "msg": "Traitement DEUX machine centos1"
}
ok: [centos2] => {
  "msg": "Traitement DEUX machine centos2"
}

PLAY RECAP *****
centos1           : ok=5    changed=0    unreachable=0    failed=0
centos2           : ok=5    changed=0    unreachable=0    failed=0
```

Exemple 2 pour include tasks:

```
# cat include2.yml
---
- name: test d'include_tasks
  hosts: centos_servers
  tasks:
    - name: Tache 1
      debug:
        msg: tache1 système {{ inventory_hostname }} debut
    - name: Inclusion d'une autre tache
      include_tasks: include_autre3.yml prenom="Paul"
...

# cat include_autre3.yml
---
- name: Fichier d'inclusion
  debug:
    msg: "tache avec {{ prenom }}"
...

```

```
# ansible-playbook include2.yml

PLAY [test d'include_tasks] *****

TASK [Gathering Facts] *****
ok: [centos2]
ok: [centos1]

TASK [Tache 1] *****
ok: [centos1] => {
  "msg": "tache1 système centos1 debut"
}
ok: [centos2] => {
  "msg": "tache1 système centos2 debut"
}

TASK [Inclusion d'une autre tache] *****
included: /root/include_autre3.yml for centos1, centos2

TASK [Fichier d'inclusion] *****
ok: [centos1] => {
  "msg": "tache avec Paul"
}
ok: [centos2] => {
  "msg": "tache avec Paul"
}

PLAY RECAP *****
centos1          : ok=4    changed=0    unreachable=0    failed=0
centos2          : ok=4    changed=0    unreachable=0    failed=0

```

Exemple3: différence entre include et import

```
# cat include3.yml
---
- name: Test d'include_tasks
  hosts: centos_servers
  tasks:
    - name: Tache 1
      debug:
        msg: tache1
    - name: Inclure le fichier si hostvar est défini
      include_tasks: "{{hostvar}}.yaml"
      when: hostvar is defined
...

# cat include_autre.yml
---
- name: Fichier d'inclusion
  debug:
    msg: "tache autre"
...
```

```
# ansible-playbook include3.yml -e hostvar=include_autre
PLAY [Test d'include_tasks] *****

TASK [Gathering Facts] *****
ok: [centos1]
ok: [centos2]

TASK [Tache 1] *****
ok: [centos1] => {
  "msg": "tache1"
}
ok: [centos2] => {
  "msg": "tache1"
}

TASK [Inclure le fichier si hostvar est défini] *****
included: /root/include_autre.yml for centos1, centos2

TASK [Fichier d'inclusion] *****
ok: [centos1] => {
  "msg": "autre tache"
}
ok: [centos2] => {
  "msg": "autre tache"
}

PLAY RECAP *****
centos1           : ok=4    changed=0    unreachable=0    failed=0
centos2           : ok=4    changed=0    unreachable=0    failed=0
```

```
# ansible-playbook include3.yml
PLAY [Test d'include_tasks] *****

TASK [Gathering Facts] *****
ok: [centos2]
ok: [centos1]

TASK [Tache 1] *****
ok: [centos1] => {
  "msg": "tache1"
}
ok: [centos2] => {
  "msg": "tache1"
}

TASK [Inclure le fichier si hostvar est défini] *****
skipping: [centos1]
skipping: [centos2]

PLAY RECAP *****
centos1           : ok=2    changed=0    unreachable=0    failed=0
centos2           : ok=2    changed=0    unreachable=0    failed=0
```

Exemple avec import tasks :

```
# cat import3.yml                                     # cat include_autre.yml
---
- name: Test d'include_tasks
  hosts: centos_servers
  tasks:
    - name: Tache 1
      debug:
        msg: tache1
    - name: Inclure le fichier si hostvar est défini
      import_tasks: "{{hostvar}}.yaml"
      when: hostvar is defined
...

```

Exécution du playbook.

```
# ansible-playbook import3.yml -e hostvar=include_autre

PLAY [Test d'include_tasks] *****

TASK [Gathering Facts] *****
ok: [centos1]
ok: [centos2]

TASK [Tache 1] *****
ok: [centos1] => {
  "msg": "tache1"
}
ok: [centos2] => {
  "msg": "tache1"
}

TASK [Fichier d'inclusion] *****
ok: [centos1] => {
  "msg": "autre tache"
}
ok: [centos2] => {
  "msg": "autre tache"
}

PLAY RECAP *****
centos1           : ok=3    changed=0    unreachable=0    failed=0
centos2           : ok=3    changed=0    unreachable=0    failed=0

```

Exécution du play book sans définir la variable hostvar.

```
# ansible-playbook import3.yml
ERROR! Error when evaluating variable in include name: {{ hostvar }}.yaml.

When using static includes, ensure that any variables used in their names are defined in
vars/vars_files or extra-vars passed in from the command line. Static includes cannot use
variables from inventory sources like group or host vars.

```

Échec avec import alors qu'avec include cela a fonctionné.

Exemple avec include tasks et include vars:

Un répertoire variables avec variables.yml :

```
# cat variables/variables.yml
---
equipe:
  - nom: jean
    uid: 1001
  - nom: marc
    uid: 1002
ip_addr: "{{ ansible_all_ipv4_addresses[0] }}"
appli_path: "{{ ip_addr }}/22"
...
```

Un répertoire tasks avec les 3 fichiers pour des tasks :

```
# cat tasks/boucle.yml
---
- name: Stocker les fichiers /etc/host* dans la variable resultat
  shell: ls /etc/host*
  register: resultat
- name: Boucler sur resultat.stdout_lines
  shell: echo "{{item}}" >> /tmp/boucle.res
  connection: local
  loop: "{{resultat.stdout_lines}}"
- name: Afficher le contenu du tableau equipe
  debug:
    msg: "Bonjour {{item.nom}}, ton uid est {{item.uid}}"
  loop: "{{equipe}}"
...

# cat tasks/regist.yml
---
- name: Stocker les fichiers /etc/host* dans la variable resultat
  shell: ls /etc/host*
  register: resultat
- name: Afficher resultat.stdout_lines
  debug:
    msg: resultat.stdout_lines
...

# cat tasks/variables_base.yml
---
- name: Afficher la variable appli_path
  debug:
    msg: La variable appli_path = {{ appli_path }}
...
```

Le playbook :

```
# cat include_exemple1.yml
---
- name: Test d'inclusions
  hosts: centos_servers
  vars_prompt:
    - name: motdepasse_admin
      prompt: Saisir le mot de passe pour la base de donnees
      # un prompt avec une saisie masquée
  tasks:
    - include_vars: variables/variables.yml
    - include_tasks: tasks/boucle.yml
    - include_tasks: tasks/regist.yml
    - include_tasks: tasks/variables_base.yml

    - name: une autre tasks
      debug:
        msg: une autre operation avec le mot de passe {{motdepasse_admin}}
...
```

Exécution du playbook.

```
# ansible-playbook include_exemple1.yml
Saisir le mot de passe pour la base de donnees:

PLAY [Test d'inclusions] *****

TASK [Gathering Facts] *****
ok: [centos2]
ok: [centos1]

TASK [include_vars] *****
ok: [centos1]
ok: [centos2]

TASK [include_tasks] *****
included: /root/tasks/boucle.yml for centos1, centos2

TASK [Stocker les fichiers /etc/host* dans la variable resultat] *****
changed: [centos1]
changed: [centos2]

TASK [Boucler sur resultat.stdout_lines] *****
changed: [centos1] => (item=/etc/hosts.allow)
changed: [centos2] => (item=/etc/hosts.allow)
changed: [centos1] => (item=/etc/hosts.deny)
changed: [centos2] => (item=/etc/hosts.deny)
changed: [centos1] => (item=/etc/hosts)
changed: [centos2] => (item=/etc/hosts)
changed: [centos1] => (item=/etc/hostname)
changed: [centos2] => (item=/etc/hostname)
changed: [centos1] => (item=/etc/hosts.conf)
changed: [centos2] => (item=/etc/hosts.conf)

TASK [Afficher le contenu du tableau equipe] *****
ok: [centos1] => (item={u'nom': u'jean', u'uid': 1001}) => {
  "msg": "Bonjour jean, ton uid est 1001"
}
ok: [centos1] => (item={u'nom': u'marc', u'uid': 1002}) => {
  "msg": "Bonjour marc, ton uid est 1002"
}
ok: [centos2] => (item={u'nom': u'jean', u'uid': 1001}) => {
  "msg": "Bonjour jean, ton uid est 1001"
}
ok: [centos2] => (item={u'nom': u'marc', u'uid': 1002}) => {
  "msg": "Bonjour marc, ton uid est 1002"
}

TASK [include_tasks] *****
included: /root/tasks/regist.yml for centos1, centos2

TASK [Stocker les fichiers /etc/host* dans la variable resultat] *****
changed: [centos1]
changed: [centos2]

TASK [Afficher resultat.stdout_lines] *****
ok: [centos1] => {
  "msg": "resultat.stdout_lines"
}
ok: [centos2] => {
  "msg": "resultat.stdout_lines"
}

TASK [include_tasks] *****
included: /root/tasks/variables_base.yml for centos1, centos2

TASK [Afficher la variable appli_path] *****
ok: [centos1] => {
```



```
    "msg": "La variable appli_path = 192.168.1.49/22"
  }
ok: [centos2] => {
    "msg": "La variable appli_path = 192.168.1.88/22"
  }

TASK [une autre tasks] *****
ok: [centos1] => {
    "msg": "une autre operation avec le mot de passe secret"
  }
ok: [centos2] => {
    "msg": "une autre operation avec le mot de passe secret"
  }

PLAY RECAP *****
centos1          : ok=12   changed=3    unreachable=0    failed=0
centos2          : ok=12   changed=3    unreachable=0    failed=0
```

Les playbooks

Les filtres

lower	pour convertir en minuscules	upper	pour convertir en majuscules
int	pour convertir en entier	float	pour convertir en nombre flottant
bool	pour convertir en booléen		

variable | default(valeur) si variable n'est pas définie, elle prend la valeur de 'valeur'
xxx | random pour une valeur aléatoire (60 | random : entre 0 et 60)

plugin ipaddr

plugin urlsplit

Les filtres

Ci-dessous quelques filtres :

lower	pour convertir en minuscules	upper	pour convertir en majuscules
int	pour convertir en entier	float	pour convertir en nombre flottant
bool	pour convertir en booléen		

Remarque: toute valeur passée par la ligne de commande est une chaîne de caractères.

```
# ansible-playbook filtre.yml -e mavar=4 -e monbool=true
```

```
# cat filtre.yml
- name: "Test filtre"
  hosts: client1
  tasks:
    - debug:
        msg: Machine CentOS
      when: ansible_distribution == "CentOS"
    - debug:
        msg: Machine CentOS avec lower donne centos
      when: ansible_distribution|lower == "centos"
    - debug:
        msg: Machine CentOS avec upper donne CENTOS
      when: ansible_distribution|upper == "CENTOS"
    - debug:
        msg: La variable est une chaîne de caractères "04" avec int on obtient un entier
      when: ansible_date_time.month|int == 4
    - debug:
        msg: La variable est une chaîne de caractères "04" n est pas convertie
        msg: ansible_date_time.month test nombre sans int
      when: ansible_date_time.month == 4
```

Test vrai

Test vrai

Test vrai

Test vrai

Test FAUX

```
- debug:
  msg: passage d un nombre en argument test nombre avec int
  when: mavar|int == 4                                Test vrai
- debug:
  msg: passage d un nombre en argument test nombre sans int
  when: mavar == 4                                    Test FAUX
- debug:
  msg: passage d un booleen en argument test nombre avec bool
  when: monbool|bool == true                          Test vrai
- debug:
  msg: passage d un booleen en argument test nombre sans bool
  when: monbool == true                              Test FAUX
```

D'autres filtres:

variable default(valeur)	si variable n'est pas définie, elle prend la valeur de 'valeur'
xxx random	pour une valeur aléatoire (60 random : entre 0 et 60)

Exemple:

```
# cat filtre2.yml
---
- name: Test de filtres
  hosts: localhost
  vars:
    liste:
      - path: /tmp/fichier1
      - path: /tmp/FICHER2
      - path: /tmp/Fichier3
      mode: 0444
  tasks:
    - name: Tache 1
      shell: echo path={{item.path}} et mode={{item.mode|default('5')}} >> jinja.res
      loop: "{{liste}}"
    - name: Tache 2
      shell: echo Minuscule={{item.path|lower}} et Majuscule={{item.path|upper}} >>
jinja.res
      loop: "{{liste}}"
    - name: Tache 3
      shell: echo Une valeur aleatoire entre 0 et 60 = {{60|random}} >> jinja.res
...
```

Résultat:

```
# cat jinja.res
path=/tmp/fichier1 et mode=5
path=/tmp/FICHER2 et mode=5
path=/tmp/Fichier3 et mode=0444
Minuscule=/tmp/fichier1 et Majuscule=/TMP/FICHER1
Minuscule=/tmp/fichier2 et Majuscule=/TMP/FICHER2
Minuscule=/tmp/fichier3 et Majuscule=/TMP/FICHER3
Une valeur aleatoire entre 0 et 60 = 22
```

Le plugin ipaddr:

Il permet de manipuler des adresses IP.

```
shell: echo Filtre ipaddr pour 192.168.10.9/24 = {{'192.168.10.9/24'|ipaddr('address')}} >> res
shell: echo Filtre ipaddr pour 300.168.10.9/24 = {{'300.168.10.9/24'|ipaddr('address')}} >> res
```

```
# cat filtre3.yml
---
- name: Test de filtres
  hosts: localhost
  tasks:
    - name: Vérification IP
      shell: echo Filtre ipaddr pour 192.168.10.9/24 = {{'192.168.10.9/24'|
ipaddr('address')}} >> res
    - name: Vérification IP
      shell: echo Filtre ipaddr pour 300.168.10.9/24 = {{'300.168.10.9/24'|
ipaddr('address')}} >> res
    ...
# ansible-playbook filtre3.yml
# cat res
Filtre ipaddr pour 192.168.10.9/24 = 192.168.10.9
Filtre ipaddr pour 300.168.10.9/24 = False
```

Le plugin urlsplit:

Il permet d'exploiter une url.

```
- shell: "echo http://www.serveur.fr:4500/chemin/page.htm le serveur =
      {{'http://www.serveur.fr:4500/chemin/page.htm'|urlsplit('hostname')}} >>jinja.res"
- shell: "echo http://www.serveur.fr:4500/chemin/page.htm le port =
      {{'http://www.serveur.fr:4500/chemin/page.htm'|urlsplit('port')}} >>jinja.res"
- shell: "echo http://www.serveur.fr:4500/chemin/page.htm le protocole =
      {{'http://www.serveur.fr:4500/chemin/page.htm'|urlsplit('scheme')}} >>jinja.res"
```

```
# cat filtre4.yml
---
- name: Test de filtres
  hosts: localhost
  tasks:
    - name: Récupération hostname
      shell: "echo http://www.serveur.fr:4500/chemin/page.htm le serveur =
      {{'http://www.serveur.fr:4500/chemin/page.htm'|urlsplit('hostname')}} >>
urlsplit.res"
    - name: Récupération du numéro de port
      shell: "echo http://www.serveur.fr:4500/chemin/page.htm le port =
      {{'http://www.serveur.fr:4500/chemin/page.htm'|urlsplit('port')}} >>
urlsplit.res"
    - name: Récupération du protocole
      shell: "echo http://www.serveur.fr:4500/chemin/page.htm le protocole =
      {{'http://www.serveur.fr:4500/chemin/page.htm'|urlsplit('scheme')}} >>
urlsplit.res"
    ...
# ansible-playbook filtre4.yml
# cat urlsplit.res
http://www.serveur.fr:4500/chemin/page.htm le serveur = www.serveur.fr
http://www.serveur.fr:4500/chemin/page.htm le port = 4500
http://www.serveur.fr:4500/chemin/page.htm le protocole = http
```

Les playbooks

Les opérations arithmétiques

```
- hosts: localhost
  tasks:
    - debug: msg="Memoire {{ansible_memtotal_mb * 1024}} kb"

    - debug: msg="Memoire {{ansible_memtotal_mb / 1024}} gb"

    - debug: msg="100/3 = {{100/3}}, partie entiere {{ (100/3)|int }}"
```

{{ mavar | pow(2) }} ...

Les opérations arithmétiques

```
# cat calcul.yml
---
- name: Test d'opérations arithmétiques
  hosts: localhost
  tasks:
    - name: Afficher la RAM en Mo et Ko
      debug:
        msg: Mem {{ansible_memtotal_mb}} mb ou {{ansible_memtotal_mb * 1024}} kb
    - name: Afficher la RAM en Go
      debug:
        msg: Mem {{(ansible_memtotal_mb / 1024)|round(2)}} gb
    - name: Calcul
      debug:
        msg: 100/3 = {{(100/3)|round(4,'ceil')}} , partie entiere {{ (100/3)|int }}
```

round(4,'ceil'): 4 chiffres significatifs après la virgule avec un arrondi haut (floor pour un arrondi bas).

Il est possible de réaliser des opérations plus complexes:

{{ mavar | pow(2) }} mavar à la puissance 2.

et bien d'autres présentées sur le site de documentation Ansible.

Résultat d'exécution.

```
# ansible-playbook calcul.yml

PLAY [Test d'opérations arithmétiques] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Afficher la RAM en Mo et Ko] *****
ok: [localhost] => {
  "msg": "Mem 1838 mb ou 1882112 kb"
}

TASK [Afficher la RAM en Go] *****
ok: [localhost] => {
  "msg": "Mem 1.79 gb"
}

TASK [Calcul] *****
ok: [localhost] => {
  "msg": "100/3 = 33.3334 , partie entiere 33"
}

PLAY RECAP *****
localhost                : ok=4    changed=0    unreachable=0    failed=0
```

Notes

Les rôles

Dans ce chapitre nous allons étudier la création, la structure et le fonctionnement des rôles.

Les rôles

- Présentation
- Structure d'un rôle
- Exécution d'un rôle
- Un exemple de rôle
- Un exemple de rôle avec des inclusions
- Ansible Galaxy

Les rôles

Présentation

- Organisation / arborescence
Simplification de l'administration
- Codes ré-exploitable
- Site de partage / Ansible Galaxy

Présentation

Pour une utilisation d'Ansible en production ou en développement, on obtient rapidement un ensemble conséquent de playbooks, de fichiers d'inventaires, de templates, etc.

Les rôles vont permettre :

- d'organiser l'ensemble de ces fichiers au sein d'une arborescence cohérente et « normalisée » (identique pour tous les rôles),
- de rendre les codes plus facilement ré-exploitable,
- de déployer simplement des rôles existants d'un site de partage.

Les actions à effectuer sur un serveur sont regroupés au sein d'un playbook.

Les rôles permettent de simplifier l'administration et d'automatiser les directives include au sein des fichiers de configuration. Il n'est plus nécessaire de préciser le chemin des fichiers de variables, ceux ci étant stockés dans des emplacements pré-définis.

Les rôles sont un moyen de charger automatiquement les tâches, les variables et les handlers.

Le playbook fera appel aux rôles qui auront besoin d'être exécutés. Depuis le répertoire tasks du rôle, tous les chemins sont relatifs.

Une plate-forme dédiée, Ansible Galaxy, permet de télécharger des rôles. Il n'est pas nécessaire de recréer ce qui a déjà été fait.

Les rôles

Structure d'un rôle

Nom	Description
tasks	Contient la liste des tâches utilisées par le rôle
handlers	Contient la liste des handlers utilisés par le rôle
defaults	Contient les variables par défaut du rôle
vars	Contient les autres variables du rôle. Elles prennent le dessus sur celles de defaults. En général, ce sont les variables modifiables par l'utilisateur.
files	Contient les fichiers utilisés via ce rôle (pour copy, ...)
templates	Contient les templates du rôle
meta	Contient les méta-données du rôle

Structure d'un rôle

Il est possible de créer un rôle vide contenant la structure arborescente.

```
# cd /etc/ansible/roles
# ansible-galaxy init common
- common was created successfully
```

La structure créée est la suivante :

```
# tree /etc/ansible/
/etc/ansible/
├── ansible.cfg
├── hosts
└── roles
    ├── common
    │   ├── defaults
    │   │   └── main.yml
    │   ├── files
    │   ├── handlers
    │   │   └── main.yml
    │   ├── meta
    │   │   └── main.yml
    │   ├── README.md
    │   ├── tasks
    │   │   └── main.yml
    │   ├── templates
    │   ├── tests
    │   │   ├── inventory
    │   │   └── test.yml
    │   └── vars
    │       └── main.yml
```

Pour supprimer un rôle :

```
# ansible-galaxy remove common
- successfully removed common
```

Un rôle est divisé en « sections ». Chaque section ayant une fonction précise.

Nom	Description
tasks	Contient la liste des tâches utilisées par le rôle
handlers	Contient la liste des handlers utilisés par le rôle
defaults	Contient les variables par défaut du rôle
vars	Contient les autres variables du rôle
files	Contient les fichiers utilisés via ce rôle (pour copy, ...)
templates	Contient les templates du rôle
meta	Contient les méta-données du rôle

Les rôles

Exécution d'un rôle

roles:

```
# more playbook.yml
- hosts: all
  roles:
    - exemple
```

include_role:
 name: nom_du_rôle

```
tasks:
- include_role:
    name: deploiement_apache
```

Exécution d'un rôle

Exécution par rôles :

L'exécution d'un playbook sollicitera les rôles qui sont définis via le mot clef **roles**.

```
# more playbook.yml
---
- name: Installation de mon_role1 et de mon_role2
  hosts: all
  roles:
    - mon_role1
    - mon_role2
...
```

Si plusieurs rôles sont définis, les règles suivantes s'appliquent :

- si roles/X/tasks/main.yml existe, les tâches listées dedans sont ajoutées au jeu de données.
- si roles/X/handlers/main.yml existe, les handlers listés dedans seront ajoutés au jeu.
- si roles/X/vars/main.yml existe, les variables listées dedans seront ajoutées au jeu.
- si roles/X/defaults/main.yml existe, les variables listées dedans seront ajoutés au jeu.
- si roles/X/meta/main.yml existe, chaque dépendance de rôle listée dedans est ajoutée.
- Chaque fichier, template ou tâche incluse dans le rôle, peut référencer des fichiers dans roles/X/{files,templates,tasks} sans avoir à les parcourir de manière relative ou absolue.

L'ordre d'exécution du playbook est le suivant:

- Toutes les pre_tasks définies dans le play
- Tous les handlers déclenchés seront exécutés

- Chaque rôle listé dans « roles : » sera exécuté à son tour. Toutes les dépendances de rôles définies dans le fichier meta/main.yml seront exécutées en premier sous réserve de conditions et de filtres.
- Toute task définie dans le play
- Tous les handlers déclenchés seront exécutés
- Toutes les post_tasks définies dans le play
- Tous les handlers déclenchés seront exécutés

Exécution par include role :

Il est possible d'intégrer l'exécution d'un rôle au sein de tâches via **include_role**.

Exemple1

```
# cat include_role_1.yml
---
- name: include role1
  hosts: all
  tasks:
    - name: Traitement un
      debug:
        msg: "Traitement UN pour le poste {{ inventory_hostname }}"
    - name: Inclure un rôle
      include_role:
        name: mon_role
    - name: Traitement deux
      debug:
        msg: "Traitement DEUX pour le poste {{ inventory_hostname }}"
...
```

Exemple2

```
# cat include_role_2.yml
---
- name: include role2
  hosts: all
  tasks:
    - name: Traitement un
      debug:
        msg: "Traitement UN pour le poste {{ inventory_hostname }}"

    - include_role:
        name: mon_role
      vars:
        ville: "Aix"
        pays: "France"
      when: ansible_distribution|lower == "centos"

    - name: Traitement deux
      debug:
        msg: "Traitement DEUX pour le poste {{ inventory_hostname }}"
...
```

Les rôles

Un exemple de rôle

```
# tree /etc/ansible/
/etc/ansible/
├── play1.yml
└── roles
    └── exemple
        ├── files
        │   └── httpd.conf
        ├── tasks
        │   └── main.yml
        └── vars
            └── main.yml
```

Un exemple de rôle

Voici un exemple simple de rôle.

```
# tree /etc/ansible/
/etc/ansible/
├── play1.yml
└── roles
    └── exemple
        ├── files
        │   └── httpd.conf
        ├── tasks
        │   └── main.yml
        └── vars
            └── main.yml
```

5 directories, 7 files

Le playbook play1.yml indique sur quels hôtes agir et quels rôles utiliser.

```
# more play1.yml
---
- name: Test du rôle exemple
  hosts: centos_servers
  roles:
    - exemple
...
```

Le répertoire roles contient la structure arborescente du rôle. Le sous répertoire tasks contient les tâches à effectuer.

Les tâches du rôle :

```
# more roles/exemple/tasks/main.yml
---
- name: Installer les outils de developpement
  yum:
    name: "@Development Tools"
    state: present

- name: Installer apache
  yum:
    name: httpd
    state: latest

- name: Copier le fichier de configuration d'apache
  copy:
    src: httpd.conf
    dest: /etc/httpd/conf/httpd.conf

- name: Redemarrer le service apache
  service:
    name: httpd
    state: restarted
...
```

A noter que le fichier contient la liste des tâches sans le mot clef « - tasks : ».

Sur le même principe, si nous avons eu des handlers, le fichier handlers/main.yml contiendrait la définition de chaque handler sans le mot clef « handlers : ».

La source du fichier à copier (httpd.conf) est un chemin relatif par rapport au rôle. Ansible va chercher le fichier à copier dans le répertoire files du rôle.

Les variables du rôle :

```
# more roles/exemple/vars/main.yml
---
vars:
  http_port: 80
  domain: mydomain.lan
...
```

L'exécution :

```
# ansible-playbook play1.yml
```


Les rôles

Un exemple de rôle avec des inclusions

```
# tree /etc/ansible/roles/exemple2/
/etc/ansible/roles/exemple2/
├── files
│   ├── apache2.conf
│   └── httpd.conf
├── tasks
│   ├── debian.yml
│   ├── main.yml
│   └── redhat.yml
└── vars
    └── main.yml
```

Un exemple de rôle avec des inclusions

Le playbook:

```
# more play2.yml
---
- name: exemple de role
  hosts: CentOS7.1,deb_server
  roles:
    - exemple2
...
```

L'arborescence du rôle:

```
# tree roles/exemple2
roles/exemple2
├── files
│   ├── apache2.conf
│   └── httpd.conf
├── tasks
│   ├── debian.yml
│   ├── main.yml
│   └── redhat.yml
└── vars
    └── main.yml

3 directories, 6 files
```

Le fichier tasks/main.yml fait référence aux fichiers debian.yml et redhat.yml avec une condition en fonction du type de l'OS.

Le rôle:

```
# more roles/exemple2/main.yml
---
- name: Installer et demarrer apache sur les serveurs Redhat
  include_tasks: redhat.yml
  when: ansible_os_family == 'RedHat'
- name: Installer et demarrer apache sur les serveurs Debian
  include_tasks: debian.yml
  when: ansible_os_family == 'Debian'
...
```

L'instruction `when` permet d'effectuer un appel conditionnel à un autre fichier indiqué par la directive `include_tasks`.

Les fichiers `redhat.yml` et `debian.yml` contiennent les mêmes instructions adaptés à l'OS:

```
# more redhat.yml
---
- name: Installer les outils de developpement
  yum:
    name: "@Development Tools"
    state: present

- name: Installer apache
  yum:
    name: httpd
    state: latest

- name: Copier le fichier de configuration d'apache
  copy:
    src: httpd.conf
    dest: /etc/httpd/conf/httpd.conf

- name: Redemarrer le service apache
  service:
    name: httpd
    state: restarted
...
```

```
# more debian.yml
---
- name: Installer les outils de developpement
  apt:
    name: build-essential
    state: present
    update_cache: yes

- name: Installer apache
  apt:
    name: apache2
    state: latest
    update_cache: yes

- name: Copier le fichier de configuration d'apache
  copy:
    src: apache2.conf
    dest: /etc/apache2/apache2.conf

- name: Redemarrer le service apache
  service:
    name: apache2
    state: restarted
...
```

L'exécution du playbook donne le résultat suivant:

```
# ansible-playbook play2.yml

PLAY [CentOS7.1,deb_server] *****

TASK [Gathering Facts] *****
ok: [deb_server]
ok: [CentOS7.1]

TASK [exemple2 : Installer et demarrer apache sur les serveurs Redhat] *****
skipping: [deb_server]
included: /etc/ansible/roles/exemple2/tasks/redhat.yml for CentOS7.1

TASK [exemple2 : Installer les outils de developpement] *****
ok: [CentOS7.1]

TASK [exemple2 : Installer apache] *****
ok: [CentOS7.1]

TASK [exemple2 : Copier le fichier de configuration d'apache] *****
ok: [CentOS7.1]

TASK [exemple2 : Redemarrer le service apache] *****
changed: [CentOS7.1]

TASK [exemple2 : Installer et demarrer apache sur les serveurs Debian] *****
skipping: [CentOS7.1]
included: /etc/ansible/roles/exemple2/tasks/debian.yml for deb_server

TASK [exemple2 : Installer les outils de developpement] *****
ok: [deb_server]

TASK [exemple2 : Installer apache] *****
ok: [deb_server]

TASK [exemple2 : Copier le fichier de configuration d'apache] *****
ok: [deb_server]

TASK [exemple2 : Redemarrer le service apache] *****
changed: [deb_server]

PLAY RECAP *****
CentOS7.1      : ok=6    changed=1    unreachable=0    failed=0
deb_server    : ok=6    changed=1    unreachable=0    failed=0
```

Le nom du rôle est affiché sur les lignes « TASK [nom_du_role : nom_de_la_tache] ».

Les rôles Ansible Galaxy

- Des rôles pré packagés disponibles
Zone de partage de la communauté Ansible
- <https://galaxy.ansible.com>
- La commande ansible-galaxy

Ansible Galaxy

Galaxy via son site (<https://galaxy.ansible.com>) met à disposition des rôles pré packagés. Ils sont facilement déployables au sein de votre projet Ansible. Vous trouverez des rôles pour l'infrastructure d'approvisionnement, le déploiement des applications et toutes vos tâches quotidiennes.

On peut également mettre ses propres rôles à la disposition de la communauté Ansible via ce site.

Recherche de rôles sur le site d'Ansible Galaxy:

[ansible-galaxy search](#)

Recherche des rôles crontab:

```
# ansible-galaxy search crontab
```

```
Found 83 roles matching your search:
```

Name	Description
----	-----
uZer.crontab	Crontab management
lciolecki.cron	Ansible role to manage crontab
viasite-ansible.cron	Add crontab tasks or variables
elao.cron	A cron role to manage crontab entries.
manala.cron	Handle cron
linuxhq.cronie	RHEL/CentOS - UNIX daemon crond (cronie)
igor_mukhin.cron	Installs and configures cron
...	

Informations sur un rôle:[ansible-galaxy info](#)

```
# ansible-galaxy info linuxhq.cronie

Role: linuxhq.cronie
  description: RHEL/CentOS - UNIX daemon crond (cronie)
  active: True
  commit: a68c03369737f17e77096e1639642c4400d6cfb9
  commit_message: Add become directive to crond handler
  commit_url: https://github.com/linuxhq/ansible-role-
cronie/commit/a68c03369737f17e77096e1639642c4400d6cfb9
  company:
  ...
```

Installation d'un rôle:[ansible-galaxy install](#)

```
# ansible-galaxy install linuxhq.cronie
- downloading role 'cronie', owned by linuxhq
- downloading role from https://github.com/.../ansible-role-cronie/archive/master.tar.gz
- extracting linuxhq.cronie to /root/.ansible/roles/linuxhq.cronie
- linuxhq.cronie (master) was installed successfully
```

A partir de `/root/.ansible/rôles/nom_du_rôle`, l'arborescence suivante est créée.

```
# ls -R /root/.ansible/roles/linuxhq.cronie/
/root/.ansible/roles/linuxhq.cronie/:
defaults handlers meta README.md tasks templates tests

/root/.ansible/roles/linuxhq.cronie/defaults:
main.yml

/root/.ansible/roles/linuxhq.cronie/handlers:
main.yml

/root/.ansible/roles/linuxhq.cronie/meta:
main.yml

/root/.ansible/roles/linuxhq.cronie/tasks:
cronie_etc.yml cronie_system.yml cronie_user.yml main.yml

/root/.ansible/roles/linuxhq.cronie/templates:
0hourly.j2 cron.allow.j2 cron.deny.j2 crond.sysconfig.j2

/root/.ansible/roles/linuxhq.cronie/tests:
inventory test.yml
```

Le fichier README.md contient un descriptif du rôle.

```
# more /root/.ansible/roles/linuxhq.cronie/README.md
# ansible-role-cronie

[![Build Status](https://travis-ci.org/linuxhq/ansible-role-cronie.svg?branch=master)]
(https://travis-ci.org/linuxhq/ansible-role-cronie)

RHEL/CentOS - UNIX daemon crond (cronie)

## Requirements

...

## Author Information

This role was created by [Taylor Kimball] (http://www.linuxhq.org).
```

Installation d'un rôle au sein d'un répertoire spécifique: `ansible-galaxy install --roles-path`

```
# mkdir -p galaxy/Roles

# ansible-galaxy install --roles-path galaxy/Roles linuxhq.cronie
- downloading role 'cronie', owned by linuxhq
- downloading role from https://github.com/.../ansible-role-cronie/archive/master.tar.gz
- extracting linuxhq.cronie to /root/Playbooks/galaxy/Roles/linuxhq.cronie
- linuxhq.cronie (master) was installed successfully
```

Création d'un rôle vierge:

`ansible-galaxy init`

```
# cd rep_roles
# ansible-galaxy init perso

# ls -R perso
perso:
defaults  files  handlers  meta  README.md  tasks  templates  tests  vars

perso/defaults:
main.yml

perso/files:

perso/handlers:
main.yml

perso/meta:
main.yml

perso/tasks:
main.yml

perso/templates:

perso/tests:
inventory  test.yml

perso/vars:
main.yml
```

Suppression d'un rôle:

`ansible-galaxy remove`

```
# ansible-galaxy remove perso
```

Gestion via un compte Ansible Galaxy

Pour cette section, vous devez au préalable créer un compte sur le site galaxy.ansible.com.

Pour se connecter:

```
# ansible-galaxy login
```

Une fois connecté, via la commande ci-dessus, les commandes suivantes sont utilisables.

Pour importer un rôle:

```
# ansible-galaxy import github_user github_repo
```

Pour supprimer un rôle:

```
# ansible-galaxy delete github_user github_repo
```

Le plus simple est de réaliser ces opérations et la gestion via son compte directement sur le site d'Ansible Galaxy.

Un exemple avec le site de **GitHub.com**:

GitHub est un espace pour gérer le versionning de ses projets et c'est un espace de développement collaboratif.

Après avoir créé un compte et déposé un rôle Ansible, on va pouvoir l'importer sur le site d'Ansible Galaxy.

Le fichier README.md du rôle apparaît comme présentation sur le site de GitHub. On pourra une attention toute particulière à son contenu.

Sur le site d'Ansible Galaxy, on crée un compte qui fera le lien avec le compte de GitHub. Il est ainsi aisé de scanner ses projets GitHub et de les transférer sur Galaxy. Attention si le fichier « meta/main.yml » de votre rôle est mal informé, l'import échoue.

Lorsque que le rôle est importé, il est exploitable via la commande « ansible-galaxy ».

il est à noter que « ansible-galaxy info le_role » affiche les informations formatées du fichier meta/main.yml suivi du contenu du fichier REAME.md.

Exemple :

```
# ansible-galaxy install barangerjeanmarc.site_lamp

# ansible-galaxy info barangerjeanmarc.site_lamp
perso:
Role: barangerjeanmarc.site_lamp
  description: Un role pour un site LAMP
  active: True
  commit: e0df31516368ce6f40056aa915415f4432cd655c
  commit_message: Add files via upload
  commit_url: https://github.com/barangerjeanmarc/site_lamp/commit/e0df31516368ce6
  company: Spharius
  created: 2018-05-01T19:03:19.484Z
  download_count: 1
  forks_count: 0
  github_branch:
  github_repo: site_lamp
  github_user: barangerjeanmarc
  id: 25427
  is_valid: True
  issue_tracker_url: https://github.com/barangerjeanmarc/site_lamp/issues
  license: license (GPLv2, CC-BY, etc)
  min_ansible_version: 1.2
  modified: 2018-05-01T19:41:26.608Z
  namespace: barangerjeanmarc
  open_issues_count: 0
  path: [u'/root/.ansible/roles', u'/usr/share/ansible/roles', u'/etc/ansible/role
  readme: site_lamp
=====

Un exemple simple de rôle.
...

Example Playbook
-----

- hosts: servers
  roles:
    - { role: barangerjeanmarc.site_lamp }
```

Notes

Fonctionnalités Avancées

Dans ce chapitre, nous allons approfondir la customisation des playbooks et des rôles.

Fonctionnalités Avancées

- | | |
|----------------------------------|-----------------------------------|
| • Les tags | Les blocks |
| • La visualisation d'un playbook | La connexion avec un autre compte |
| • Gather_facts | Le prompt |
| • La délégation par delegate_to | Le fichier d'inventaire dynamique |
| • Les pré et post tasks | et temporaire |
| • Le mot clef run_once | set_fact |
| • Le parallélisme | assert et fail |
| • Le traitement avec serial | lookup |
| • any_errors_fatal | dry-run, step-by-step et diff |

Fonctionnalités avancées

Les tags

```
# cat tags.yml
- name: "Tests des tags"
  hosts: client1
  tasks:
    - name: "Commande 1"
      shell:
        echo commande 1 >> /tmp/tags.res
      tags: ["c1"]
```

```
# ansible-playbook tags.yml --list-tags
```

```
# ansible-playbook tags.yml --tags c2
```

```
# ansible-playbook tags.yml --skip c1,c2
```

Les tags

On peut définir un tag sur une tâche ou directement sur un playbook.

Lors de l'exécution d'un playbook, on peut spécifier la liste des tags qu'il faut exclusivement exécuter ou au contraire ceux qu'ils ne faudra pas traiter.

Le mot clef est: tags: [«le_nom_du_tag»]

Le tag "always" est toujours traité.

```
# cat tags.yml
---
- name: Tests des tags
  hosts: centos1
  tasks:
    - name: Commande 1
      shell:
        echo commande 1 >> /tmp/tags.res
      tags: c1
    - name: Commande 2
      shell:
        echo commande 2 >> /tmp/tags.res
      tags: ["c2"]
    - name: Commande 1 et 2
      shell:
        echo commande 1 et 2 >> /tmp/tags.res
      tags: [c1, c2]
    - name: Commande 3
      shell:
        echo commande 3 >> /tmp/tags.res
    - name: Commande 4
      shell:
        date >> /tmp/tags.res
      tags: always
...
```

Pour lister les tags: `--list-tags`

```
# ansible-playbook tags.yml --list-tags

playbook: tags.yml

  play #1 (client1): Tests des tags      TAGS: []
    TASK TAGS: [always, c1, c2]
```

Exécution complète:

```
# ansible-playbook tags.yml
centos1# cat /tmp/tags.res
commande 1
commande 2
commande 1 et 2
commande 3
dim. déc. 30 06:08:13 CET 2018
```

Pour exécuter uniquement les tâches associées à des tags: `--tags`

```
# ansible-playbook tags.yml --tags c2
client1# cat /tmp/tags.res
commande 2
commande 1 et 2
dim. déc. 30 06:14:58 CET 2018
```

Pour exclure de l'exécution de tâches associées à des tags: `--skip`

```
# ansible-playbook tags.yml --skip c2
client1# cat /tmp/tags.res
commande 1
commande 3
dim. déc. 30 06:17:00 CET 2018
```

```
# ansible-playbook tags.yml --skip c1,c2
client1# cat /tmp/tags.res
commande 3
dim. déc. 30 06:18:09 CET 2018
```

Fonctionnalités avancées

La visualisation d'un playbook

```
# ansible-playbook --list-hosts play1.yml

# ansible-playbook --list-tasks play1.yml

# ansible-playbook --list-tags play1.yml
```

La visualisation d'un playbook

Lister les hôtes du playbook.

```
# ansible-playbook --list-hosts play1.yml

playbook: play1.yml

play #1 (centos_servers): centos_servers      TAGS: []
  pattern: [u'centos_servers']
  hosts (2):
    CentOS6.5
    CentOS7.1
```

Lister les tâches du playbook.

```
# ansible-playbook --list-tasks play1.yml

playbook: play1.yml

play #1 (centos_servers): centos_servers      TAGS: []
  tasks:
    exemple : Installer les outiles de developpement TAGS: []
    exemple : Installer apache                  TAGS: []
    exemple : Copier le fichier de configuration d'apache TAGS: []
    exemple : Redemarrer le service apache     TAGS: []
```

Lister les tags du playbook.

```
# ansible-playbook --list-tags play1.yml

playbook: play1.yml

play #1 (centos_servers): centos_servers      TAGS: []
  TASK TAGS: []
```

Fonctionnalités avancées

Gather_facts

Si gather_facts à no pas de récupération des facts

```
- hosts: all
  gather_facts: yes
  tasks:
    - debug:
        msg: La variable = {{ansible_distribution}}
```

Gather_facts

Les facts sont récupérées au début du traitement d'un playbook pour tous les hôtes. Elles sont nécessaires pour utiliser les variables Ansible au sein d'un playbook.

Il est possible de ne pas récupérer les facts via le mot clef gather_facts. Par exemple pour un parc important de machines et pour optimiser le temps d'exécution, lorsque ces variables sont inutiles au bon fonctionnement du playbook,

```
# cat variables_gatherfacts.yml
---
- name: Test de gather_facts
  hosts: centos1
  gather_facts: true
  tasks:
    - name: Afficher la distribution
      debug:
        msg: Distribution={{ansible_distribution}}
...
```

```
# ansible-playbook variables_gatherfacts.yml

PLAY [Test de gather_facts] *****

TASK [Gathering Facts] *****
ok: [centos1]

TASK [Afficher la distribution] *****
ok: [centos1] => {
  "msg": "Distribution=CentOS"
}

PLAY RECAP *****
centos1 : ok=2    changed=0    unreachable=0    failed=0
```

Le playbook avec `gather_facts` à `false`.

```
# cat variables_gatherfacts_false.yml
---
- name: Test de gather_facts
  hosts: centos1
  gather_facts: false
  tasks:
    - name: Afficher la distribution
      debug:
        msg: Distribution={{ansible_distribution}}
...

```

L'exécution du playbook échoue car la variable n'est pas définie puisque les facts n'ont pas été récupérés.

```
# ansible-playbook variables_gatherfacts_false.yml

PLAY [Test de gather_facts] *****

TASK [Afficher la distribution] *****
fatal: [centos1]: FAILED! => {"msg": "The task includes an option with an undefined
variable. The error was: 'ansible_distribution' is undefined\n\nThe error appears to have
been in '/root/variables_gatherfacts_no.yml': line 6, column 7, but may\nbe elsewhere in
the file depending on the exact syntax problem.\n\nThe offending line appears to be:\n\n
tasks:\n  - name: Afficher la distribution\n      ^ here\n"}
to retry, use: --limit @/root/variables_gatherfacts_no.retry

PLAY RECAP *****
centos1 : ok=0    changed=0    unreachable=0    failed=1

```

Fonctionnalités avancées

La délégation par `delegate_to`

`delegate_to` pour déporter le résultat d'une action d'un poste vers une autre machine

```
- hosts: all
  tasks:
    - name: Page d'informations
      template:
        src: delegate_to_modele.html
        dest: /var/www/html/{{inventory_hostname}}.html
        owner: apache
        group: apache
      delegate_to: client1
```

La délégation par `delegate_to`

La délégation par `delegate_to` permet de déporter le résultat d'une action d'un poste vers une autre machine.

L'exemple ci-dessous récupère sur le serveur Ansible (localhost) le résultat de la commande « `ls /etc/host*` » réalisée sur le poste `client1`.

```
# cat delegate_boucle.yml
---
- name: test de delegate_to
  hosts: centos1
  vars:
    equipe:
      - nom: jean
        uid: 1001
      - nom: marc
        uid: 1002
  tasks:
    - name: Recuperation du resultat d'execution
      shell: ls /etc/host*
      register: fic
    - name: Recuperation du resultat en local
      shell: echo "{{item}}" >> /tmp/boucle.res
      loop:
        "{{fic.stdout_lines}}"
      delegate_to: localhost
...
```


Exécution du playbook.

```
# ansible-playbook delegate_boucle.yml

PLAY [test de delegate_to] *****

TASK [Gathering Facts] *****
ok: [centos1]

TASK [Recuperation du resultat d'execution] *****
changed: [centos1]

TASK [Recuperation du resultat en local] *****
changed: [centos1 -> localhost] => (item=/etc/host.conf)
changed: [centos1 -> localhost] => (item=/etc/hostname)
changed: [centos1 -> localhost] => (item=/etc/hosts)
changed: [centos1 -> localhost] => (item=/etc/hosts.allow)
changed: [centos1 -> localhost] => (item=/etc/hosts.deny)

PLAY RECAP
*****
centos1 : ok=3    changed=2    unreachable=0    failed=0
```

Le fichier a bien été créé en local.

```
# cat /tmp/boucle.res
/etc/host.conf
/etc/hostname
/etc/hosts
/etc/hosts.allow
/etc/hosts.deny
```

Autre exemple

Le poste client1 est un serveur Apache. Le principe est de récupérer sur le serveur Apache l'ensemble de la configuration des machines du parc. Chaque machine aura une page html portant son nom.

Le playbook

```
# cat delegate_to.yml
---
- name: Recuperation du resultat sur client1
  hosts: all
  tasks:
    - name: Page d'informations
      template:
        src: delegate_to_modele.html
        dest: /var/www/html/{{inventory_hostname}}.html
        owner: apache
        group: apache
        delegate_to: client1
...

```

Le fichier template à copier.

```
# cat delegate_to_modele.html
<html>
  <head><title>Page de garde</title></head>
  <body>
    <br><br><hr><br>
    <center><h1>Bonjour {{ansible_user_id}}</h1></center>
    <br><hr>
    <center><b>
La liste des adresses IP :<br>
{% for element in ansible_all_ipv4_addresses %}
  {{element}}<br>
{% endfor %}
<br>
La distribution est : {{ansible_distribution}}
<br><br>
    </b></center>
  </body>
</html>
```

Exécution du playbook

```
# ansible-playbook delegate_to.yml

PLAY [Recuperation du resultat sur client1] *****

TASK [Gathering Facts] *****
ok: [debian1]
ok: [client1]
ok: [client2]

TASK [Page d'informations] *****
changed: [client1 -> client1]
changed: [debian1 -> client1]
changed: [client2 -> client1]

PLAY RECAP *****
client1      : ok=2    changed=1    unreachable=0    failed=0
client2      : ok=2    changed=1    unreachable=0    failed=0
debian1      : ok=2    changed=1    unreachable=0    failed=0
```

Le résultat sur la machine client1 est le suivant :



Bonjour root

La liste des adresses IP :
192.168.1.8

La distribution est : CentOS



Bonjour root

**La liste des adresses IP :
192.168.1.9**

La distribution est : CentOS



Bonjour root

**La liste des adresses IP :
192.168.1.25**

La distribution est : Debian

Fonctionnalités avancées

Les pré et post tasks

```
# cat pre_post_tasks.yml
- hosts: client1
  vars:
    mavar: Jean
  pre_tasks:
    - debug:
        msg: Pre tacheA pour {{mavar}}
    - debug:
        msg: Pre tacheB pour {{mavar}}
  post_tasks:
    - debug:
        msg: Post tacheA pour {{mavar}}
    - debug:
        msg: Post tacheB pour {{mavar}}
  tasks:
    - debug:
        msg: tache1 pour {{mavar}} debut
    - debug:
        msg: tache2 pour {{mavar}} debut
```

```
# ansible-playbook pre_post_tasks.yml
```

Les pré et post tasks

Il est possible de définir des opérations avant le traitement principal, c'est la section `pre_tasks`. De même, il est possible de définir des opérations après le traitement principal, c'est la section `post_tasks`.

```
# cat pre_post_tasks.yml
---
- name: test des pré et post tasks
  hosts: centos1
  vars:
    mavar: Martin
  pre_tasks:
    - name: Pre tacheA pour {{inventory_hostname}}
      debug:
        msg: Pre tacheA pour {{mavar}}
    - name: Pre tacheB pour {{inventory_hostname}}
      debug:
        msg: Pre tacheB pour {{mavar}}
  post_tasks:
    - name: Post tacheA pour {{inventory_hostname}}
      debug:
        msg: Post tacheA pour {{mavar}}
    - name: Post tacheB pour {{inventory_hostname}}
      debug:
        msg: Post tacheB pour {{mavar}}
  tasks:
    - name: tache1 pour {{inventory_hostname}}
      debug:
        msg: tache1 pour {{mavar}} debut
    - name: tache2 pour {{inventory_hostname}}
      debug:
        msg: tache2 pour {{mavar}} debut
...
```

Exécution du playbook.

```
# ansible-playbook pre_post_tasks.yml

PLAY [test des pré et post tasks] *****

TASK [Gathering Facts] *****
ok: [centos1]

TASK [Pre tacheA pour centos1] *****
ok: [centos1] => {                                     section pré-tasks
    "msg": "Pre tacheA pour Martin"
}

TASK [Pre tacheB pour centos1] *****
ok: [centos1] => {                                     section pré-tasks
    "msg": "Pre tacheB pour Martin"
}

TASK [tache1 pour centos1] *****
ok: [centos1] => {                                     section tasks
    "msg": "tache1 pour Martin debut"
}

TASK [tache2 pour centos1] *****
ok: [centos1] => {                                     section tasks
    "msg": "tache2 pour Martin debut"
}

TASK [Post tacheA pour centos1] *****
ok: [centos1] => {                                     section post-tasks
    "msg": "Post tacheA pour Martin"
}

TASK [Post tacheB pour centos1] *****
ok: [centos1] => {                                     section post-tasks
    "msg": "Post tacheB pour Martin"
}

PLAY RECAP *****
centos1                : ok=7    changed=0    unreachable=0    failed=0
```

Fonctionnalités avancées

Le mot clef run_once

```
# cat runonce.yml
- name: Test runonce
  hosts: all
  tasks:
    - name: Redemarrage du service crond
      service:
        name: crond
        state: restarted
      run_once: yes
    - name:
      shell:
        date > /tmp/etat
```

Le mot clef run_once

le mot clef run_once permet d'exécuter une tâche qu'une seule fois.

```
# cat runonce.yml
---
- name: Test runonce
  hosts: all
  tasks:
    - name: Redemarrage du service crond
      service:
        name: crond
        state: restarted
      run_once: yes
    - name:
      shell:
        date > /tmp/etat
...
# ansible-playbook runonce.yml

PLAY [Test runonce] *****

TASK [Gathering Facts] *****
ok: [client2]
ok: [client1]

TASK [Redemarrage d un service] *****
changed: [client1]

TASK [shell] *****
changed: [client2]
changed: [client1]

PLAY RECAP *****
client1          : ok=3    changed=2    unreachable=0    failed=0
client2          : ok=2    changed=1    unreachable=0    failed=0
```

On constate que le service n'a été redémarré que sur un seul serveur (client1).

Fonctionnalités avancées

Le parallélisme

```
# grep forks /etc/ansible/ansible.cfg
#forks                = 5
```

```
# ansible spherius_servers xxxxxxxxx -f 20
```

Le parallélisme

Le paramètre forks du fichier de configuration permet de contrôler le nombre de processus qui peuvent être exécutés en simultané. Par défaut, il vaut 5.

```
# grep forks /etc/ansible/ansible.cfg
#forks                = 5
```

L'option -f de la commande ansible permet d'écraser la valeur par défaut spécifié dans le fichier.

Les tâches s'exécutent en simultané sur les deux machines.

```
# ansible all -m shell -a "date; sleep 5; date"
client1 | SUCCESS | rc=0 >>
lun. avril 23 11:34:34 CEST 2018
lun. avril 23 11:34:39 CEST 2018

client2 | SUCCESS | rc=0 >>
lun. avril 23 11:34:34 CEST 2018
lun. avril 23 11:34:39 CEST 2018
```

Les tâches sont bien exécutées une par une.

```
# ansible all -m shell -a "date; sleep 5; date" -f 1
client1 | SUCCESS | rc=0 >>
lun. avril 23 11:35:38 CEST 2018
lun. avril 23 11:35:43 CEST 2018

client2 | SUCCESS | rc=0 >>
lun. avril 23 11:35:44 CEST 2018
lun. avril 23 11:35:49 CEST 2018
```

Fonctionnalités avancées

Le traitement avec serial

serial: 3

3 serveurs sont traités. Puis lorsque c'est terminé, c'est les 3 suivants et ainsi de suite.

serial: «20%»

comme précédemment, mais par séquence de 20% des serveurs.

serial: [1, 5, 10]

un serveur est traité, puis 5, puis par séquence de 10.

serial: [2, «100%»]

deux serveurs sont traités, puis tous les autres.

Le traitement avec serial

Lorsque vous avez 10 serveurs Apache, vous souhaitez certainement une continuité de service lors d'une mise à jour de vos serveurs. On peut envisager la mise à jour sur 2 serveurs, puis lorsque cela sera fait passer aux autres serveurs. Pour cette opération, on utilisera le mot clef serial positionnée à la valeur [2, «100%»].

Syntaxe:**serial: 3**

3 serveurs sont traités. Puis lorsque c'est terminé, c'est les 3 suivants et ainsi de suite.

serial: «20%»

comme précédemment, mais par séquence de 20% des serveurs.

serial: [1, 5, 10]

un serveur est traité, puis 5, puis par séquence de 10.

serial: [2, «100%»]

deux serveurs sont traités, puis tous les autres.

Ce type de traitement peut-être utile pour assurer une continuité de services, ou pour éviter une montée en charge de la consommation des ressources telle que l'utilisation de la bande passante lors de transferts réseaux.

Exemple:

```
# cat serial.yml
---
- name: test de serial
  hosts: all
  serial: [1, 100%]
  tasks:
    - name: Execution de la commande date à 5 secondes d'intervalle
      shell: "date; sleep 5; date"
...
```

Il y aura le traitement sur UN poste (ici client1), on constate la temporisation de 5 secondes. Puis la même tâche se réalise sur tous les autres serveurs.

Exécution du playbook.

```
# ansible-playbook serial.yml

PLAY [test de serial] *****

TASK [Gathering Facts] *****
ok: [centos1]                  les facts d'une seule machine sont récupérés

TASK [Execution de la commande date à 5 secondes d'intervalle] *****
changed: [centos1]             après 5 secondes cette ligne apparaît

PLAY [test de serial] *****

TASK [Gathering Facts] *****
ok: [debian1]                  les machines restantes sont ensuite traitées
ok: [centos2]

TASK [Execution de la commande date à 5 secondes d'intervalle] *****
changed: [debian1]
changed: [centos2]

PLAY RECAP *****
centos1      : ok=2    changed=1    unreachable=0    failed=0
centos2      : ok=2    changed=1    unreachable=0    failed=0
debian1      : ok=2    changed=1    unreachable=0    failed=0
```

Le play est d'abord exécuté sur une machine puis sur les autres machines.

Fonctionnalités avancées

any_errors_fatal

En cas d'erreur d'exécution d'une task sur un hôte, any_errors_fatal permet d'arrêter (ou pas) les tasks sur l'ensemble des serveurs.

```
# cat any_errors_fatal.yml
- hosts: all
  any_errors_fatal: true
  tasks:
    - name:
```

```
TASK [Operation a risque]
*****
fatal: [debian1]: FAILED! => . . .
changed: [client2]
changed: [client1]

NO MORE HOSTS LEFT
*****
to retry, use: --limit @/root/Playbooks/any_errors_fatal.retry
```

any_errors_fatal

En cas d'erreur d'exécution d'une task sur un hôte, any_errors_fatal permet d'arrêter (ou pas) les tasks sur l'ensemble des serveurs.

Exemple avec any_errors_fatal: true

```
# cat any_errors_fatal_true.yml
---
- name: test d'any_errors_fatal
  hosts: all
  any_errors_fatal: true
  tasks:
    - name: Operation a risque
      command: ls /etc/httpd
    - name: Operation suivante
      command: date
...
```

La première opération fonctionne pour les serveurs centos mais pas sur le serveur debian (car il aurait fallu indiquer /etc/apache2). La deuxième tâche ne sera pas exécutée et ceci sur aucun serveur.

```
# ansible-playbook any_errors_fatal_true.yml

PLAY [test d'any_errors_fatal] *****

TASK [Gathering Facts] *****
ok: [debian1]
ok: [client2]
ok: [client1]
```

```
TASK [Operation a risque] *****
fatal: [debian1]: FAILED! => {"changed": true, "cmd": "ls /etc/httpd", "delta":
"0:00:00.002270", "end": "2018-04-27 16:40:58.690715", "msg": "non-zero return code",
"rc": 2, "start": "2018-04-27 16:40:58.688445", "stderr": "ls: impossible d'accéder à
'/etc/httpd': Aucun fichier ou dossier de ce type", "stderr_lines": ["ls: impossible
d'accéder à '/etc/httpd': Aucun fichier ou dossier de ce type"], "stdout": "",
"stdout_lines": []}
changed: [client2]
changed: [client1]

NO MORE HOSTS LEFT *****
to retry, use: --limit @/root/Playbooks/any_errors_fatal.retry

PLAY RECAP *****
client1      : ok=2    changed=1    unreachable=0    failed=0
client2      : ok=2    changed=1    unreachable=0    failed=0
debian1      : ok=1    changed=0    unreachable=0    failed=1
```

Exemple avec any_errors_fatal: false

```
# cat any_errors_fatal_false.yml
- name: test d'any_errors_fatal
  hosts: all
  any_errors_fatal: false
  tasks:
    - name: Operation a risque
      command: ls /etc/httpd
    - name: Operation suivante
      command: date
```

Comme précédemment, la première opération fonctionne pour les serveurs centos mais pas sur le serveur debian.

La deuxième tâche ne sera pas exécutée sur le serveur débien MAIS elle sera exécutée sur les deux autres serveurs centos.

```
# ansible-playbook any_errors_fatal_false.yml

PLAY [test d'any_errors_fatal] *****

TASK [Gathering Facts] *****
ok: [debian1]
ok: [client2]
ok: [client1]

TASK [Operation a risque] *****
fatal: [debian1]: FAILED! => {"changed": true, "cmd": "ls /etc/httpd", "delta":
"0:00:00.002256", "end": "2018-04-27 16:44:33.404066", "msg": "non-zero return code",
"rc": 2, "start": "2018-04-27 16:44:33.401810", "stderr": "ls: impossible d'accéder à
'/etc/httpd': Aucun fichier ou dossier de ce type", "stderr_lines": ["ls: impossible
d'accéder à '/etc/httpd': Aucun fichier ou dossier de ce type"], "stdout": "",
"stdout_lines": []}
changed: [client1]
changed: [client2]

TASK [Operation suivante] *****
changed: [client1]
changed: [client2]
to retry, use: --limit @/root/Playbooks/any_errors_fatal.retry

PLAY RECAP *****
client1      : ok=3    changed=2    unreachable=0    failed=0
client2      : ok=3    changed=2    unreachable=0    failed=0
debian1      : ok=1    changed=0    unreachable=0    failed=1
```

Fonctionnalités avancées

Les blocks

```
- hosts: all
  tasks:
    - name: Install Apache
      block:
        - yum:
            name: "{{ item }}"
            state: installed
          with_items:
            - httpd
            - memcached
        - template:
            src: templates/src.j2
            dest: /etc/foo.conf
        - service:
            name: bar
            state: started
            enabled: True
      when: ansible_distribution == 'CentOS'
```

Le block RESCUE

Le block ALWAYS

Les blocks

La section **block** regroupe un ensemble de tasks qui peuvent être associée à une même condition, ou liste, etc.

La section **rescue** est traitée lorsqu'il y a eut une anomalie au sein du block précédent.

La section **always** est traitée dans tous les cas de figures.

Exemple:

```
# cat block.yml
- name: Test de block
  hosts: all
  tasks:
    - block:
        - name: Installation des packages
          yum:
            name:
              - httpd
              - memcached
            state: present
        - name: Copie du fichier template
          template:
            src: templates/src.j2
            dest: /etc/foo.conf
        - name: Demarrage du service httpd
          service:
            name: httpd
            state: started
            enabled: true
      when: ansible_distribution == 'CentOS'
  ...
```

Exécution du playbook.

```
# ansible-playbook block.yml
```

```
PLAY [Test de block] *****

TASK [Gathering Facts] *****
ok: [debian1]
ok: [centos1]
ok: [centos2]

TASK [Installation des packages] *****
skipping: [debian1]
ok: [centos1]
ok: [centos2]

TASK [Copie du fichier template] *****
skipping: [debian1]
changed: [centos2]
changed: [centos1]

TASK [Redemarrage du service httpd] *****
skipping: [debian1]
changed: [centos1]
changed: [centos2]

PLAY RECAP *****
centos1           : ok=4    changed=2    unreachable=0    failed=0
centos2           : ok=4    changed=2    unreachable=0    failed=0
debian1           : ok=1    changed=0    unreachable=0    failed=0
```

Exemple avec des sections rescue et always:

```
# cat block2.yml
---
- name: Tests des blocs rescue et always
  hosts: centos1
  tasks:
    - name: Les blocs rescue et always
      block:
        - name: Affichage du message
          debug:
            msg: "Je m'execute normalement"
        - name: Exécution d'une commande
          command: "ls {{argument}}"
        - name: Affichage du message
          debug:
            msg: "Si je m'execute c'est que la commande precedente a fonctionne"
      rescue:
        - name: Affichage du message d'erreur
          debug:
            msg: "Il y a une tasks en erreur"
        - name: Execution de la commande /bin/false
          command: /bin/false
        - name: Après une erreur dans le block rescue
          debug:
            msg: "Je ne serais jamais executee"
      always:
        - name: Affichage du message permanent
          debug:
            msg: "Je m'execute toujours"
...

```

Bonne exécution du playbook.

```
# ansible-playbook block2.yml -e argument=/etc

PLAY [Tests des blocs rescue et always] *****

TASK [Gathering Facts] *****
ok: [centos1]

TASK [Affichage du message] *****
ok: [centos1] => {                                     Au sein du block
    "msg": "Je m'execute normalement"
}

TASK [Exécution d'une commande] *****
changed: [centos1]                                     Au sein du block

TASK [Affichage du message en cas de réussite] *****
ok: [centos1] => {                                     Au sein du block
    "msg": "Si je m'execute c'est que la commande precedente a fonctionne"
}

TASK [Affichage du message permanent] *****
ok: [centos1] => {                                     Block ALWAYS
    "msg": "Je m'execute toujours"
}

PLAY RECAP *****
centos1 : ok=5    changed=1    unreachable=0    failed=0
```

Mauvaise exécution du playbook.

```
# ansible-playbook block2.yml -e argument=/xxx

PLAY [Tests des blocs rescue et always] *****

TASK [Gathering Facts] *****
ok: [centos1]

TASK [Affichage du message] *****
ok: [centos1] => {                                     Au sein du block
    "msg": "Je m'execute normalement"
}

TASK [Exécution d'une commande] *****
fatal: [centos1]: FAILED! => {"changed": true, "cmd": ["ls", "/xxx"], "delta":
"0:00:00.010622", "end": "2018-12-30 13:03:30.567058", "msg": "non-zero return code",
"rc": 2, "start": "2018-12-30 13:03:30.556436", "stderr": "ls: impossible d'accéder à
/xxx: Aucun fichier ou dossier de ce type", "stderr_lines": ["ls: impossible d'accéder
à /xxx: Aucun fichier ou dossier de ce type"], "stdout": "", "stdout_lines": []}

TASK [Affichage du message d'erreur] *****
ok: [centos1] => {                                     Block RESCUE
    "msg": "Il y a une tasks en erreur"
}

TASK [Execution de la commande /bin/false] *****
fatal: [centos1]: FAILED! => {"changed": true, "cmd": ["/bin/false"], "delta":
"0:00:00.008710", "end": "2018-12-30 13:03:33.566982", "msg": "non-zero return code",
"rc": 1, "start": "2018-12-30 13:03:33.558272", "stderr": "", "stderr_lines": [],
"stdout": "", "stdout_lines": []}

TASK [Affichage du message permanent] *****
ok: [centos1] => {                                     Block ALWAYS
    "msg": "Je m'execute toujours"
}

    to retry, use: --limit @/root/block2.retry

PLAY RECAP *****
centos1 : ok=4    changed=0    unreachable=0    failed=2
```

Fonctionnalités avancées

La connexion avec un autre compte

```
# ssh-copy-id -i /root/.ssh/id_rsa.pub user1@client1

# cat user1_client.inv
client1      ansible_user=user1

# ansible all -m lineinfile \
-a "path=/etc/sudoers line='user1 ALL=(ALL:ALL) NOPASSWD: ALL'"

# ansible all -i user1_client -m service \
-a "name=cron state=restarted" --become
```

La connexion avec un autre compte

La connexion aux postes clients peut être réalisée avec un compte utilisateur autre que root.

Pour cela, il est nécessaire de transférer la clef SSH à l'utilisateur du hôte client.

Le compte à utiliser pour les commandes Ansible, playbooks et rôles est indiqué par `ansible_user`.

Si les tâches à exécuter nécessitent les droits d'administration, il faut mettre à jour les fonctionnalités de sudo.

Exemple:

Sur le poste client, la connexion doit se faire via le compte user1.

Sur le poste serveur Ansible, les actions sont exécutées avec le compte root.

Copie de la clef SSH :

```
# ssh-copy-id -i /root/.ssh/id_rsa.pub user1@client1
```

Mise à jour de la variable `ansible_user` (par exemple au sein du fichier d'inventaire).

```
# cat user1_client.inv
client1      ansible_user=user1
```

le module `setup` récupère les informations du poste client.

```
# ansible -i user1_client.inv -m setup all
```

`ansible_user_id` : l'utilisateur pour la connexion Ansible.

`ansible_user_uid` : l'uid de l'utilisateur pour la connexion Ansible.

Via user1, il est possible d'exécuter un panel d'actions qui ne nécessite pas les droits de root. Par exemple, on peut utiliser ping mais on ne peut pas redémarrer un service.

S'il est nécessaire de traiter des tâches nécessitant les droits d'administration, il faut mettre à jour le fichier /etc/sudoers pour l'utilisateur user1.

Mettre à jour le fichier /etc/sudoers sur les clients :

```
CLIENT1# visudo
user1 ALL=(ALL:ALL) NOPASSWD: ALL
```

Par la suite, l'exécution doit se faire avec l'option --become.

```
# ansible all -i user1_client.inv -m service \
-a "name=cron state=restarted" --become
```

Utilisation avec un playbook :

```
# cat playbook.yml
- hosts: client1
  remote_user: user1
  become: yes
  tasks:
    - manip: via root
      arg1: xxx
      arg2: xxx
    - manip: via root
      arg1: xxx
      arg2: xxx
```

Autre exemple :

```
# cat playbook.yml
- hosts: client1
  remote_user: user1
  tasks:
    - manip: via user1
      arg1: xxx
      arg2: xxx
    - manip: via root
      arg1: xxx
      arg2: xxx
      become: yes
    - manip: via apache
      arg1: xxx
      arg2: xxx
      become: yes
      become_user: apache
```


Fonctionnalités avancées

Le prompt

```
- name: nom_table
  prompt: Saisir le nom de la table
  default: table_base
  private: no
```

```
- name:      Nom de la variable
  prompt:    Libellé du prompt
  default:   optionnel: Une valeur par défaut
  private:   optionnel: no pour ne pas masquer la saisie
```

Le prompt

Le prompt apporte l'interactivité lors de l'exécution d'un playbook.

L'utilisateur doit saisir du texte qui initialise une variable.

```
# cat prompt.yml
---
- name: test de prompt
  hosts: centos_servers
  vars_prompt:
    - name: nom_table                # nom de la variable
      prompt: Saisir le nom de la table # libellé du prompt
      default: table_base              # optionnel: valeur par défaut
      private: false                  # optionnel: ne pas masquer la saisie
    - name: nom_champ
      prompt: Saisir le nom du champ
      private: false
  tasks:
    - name: Afficher les résultats
      debug:
        msg: select {{nom_champ}} from {{nom_table}}
...

```

```
# ansible-playbook prompt.yml
Saisir le nom de la table [table_base]: pays
Saisir le nom du champs: ville

TASK [Afficher les résultats] *****
ok: [centos1] => {
  "msg": "select ville from pays"
}
ok: [centos2] => {
  "msg": "select ville from pays"
}

```

Fonctionnalités avancées

Le fichier d'inventaire dynamique et temporaire

```
- name: "Un inventaire temporaire"
  hosts: all
  gather_facts: yes
  tasks:
    - name: "Creation des groupes de hotes hote_distribution"
      group_by: key="hote_{{ansible_distribution}}"

- name: "Traitement pour les CentOS"
  hosts: "hote_CentOS"
  gather_facts: no
  tasks: . . . Tasks pour CentOS

- name: "Traitement pour les Debian"
  hosts: "hote_Debian"
  gather_facts: no
  tasks: . . . Tasks pour Debian"
```

Le fichier d'inventaire dynamique et temporaire

Le fichier d'inventaire dynamique

L'inventaire est généré dynamiquement au moment où l'on exécute la commande ansible (ou un playbook, etc). En fait, cela passe par un script en général écrit en Python, mais cela peut être du script shell, du PHP ou autre.

La plupart des scripts existent déjà (pour AWS, VMware, Docker, Cobber, etc) :

http://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html

Pour développer son script :

http://docs.ansible.com/ansible/latest/dev_guide/developing_inventory.html

Le script doit générer un fichier d'inventaire au format JSON et respecter quelques règles de mise en page, gérer les groupes all et ungrouped ou avoir les options --list et --host.

Travailler avec une base de données :

Si la liste des machines est localisée au sein d'une base MySQL, on peut exploiter cette liste pour générer un inventaire dynamique. Pour cela, un script est disponible sur le site de github :

<https://github.com/productsupcom/ansible-dyninv-mysql>

Pour travailler avec Azure :

Afin de générer une liste dynamique à partir de la liste des machines enregistrées au sein d'Azure, on peut exploiter un script (azure_rm.py) :

```
# wget
https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/azure_rm.py

# chmod 755 azure_rm.py

# ansible -i azure_rm.py ansible-inventory-test-rg -m ping
```

Le fichier d'inventaire temporaire

Lorsque la liste des machines de votre parc évolue régulièrement, il n'est pas pratique de mettre à jour constamment son fichier d'inventaire. L'inventaire temporaire permet de créer un inventaire directement au sein du playbook. Il est donc généré au moment de l'exécution du playbook ou du rôle, et n'a qu'une existence temporaire le temps de l'exécution.

Le parc des hôtes est référencé au sein du fichier hosts. Des groupes de machines sont générés lors de l'exécution du playbook.

Exemple:

L'exemple suivant est un playbook qui va créer des groupes de hôtes et des groupes enfants en fonction de critères de chaque machine du parc: la distribution du système d'exploitation et sa version.

La première étape est de créer ces groupes :

« gather_facts » est à true pour récupérer les facts des machines du parc.

Ceci afin d'exploiter les caractéristiques de chaque machine pour les associer à différents groupes.

« group by » permet de regrouper des machines par groupe en fonction de critères (clef).

Les étapes suivantes :

Les tasks qui suivent pourront exploiter ces groupes pour réaliser des actions spécifiques.

« gather_facts » est à false car les tâches qui suivent n'ont pas besoin d'exploiter les facts. On obtient un gain de temps sur l'exécution du playbook.

Le playbook.

```
# cat temporaire.yml
---
- name: Un inventaire temporaire
  hosts: all
  gather_facts: true
  tasks:
    - name: Creation des groupes de hotes hote_distribution
      group_by: key=hote_{{ansible_distribution}}
    - name: Creation avec parents host_major-version_architecture
      group_by:
        key: host_{{ansible_distribution_major_version}}_{{ansible_architecture}}
        parents: host_{{ansible_distribution_major_version}}
    - name: Creation avec parents host_major-version_distribution
      group_by:
        key: host_{{ansible_distribution_major_version}}_{{ansible_distribution}}
        parents: host_{{ansible_distribution_major_version}}

- name: Traitement pour les CentOS
  hosts: hote_CentOS
  gather_facts: false
  tasks:
    - name: Action pour ce groupe
      debug:
        msg: Action pour CentOS

- name: Traitement pour les Debian
  hosts: hote_Debian
  gather_facts: false
  tasks:
    - name: Action pour ce groupe
      debug:
        msg: Action pour Debian

- name: Traitement pour les versions 7 architecture x86_64
  hosts: host_7_x86_64
  gather_facts: false
  tasks:
    - name: Action pour ce groupe
      debug:
        msg: Action pour CentOS version 7

- name: Traitement pour les 7
  hosts: host_7
  gather_facts: false
  tasks:
    - name: Action pour ce groupe
      debug:
        msg: Action pour les versions 7

- name: Traitement pour les versions 7 distribution CentOS
  hosts: host_7_CentOS
  gather_facts: false
  tasks:
    - name: Action pour ce groupe
      debug:
        msg: Action pour les versions 7 distribution CentOS
```

```
# ansible-playbook temporaire.yml
PLAY [Un inventaire temporaire] *****

TASK [Gathering Facts] *****
ok: [debian1]
ok: [centos2]
ok: [centos1]

TASK [Creation des groupes de hotes hote_distribution] *****
ok: [centos1]
ok: [centos2]
ok: [debian1]

TASK [Creation avec parents host_major-version_architecture] *****
ok: [debian1]
ok: [centos1]
ok: [centos2]

TASK [Creation avec parents host_major-version_distribution] *****
ok: [debian1]
ok: [centos1]
ok: [centos2]

PLAY [Traitement pour les CentOS] *****

TASK [Action pour ce groupe] *****
ok: [centos1] => {
    "msg": "Action pour CentOS"
}
ok: [centos2] => {
    "msg": "Action pour CentOS"
}

PLAY [Traitement pour les Debian] *****

TASK [Action pour ce groupe] *****
ok: [debian1] => {
    "msg": "Action pour Debian"
}

PLAY [Traitement pour les versions 7 architecture x86_64] *****

TASK [Action pour ce groupe] *****
ok: [centos1] => {
    "msg": "Action pour CentOS version 7"
}
ok: [centos2] => {
    "msg": "Action pour CentOS version 7"
}

PLAY [Traitement pour les 7] *****

TASK [Action pour ce groupe] *****
ok: [centos1] => {
    "msg": "Action pour les versions 7"
}
ok: [centos2] => {
    "msg": "Action pour les versions 7"
}

PLAY [Traitement pour les versions 7 distribution CentOS] *****

TASK [Action pour ce groupe] *****
ok: [centos1] => {
    "msg": "Action pour les versions 7 distribution CentOS"
}
ok: [centos2] => {
    "msg": "Action pour les versions 7 distribution CentOS"
}

PLAY RECAP *****
centos1          : ok=8    changed=0    unreachable=0    failed=0
centos2          : ok=8    changed=0    unreachable=0    failed=0
debian1          : ok=5    changed=0    unreachable=0    failed=0
```

Fonctionnalités avancées

lookup

```
lookup( 'pipe' , 'date +%A' )

lookup( 'env' , 'HOSTNAME' )

lookup('vars', 'nom_' + mavar)

lookup('vars', 'nom_' + mavar, default='variable non existante')

lookup( 'file' , '/etc/motd' )

lookup( 'file' , '/tmp/fic1', errors='ignore' )
```

lookup

Les plugins lookup permettent à Ansible d'accéder à des données provenant de sources externes. Cela peut inclure la consultation d'un fichier, l'exécution d'une commande ou l'exploitation d'une variable système.

Ces plugins sont évalués sur le serveur Ansible, et non sur le client.

lookup('pipe', ...) permet de récupérer le résultat d'une commande.

```
# cat lookup_pipe.yml
---
- name: Test de lookup('pipe')
  hosts: client1
  vars:
    jour: "{{ lookup('pipe','date +%A') }}"          # %A pour : lundi, mardi, ...
  tasks:
    - name: Afficher la variable jour
      debug:
        msg: "On est le {{ jour }}"
...
```

Extrait du résultat d'exécution :

```
# ansible-playbook lookup_pipe.yml

TASK [Afficher la variable jour] *****
ok: [client1] => {
  "msg": "On est le mardi"
}
```

lookup('env', ...) permet de récupérer la valeur d'une variable d'environnement.

```
# cat lookup_env.yml
---
- name: Test de lookup('env')
  hosts: client1
  vars:
    machine: "{{ lookup( 'env' , 'HOSTNAME' ) }}"
  tasks:
    - name: Afficher la variable
      debug:
        msg: "la variable est {{ machine }}"
...
```

Extrait du résultat d'exécution :

```
# ansible-playbook lookup_pipe.yml

TASK [Afficher la variable] *****
ok: [client1] => {
  "msg": "la variable est ansible_server"
}
```

lookup('vars', ...) permet de manipuler des variables.

```
# cat lookup_var.yml
- name: test de lookup
  hosts: localhost
  vars:
    nom_variable: formidable
    mavar: variable
  tasks:
    - name: Concaténation de variables
      debug:
        msg: "{{ lookup('vars', 'nom_' + mavar) }}"
```

nom_ est concaténé avec le contenu de mavar.

```
# ansible-playbook lookup_var.yml

PLAY [test de lookup] *****

TASK [Gathering Facts]
*****
ok: [localhost]

TASK [Concaténation de variables] *****
ok: [localhost] => {
  "msg": "formidable"
}

PLAY RECAP *****
localhost : ok=2    changed=0    unreachable=0    failed=0
```

Exemple avec un message par défaut si la variable n'existe pas :

```
# cat lookup2_var.yml
---
- name: test de lookup
  hosts: localhost
  tasks:
    - name: Concaténation de variables
      debug:
        msg: "{{ lookup('vars', 'nom_' + mavar, default='variable non existante') }}"
  vars:
    nom_variable: formidable
    mavar: var
...
```

```
# ansible-playbook lookup2_var.yml

PLAY [test de lookup] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Concaténation de variables] *****
ok: [localhost] => {
  "msg": "variable non existante"
}

PLAY RECAP *****
localhost                : ok=2    changed=0    unreachable=0    failed=0
```

lookup('file', ...) permet de manipuler un fichier.

```
# cat lookup_file.yml
---
- hosts: centos1
  tasks:
    - debug: msg="{{ lookup( 'file' , '/etc/motd' ) }}"
```

```
# ansible-playbook lookup_file.yml

PLAY [centos1] *****

TASK [Gathering Facts] *****
ok: [centos1]

TASK [debug] *****
ok: [centos1] => {
  "msg": "Bienvenue sur la machine Ansible Server"
}

PLAY RECAP *****
centos1                  : ok=2    changed=0    unreachable=0    failed=0
```


Quelques variantes d'utilisation de lookup('file', ...) :

```
# cat lookup_file.yml
- name: Test de lookup
  hosts: localhost
  tasks:
    - name: Vérifier si le fichier /tmp/fic1 existe
      debug:
        msg: "{{ lookup('file', '/tmp/fic1') }}" le fichier existe"
    - name: Autre tache
      debug:
        msg: autre tache
```

Exécution avec le fichier existant.

```
# ansible-playbook lookup_file.yml
PLAY [Test de lookup] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Vérifier si le fichier /tmp/fic1 existe] *****
ok: [localhost] => {
  "msg": " le fichier existe"
}

TASK [Autre tache] *****
ok: [localhost] => {
  "msg": "autre tache"
}

PLAY RECAP *****
localhost                : ok=3    changed=0    unreachable=0    failed=0
```

Exécution avec le fichier manquant.

```
# ansible-playbook lookup_file.yml
PLAY [Test de lookup] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Vérifier si le fichier /tmp/fic1 existe] *****
[WARNING]: Unable to find '/tmp/fic1' in expected paths (use -vvvvv to see paths)

fatal: [localhost]: FAILED! => {"msg": "An unhandled exception occurred while running the
lookup plugin 'file'. Error was a <class 'ansible.errors.AnsibleError'>, original
message: could not locate file in lookup: /tmp/fic1"}
    to retry, use: --limit @/root/lookup_file.retry

PLAY RECAP *****
localhost                : ok=1    changed=0    unreachable=0    failed=1
```

Pour ignorer les erreurs:

errors='ignore'

```
# cat lookup_file_ignore_errors.yml
- name: Test de lookup
  hosts: localhost
  tasks:
    - name: Vérifier si le fichier /tmp/fic1 existe
      debug: msg="{ lookup('file', '/tmp/fic1', errors='ignore' ) }}" les erreurs sont ignorées"
    - name: Autre tâche
      debug: msg="autre tâche"
```

```
# ansible-playbook lookup_file_ignore_errors.yml

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Vérifier si le fichier /tmp/fic1 existe] *****
[WARNING]: Unable to find '/tmp/fic1' in expected paths (use -vvvvv to see paths)

ok: [localhost] => {
  "msg": " les erreurs sont ignorées"
}

TASK [Autre tâche] *****
ok: [localhost] => {
  "msg": "autre tâche"
}

PLAY RECAP *****
localhost : ok=3    changed=0    unreachable=0    failed=0
```

Pour mettre les erreurs en alerte:

errors='warn'

```
# cat lookup_file_warning.yml
- name: Test de lookup
  hosts: localhost
  tasks:
    - name: Vérifier si le fichier /tmp/fic1 existe
      debug: msg="{ lookup('file', '/tmp/fic1', errors='warn' ) }}" les erreurs sont en warning"
    - name: Autre tâche
      debug: msg="autre tâche"
```

```
# ansible-playbook lookup_file_warning.yml

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Vérifier si le fichier /tmp/fic1 existe] *****
[WARNING]: Unable to find '/tmp/fic1' in expected paths (use -vvvvv to see paths)

[WARNING]: An unhandled exception occurred while running the lookup plugin 'file'. Error
was a <class 'ansible.errors.AnsibleError'>, original message: could not locate file
in lookup: /tmp/fic1

ok: [localhost] => {
  "msg": " les erreurs sont en warning"
}

TASK [Autre tâche] *****
ok: [localhost] => {
  "msg": "autre tâche"
}

PLAY RECAP *****
localhost : ok=3    changed=0    unreachable=0    failed=0
```

Fonctionnalités avancées

set_fact

```
# cat diff_setfact.yml
- hosts: localhost
  tasks:
    - set_fact:
        fact_time: "Var: {{lookup('pipe', 'date \"+%H:%M:%S\"')}}"
```

set_fact

La section `set_fact` permet de créer des variables au sein d'une tâche.

Lorsqu'une variable est définie avec le résultat d'une commande, il peut être important que cette variable s'initialise à sa première utilisation (comportement de `set_fact`;) et non à chaque fois qu'elle sera appelée (comportement de `vars`;).

Une variable `set_fact` à une priorité élevée. Cela peut poser des problèmes au développeur pour la surcharger.

Il ne faut utiliser `set_fact` que si c'est strictement nécessaire.

Les exemples ci-dessous mettent en évidence la différence entre une variable `vars` et `set_fact`:

```
# cat vars.yml
---
- name: Difference entre vars et set_fact
  hosts: localhost
  vars:
    var_time: "Var: {{lookup('pipe', 'date \"+%H:%M:%S\"')}}"
```

Execution du playbook.

```
# ansible-playbook vars.yml

PLAY [Difference entre vars et set_fact] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Afficher la variable var_time] *****
ok: [localhost] => {
  "var_time": "Var: 15:51:39"
}

TASK [Dormir 2 secondes] *****
changed: [localhost]

TASK [Afficher la variable var_time] *****
ok: [localhost] => {
  "var_time": "Var: 15:51:41"
}

PLAY RECAP *****
localhost : ok=4  changed=1  unreachable=0  failed=0
```

Avec setfact la variable est initialisée à la première utilisation.

```
# cat setfact.yml
---
- name: Difference entre vars et set_fact
  hosts: localhost
  tasks:
    - name: Initialiser fact_time
      set_fact:
        fact_time: "Var: {{lookup('pipe', 'date \"+%H:%M:%S\"')}}"
```

```
# ansible-playbook diff_setfact.yml

PLAY [Difference entre vars et set_fact] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Initialiser fact_time] *****
ok: [localhost]

TASK [Afficher la variable fact_time] *****
ok: [localhost] => {
  "fact_time": "Var: 17:01:58"
}

TASK [Dormir 2 secondes] *****
changed: [localhost]

TASK [Afficher la variable fact_time] *****
ok: [localhost] => {
  "fact_time": "Var: 17:01:58"
}

PLAY RECAP *****
localhost : ok=5  changed=1  unreachable=0  failed=0
```

Fonctionnalités avancées

Test de cohérence : assert et fail

```
- assert:                                arrêt si le test est faux
  that:
    - "ansible_os_family != 'RedHat'"

- assert:                                arrêt si UN test est faux
  that:
    - "'foo' in some_command_result.stdout"
    - "number_of_the_counting == 3"

- assert:
  that:
    - "my_param <= 100"
    - "my_param >= 0"
  msg: "'my_param' must be between 0 and 100"

- fail:                                arrêt si le test est vrai
  msg: "Le groupe propriétaire n'est pas root"
  when: chemin.stat.gr_name != "root"
```

Test de cohérence : assert et fail

Un assert indique si une expression est vraie ou fausse.

Si le test est faux, le traitement du playbook s'arrête avec un message d'erreur. On a la possibilité de personnaliser ce message.

Avec fail, si le test est vrai le traitement du playbook s'arrête.

Concrètement, on peut effectuer des tests de cohérences avant de continuer l'exécution d'autres tâches.

```
- assert:                                arrêt si le test est faux
  that:
    - "ansible_os_family != 'RedHat'"

- assert:                                arrêt si UN test est faux
  that:
    - "'foo' in some_command_result.stdout"
    - "number_of_the_counting == 3"

- assert:                                arrêt si UN test est faux
  that:
    - "my_param <= 100"
    - "my_param >= 0"
  msg: "'my_param' must be between 0 and 100"

- fail:                                arrêt si le test est vrai
  msg: "Le groupe propriétaire n'est pas root"
  when: chemin.stat.gr_name != "root"
```

Exemple1:

avec fail.

```
# cat fail_exemple.yml
---
- name: Test de fail
  hosts: localhost
  tasks:
    - name: Utilisation de stat pour faire des test
      stat:
        path: /opt/appli/etc
        register: chemin
    - name: Tester si le répertoire existe
      fail:
        msg: Le répertoire /opt/appli/etc n'existe pas
        when: not chemin.stat.exists and not chemin.stat.isdir
    - name: Test les droits du répertoire
      fail:
        msg: Le répertoire n'est pas en écriture pour le groupe
        when: not chemin.stat.wgrp
    - name: Vérifier le propriétaire du répertoire
      fail:
        msg: Le groupe propriétaire n'est pas root
        when: chemin.stat.gr_name != "root"
...

```

Avec fail le playbook s'arrête de s'exécuter dès qu'un test est faux.

```
# ansible-playbook fail_exemple.yml

PLAY [Test de fail] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Utilisation de stat pour faire des test] *****
ok: [localhost]

TASK [Tester si le répertoire existe] *****
skipping: [localhost]

TASK [Test les droits du répertoire] *****
fatal: [localhost]: FAILED! => {"changed": false, "msg": "Le répertoire n'est pas en
écriture pour le groupe"}
    to retry, use: --limit @/root/fail_exemple.retry

PLAY RECAP *****
localhost                : ok=2    changed=0    unreachable=0    failed=1

```

Exemple2:

Le playbook:

```
# cat  assert_fail.yml
---
- name: Test de assert
  hosts: localhost
  tasks:
    - name: Utilisation de stat
      stat:
        path: /opt/appli/etc
        register: chemin
    - name: Utilisation de assert pour effectuer un test
      assert:
        that:
          - chemin.stat.exists and chemin.stat.isdir
          - chemin.stat.wusr
        msg: Anomalie sur le repertoire /opt/appli/etc
    - name: Utilisation de fail pour effectuer un test
      fail:
        msg: Le groupe proprietaire n'est pas root
        when: chemin.stat.gr_name != "root"
    - name: Les autres taches
      debug:
        msg: SUITE DES TACHES
...

```

Fonctionnement où tout est OK:

```
# ansible-playbook  assert_exemple.yml

PLAY [Test de assert] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Utilisation de stat] *****
ok: [localhost]

TASK [Utilisation de assert pour effectuer un test] *****
ok: [localhost] => {
  "changed": false,
  "msg": "All assertions passed"
}

TASK [Utilisation de fail pour effectuer un test] *****
skipping: [localhost]

TASK [Les autres taches] *****
ok: [localhost] => {
  "msg": "SUITE DES TACHES"
}

PLAY RECAP *****
localhost                : ok=4    changed=0    unreachable=0    failed=0

```

Fonctionnement avec une action du assert :

```
# ansible-playbook  assert_exemple.yml

PLAY [Test de assert] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Utilisation de stat] *****
ok: [localhost]

TASK [Utilisation de assert pour effectuer un test] *****
fatal: [localhost]: FAILED! => {
  "assertion": "chemin.stat.exists and chemin.stat.isdir",
  "changed": false,
  "evaluated_to": false,
  "msg": "Anomalie sur le repertoire /opt/appli/etc"
}
    to retry, use: --limit @/root/assert_exemple.retry

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

Fonctionnement avec une action du fail :

```
# ansible-playbook  assert_exemple.yml

PLAY [Test de assert] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Utilisation de stat] *****
ok: [localhost]

TASK [Utilisation de assert pour effectuer un test] *****
ok: [localhost] => {
  "changed": false,
  "msg": "All assertions passed"
}

TASK [Utilisation de fail pour effectuer un test] *****
fatal: [localhost]: FAILED! => {"changed": false, "msg": "Le groupe propriétaire n'est pas root"}
    to retry, use: --limit @/root/assert_exemple.retry

PLAY RECAP *****
localhost                : ok=3    changed=0    unreachable=0    failed=1
```


Fonctionnalités avancées

dry-run, step-by-step et diff

Le mode verbeux : `-v` `-vv` `-vvv` `-vvvv`

```
# ansible-playbook --check mon_playbook.yml
```

```
# ansible-playbook --step mon_playbook.yml
```

```
Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: y
```

```
Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: *****
```

```
TASK [Gathering Facts] *****
```

```
# ansible-playbook --check --diff mon_playbook.yml
```

dry-run, step-by-step et diff

Exécution du playbook en mode verbeux:

les options suivantes sont disponibles

<code>-v</code>	peu verbeux
<code>-vv</code>	verbeux
<code>-vvv</code>	très verbeux
<code>-vvvv</code>	très très verbeux

```
# ansible-playbook playbook.yml -vv
```

Exécution du playbook en mode check :

Exécuter un playbook en dry-run est une exécution en mode check. Les actions sont simulées.

```
# ansible-playbook --check mon_playbook.yml
```

On peut activer ou désactiver le mode check (dry-run) au niveau d'une tâche.

tasks:

```
# cette tâche va apporter des modifications au système même en mode check
- command: /xxx/zzz/une_commande
  check_mode: no
```

```
# cette tâche sera toujours exécutée sous le mode check et ne changera pas le système
- command: /xxx/zzz/une_commande
  check_mode: yes
```

Exécution du playbook en mode pas à pas :

En exécutant un playbook en mode step by step, il sera demandé à chaque tâche de confirmer ou pas l'action.

```
# ansible-playbook --step mon_playbook.yml

PLAY [] *****
Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: y

Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: *****

TASK [Gathering Facts] *****
ok: [localhost]
Perform task: TASK: Fichier de conf d un service (N)o/(y)es/(c)ontinue: y

Perform task: TASK: Fichier de conf d un service (N)o/(y)es/(c)ontinue: *****

TASK [Fichier de conf d un service] *****
changed: [localhost]
Perform task: TASK: shell (N)o/(y)es/(c)ontinue: y

Perform task: TASK: shell (N)o/(y)es/(c)ontinue: *****

TASK [shell] *****
changed: [localhost]
```

Le mot clé diff :

On peut visualiser les différences qui seraient apportées à un fichier : --diff

On peut l'utiliser seul ou avec --check.

```
# ansible-playbook --check --diff mon_playbook.yml
...
TASK [Installation du fichier de configuration Apache] *****
ok: [localhost]

TASK [Installation de la page de garde] *****
--- before: /var/www/html/index.html
+++ after: /root/.ansible/tmp/ansible-local-12376jXlmib/tmp8gTyY0/index_template.html
@@ -3,7 +3,7 @@
  <body>
    <hr>
    <center><h1>Machine  localhost</h1>
-     du domaine  </center>
+     du domaine  home</center>
    <br><hr>
    <center><b>
<br> Bonjour root
@@ -15,7 +15,7 @@
  <br>
  <br> La liste des adresses IP :<br>
    192.168.122.1<br>
-    10.0.2.15<br>
+    192.168.1.32<br>
  <br> La liste des interfaces réseaux :<br>
    interface : lo<br>
    interface : enp0s3<br>

changed: [localhost]
...
```

Notes

Compléments

Dans ce chapitre, nous allons étudier quelques fonctionnalités supplémentaires, telles que Ansible Vault ou la création d'un module.

Compléments

- La création d'un module
- Ansible Vault et l'encryptage

Compléments

La création d'un module

```
# cat library/monmodule1.py
#!/usr/bin/python

from ansible.module_utils.basic import *

def main():

    module = AnsibleModule(argument_spec={})
    response = {"Bonjour": "Aurevoir"}
    module.exit_json(changed=False, meta=response)

if __name__ == '__main__':
    main()
```

La création d'un module

La création d'un module nécessite des compétences en programmation Python. Il est nécessaire d'importer des modules Python spécifiques à Ansible.

Exemple de base :

L'arborescence :	playbook.yml	pour le playbook utilisant le module
	library/module.py	pour le module

```
# cat library/monmodule1.py
#!/usr/bin/python

from ansible.module_utils.basic import *

def main():

    module = AnsibleModule(argument_spec={})
    response = {"Bonjour": "Aurevoir"}
    module.exit_json(changed=False, meta=response)

if __name__ == '__main__':
    main()
```

Le playbook

```
# cat monmodule1.yml
---
- name: Test de mon module
  hosts: localhost
  tasks:
    - name: Test de monmodule1
      monmodule1:
        register: resultat
    - name: Affichage du resultat
      debug: var=resultat
...
```

Résultat d'exécution.

```
# ansible-playbook monmodule1.yml

PLAY [Test de mon module] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Test de monmodule1] *****
ok: [localhost]

TASK [Affichage du resultat] *****
ok: [localhost] => {
  "resultat": {
    "changed": false,
    "failed": false,
    "meta": {
      "Bonjour": "Aurevoir"
    }
  }
}

PLAY RECAP *****
localhost : ok=3    changed=0    unreachable=0    failed=0
```

Un exemple plus évolué :

Ce module nécessite deux arguments (obligatoires) : fichier et ligne.

Le module ajoute à la fin du fichier (variable « fichier ») une ligne (variable « ligne »). Si le fichier n'existe pas, il est créé.

La documentation est définie via la variable DOCUMENTATION.

Le module en python :

```
# cat library/monmodule2.py
#!/usr/bin/python
# -*- coding: utf-8 -*-
#

# La documentation du module
DOCUMENTATION = '''
---
author: Baranger
version_added: "1.0"
module: module2
short_description: un module simple
description:
    - Ajoute un texte en fin de fichier. Si le fichier n existe pas, il est cree.
options:
    fichier:
        description:
            Le nom du fichier
    ligne:
        description:
            Ligne a inserer en fin de fichier
notes:
requirements: []
'''

EXAMPLES = '''
- name: "Un exemple"
  module2: fichier=fic1 ligne="Voici mon texte"
'''

# Le code du module
def main():
    # La declaration du module
    module = AnsibleModule(
        argument_spec=dict(
            fichier = dict(required=True),          # définit un argument obligatoire
            ligne = dict(required=True),
        )
    )
    # Recuperation des arguments
    fichier = module.params['fichier']              # récupère un argument
    ligne = module.params['ligne']

    # Traitement
    try:
        lefichier = open(fichier, "a")              # fichier en mode append
        lefichier.write(ligne)                      # ajoute une ligne dans le fichier
        lefichier.close()
    except:                                          # gestion des erreurs
        module.fail_json(msg="Anomalie au sein du module.")

    module.exit_json(changed=True, meta=fichier)

# Import Ansible Utilities
from ansible.module_utils.basic import *
main()
```


Le playbook:

```
# cat monmodule2.yml
---
- name: Test du module
  hosts: localhost
  tasks:
    - name: Test de monmodule2
      monmodule2:
        fichier: essai
        ligne: C est une belle journee
        register: resultat
    - name: Affichage du resultat
      debug: var=resultat
...

```

L'exécution:

```
# ansible-playbook monmodule2.yml

PLAY [Test du module] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Test de monmodule2] *****
changed: [localhost]

TASK [Affichage du resultat] *****
ok: [localhost] => {
  "resultat": {
    "changed": true,
    "failed": false,
    "meta": "essai"
  }
}

PLAY RECAP *****
localhost                : ok=3    changed=1    unreachable=0    failed=0

```

Le fichier résultant de l'exécution du module :

```
# cat essai
C est une belle journee

```

Modification du playbook pour générer une erreur :

```
# ansible-playbook monmodule2.yml
                                (avec fichier=/aaaaa/xxxxx/bbbb/essai)

PLAY [localhost]

TASK [Gathering Facts]
ok: [localhost]

TASK [Test de monmodule2]
fatal: [localhost]: FAILED! => {"changed": false, "msg": "Anomalie au sein du module."}
    to retry, use: --limit @/root/Playbooks/monmodule2.retry

PLAY RECAP
localhost                : ok=1    changed=0    unreachable=0    failed=1

```

Affichage de la documentation du module :

```
# ansible-doc -M library monmodule2
> MODULE2      (/root/Playbooks/library/monmodule2.py)

    Ajoute un texte en fin de fichier. Si le fichier n existe pas, il
    est cree.

OPTIONS (= is mandatory):

- fichier
    Le nom du fichier
    [Default: (null)]

- ligne
    Ligne a inserer en fin de fichier
    [Default: (null)]

AUTHOR: Baranger

EXAMPLES:
- name: "Un exemple"
  module2: fichier=fic1 ligne="Voici mon texte"
```

Compléments

Ansible Vault et l'encryptage

```
# ansible-vault encrypt fic_mdp.yml
# ansible-vault encrypt fic_mdp.yml --vault-id vault_mdp.key

# ansible-playbook playbook.yml -e @fic_mdp.yml --ask-vault-pass
# ansible-playbook playbook.yml -e @fic_mdp.yml --vault-id
vault_mdp.key

# ansible-vault encrypt_string 'secure!' --vault-id vault_mdp.key

# ansible-vault rekey fic_mdp.ymlb --vault-password-file \
    vault_mdp.key --new-vault-password-file new_vault_mdp.key
# ansible-vault create fic_mdp.yml --vault-id vault_mdp.key
# ansible-vault view fic_mdp.yml --vault-id vault_mdp.key
# ansible-vault edit fic_mdp.yml --vault-id vault_mdp.key
# ansible-vault decrypt fic_mdp.yml --vault-id vault_mdp.key
```

Ansible Vault et l'encryptage

Ansible Vault permet de gérer le cryptage de données. La commande est `ansible-vault`.

Pour crypter un fichier :

[encrypt](#)

Le mot de passe Vault sera demandé.

```
# ansible-vault encrypt fic_mdp.yml
```

Pour éviter une saisie clavier le mot de passe Vault peut-être stocké dans un fichier. L'option `--vault-id` ou `--vault-password-file` permet de spécifier ce fichier.

```
# ansible-vault encrypt fic_mdp.yml --vault-id vault_mdp.key
# ansible-vault encrypt fic_mdp.yml --vault-password-file vault_mdp.key
```

Pour exécuter un playbook:

Avec saisie du mot de passe Vault

```
# ansible-playbook playbook.yml -e @fic_mdp.yml --ask-vault-pass
```

En indiquant le nom du fichier contenant le mot de passe Vault

```
# ansible-playbook playbook.yml -e @fic_mdp.yml --vault-id vault_mdp.key
# ansible-playbook playbook.yml -e @fic_mdp.yml --vault-password-file \
vault_mdp.key
```

Pour encrypter une chaîne de caractères :

[encrypt_string](#)

```
# ansible-vault encrypt_string 'secure!' --vault-id vault_mdp.key
```

Pour recrypter un fichier avec un nouveau mot de passe Vault :

[rekey](#)

```
# ansible-vault rekey fic_mdp.ymlb --vault-password-file vault_mdp.key \
--new-vault-password-file new_vault_mdp.key
```

Pour créer un fichier crypté :

[create](#)

```
# ansible-vault create fic_mdp.yml --vault-id vault_mdp.key
```

Pour visualiser un fichier crypté en clair :

[view](#)

```
# ansible-vault view fic_mdp.yml --vault-id vault_mdp.key
```

Pour éditer et modifier un fichier crypté :

[edit](#)

```
# ansible-vault edit fic_mdp.yml --vault-id vault_mdp.key
```

Pour décrypter un fichier crypté :

[decrypt](#)

```
# ansible-vault decrypt fic_mdp.yml --vault-id vault_mdp.key
```

Exemples:

Contenu du fichier contenant les mots de passe.

```
# cat fichier_motdepasse.yml
---
mot_de_passe: secure!
mdp_bdd: ansible123
...
```

Cryptage du fichier contenant les mots de passe.

```
# ansible-vault encrypt fichier_motdepasse.yml
New Vault password: mot de passe saisie: mdpvault
Confirm New Vault password:
Encryption successful
```

```
# cat fichier_motdepasse.yml
$ANSIBLE_VAULT;1.1;AES256
31306263303730396666373061633939386664343461326665383166303266336538356337376139
3036656261343839623839323735393866386138656164620a303937343362646631353766613831
37613534646666383837313036346261653565363732366361323232343234666266643430623730
6166663130623831360a353733306535323138623261366234363231386432626533316234306133
38613737336136326266383534623164313863666232313663656261396633346566663038376632
3763613832393961363461613266646134613563383733366363
```

Contenu du playbook

```
# cat vault1.yml
---
- name: test de Vault
  hosts: localhost
  tasks:
    - name: Afficher la valeur de mot_de_passe
      debug:
        msg: "Reponse: {{mot_de_passe}}"
...
```

Execution du playbook. Le mot de passe Vault est demandé.

```
# ansible-playbook vault1.yml -e @fichier_motdepasse.yml --ask-vault-pass
Vault password:          saisie de : mdpvault

PLAY [test de Vault] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Afficher la valeur de mot_de_passe] *****
ok: [localhost] => {
  "msg": "Reponse: secure1!"
}

PLAY RECAP *****
localhost                : ok=2    changed=0    unreachable=0    failed=0
```

Au lieu de saisir le mot de passe Vault en mode interactif il est possible de le stocker dans un fichier.

```
# cat mdp_ansible_vault.key
mdpvault
```

Le fichier à crypter :

```
# cat fichier_mdp.yml
---
mot_de_passe: secure1!
mdp_bdd: ansible123
...
```

L'option --vault-id permet de spécifier le fichier contenant le mot de passe Vault.

```
# ansible-vault encrypt fichier_mdp.yml --vault-id mdp_ansible_vault.key
Encryption successful

# cat fichier_mdp.yml
$ANSIBLE_VAULT;1.1;AES256
33343766383436316365633534303436613234656463616432336139386166366336333566636264
3635336631666636313539343263346230646562303462630a303934313035653437396665353932
37326537336134316363653734356437316137643736336336666664336137396163343231663735
3535643530333532360a336136313561653461353462353038663862363730383633646463313832
38366165383432396265333565663363666230353132636566323533616363623336656334633033
3165326532306139326433306233383531636366316135323937
```

Exécution du playbook avec l'option --vault-id ou --vault-password-file.

```
# ansible-playbook vault1.yml \
  -e @fichier_mdp.yml --vault-id mdp_ansible_vault.key
ou
# ansible-playbook vault1.yml \
  -e @fichier_mdp.yml --vault-password-file mdp_ansible_vault.key

PLAY [test de Vault] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Afficher la valeur de mot_de_passe] *****
ok: [localhost] => {
  "msg": "Reponse: secure1!"
}

PLAY RECAP *****
localhost                : ok=2    changed=0    unreachable=0    failed=0
```

Modification du contenu du fichier (pour modifier un mot de passe ou ajouter un mot de passe) :

```
# ansible-vault edit fichier_mdp.yml --vault-id mdp_ansible_vault.key
```

Lance directement l'éditeur de texte par défaut du système d'exploitation (ici vi) avec le fichier décrypté (évidemment!). La manipulation a été de modifier la variable du mot_de_passe par **new!secure**.

Le fichier a bien été recrypté.

```
# cat fichier_mdp.yml
$ANSIBLE_VAULT;1.1;AES256
33663334343336323335363161303131643636366138353064623636383264336631343131386532
6236316563353063393162623235613764333066326234320a386533383066306166623037373437
33623632623438303735313136643461636436643463636234343536326162356431626161653638
3531613235346635610a636433343461323736366436346339356364633239313764636337653466
64376337636232623066653334663465316239303762336330366138343436666638336334663331
66396665633064333161626332373637383864343832616366393634383066643438343637653266
666164363535336535363233303135313432
```

Le nouveau mot de passe est bien pris en compte.

```
# ansible-playbook vault1.yml \
    -e @fichier_mdp.yml --vault-id mdp_ansible_vault.key

PLAY [test de Vault] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Afficher la valeur de mot_de_passe] *****
ok: [localhost] => {
  "msg": "Reponse: new!secure"
}

PLAY RECAP *****
localhost                : ok=2    changed=0    unreachable=0    failed=0
```

Changement du mot de passe pour Vault:

```
# cat new_mdp_ansible_vault.key
new_vaultpass1!
```

On recrypte le fichier contenant les mots de passes de notre playbook avec le nouveau fichier de mot de passe Vault :

```
# ansible-vault rekey fichier_mdp.yml \
    --vault-password-file mdp_ansible_vault.key \
    --new-vault-password-file new_mdp_ansible_vault.key
Rekey successful
```

Le fichier a bien été recrypté.

```
# cat fichier_mdp.yml
$ANSIBLE_VAULT;1.1;AES256
65646561363464393138353762646338616433303365383636383939366438636333393930633431
3061306561643139373766323430373635316462343532660a386534306434666337633230373538
39663566303935313331613665386337306163646661613430316161343665306632303165616463
6661653264376635360a363330313230353235336637376639346165393834306565636165656135
31396538363832646435346366666365666663346237663237353361613036303039373537616137
66363463323131613137356661363437386633653931396462323066663730313436373832396665
616230363134643865626633363961363234
```

Exécution du playbook.

```
# ansible-playbook vault1.yml \  
    -e @fichier_mdp.yml --vault-id new_mdp_ansible_vault.key  
  
PLAY [test de Vault] *****  
  
TASK [Gathering Facts] *****  
ok: [localhost]  
  
TASK [Afficher la valeur de mot_de_passe] *****  
ok: [localhost] => {  
    "msg": "Reponse: new1!secure"  
}  
  
PLAY RECAP *****  
localhost                : ok=2    changed=0    unreachable=0    failed=0
```

Cryptage d'une chaîne de caractères

Le fichier de mot de passe avant cryptage.

```
# cat fichier_motdepasse2.yml                                # cat mdp_vault.key
---                                                         mdpvault
mdp_appli: new_appli_123
mot_de_passe: secure!
mdp_bdd: ansible123
...
```

Génération des chaînes de caractères cryptées pour les mots de passe.

```
# ansible-vault encrypt_string 'secure!' --vault-id mdp_vault.key
!vault |
     $ANSIBLE_VAULT;1.1;AES256
3732336238323739636462633333561306561626338333166303162363338663261313835356339
3438326365383666383164663430616537633862613063640a636365303938643834396237613762
63656363376134363037343965633862373439323334323765623431306566613664393965663634
3936373730626561630a613962393062653466663339386535643062363563396132353064643633
3039
Encryption successful
```

```
# ansible-vault encrypt_string 'ansible123' --vault-id mdp_vault.key
!vault |
     $ANSIBLE_VAULT;1.1;AES256
38623632646135346433386635656537653337383631613939376165366137643336613433333631
3632656464336436303134613331306137343234313863390a656234343033343565666464326164
65326236343534386132373065303466373964343237373963373466333639383237663039613238
3134393462356536380a363236663535393638353936346434313438613536633866363636653563
3061
Encryption successful
```

Remplacez les mots de passe en clair du playbook par leur valeurs cryptées.

```
# cat fichier_motdepasse2.yml
---
mdp_appli: new_appli_123
mot_de_passe: !vault |
     $ANSIBLE_VAULT;1.1;AES256
3732336238323739636462633333561306561626338333166303162363338663261313835356339
3438326365383666383164663430616537633862613063640a636365303938643834396237613762
63656363376134363037343965633862373439323334323765623431306566613664393965663634
3936373730626561630a613962393062653466663339386535643062363563396132353064643633
3039
mdp_bdd: !vault |
     $ANSIBLE_VAULT;1.1;AES256
38623632646135346433386635656537653337383631613939376165366137643336613433333631
3632656464336436303134613331306137343234313863390a656234343033343565666464326164
65326236343534386132373065303466373964343237373963373466333639383237663039613238
3134393462356536380a363236663535393638353936346434313438613536633866363636653563
3061
...
```

Le playbook.

```
# cat vault2.yml
---
- name: Test de Vault
  hosts: localhost
  tasks:
    - name: Afficher la valeur de mot_de_passe
      debug:
        msg: "mot_de_passe contient {{mot_de_passe}}"
    - name: Afficher la valeur de mdp_bdd
      debug:
        msg: "mdp_bdd contient {{mdp_bdd}}"
    - name: Afficher la valeur de mdp_appli
      debug:
        msg: "mdp_appli contient {{mdp_appli}}"
...
```


Exécution du playbook.

```
# ansible-playbook vault2.yml \  
-e @fichier_motdepasse2.yml --vault-id mdp_vault.key
```

Extrait du résultat d'exécution.

```
TASK [Afficher la valeur de mot_de_passe] *****  
ok: [localhost] =>      "msg": "mot_de_passe contient secure!"  
  
TASK [Afficher la valeur de mdp_bdd] *****  
ok: [localhost] =>      "msg": "mdp_bdd contient ansible123"  
  
TASK [Afficher la valeur de mdp_appli] *****  
ok: [localhost] =>      "msg": "mdp_appli contient new_appli_123"
```

La commande suivante permet de créer un fichier. Il est automatiquement crypté.

```
# ansible-vault create fic_mdp.yml --vault-id mdp_vault.key  
Lance l'éditeur de texte par défaut (ici vi) pour la saisie du contenu :  
---  
mdp_autre: insolite_password  
mot_de_passe: secure!  
mdp_bdd: ansible123  
...
```

Affichage du fichier généré.

```
# cat fic_mdp.yml  
$ANSIBLE_VAULT;1.1;AES256  
65313136643761333363383836663632396539613961633534343865303531323435626135666336  
6335373933323330393361363465366563383833653730380a383761323136383135303635376636  
64396166666666373532383431646161623539656535343232613935626232353136306138336562  
3033656462633331380a633362643864386537653032323063373265346164323634326665333032  
32353035343363366430343766643464666266303730353163333531366366616164313736346162  
66643962323165346632343337626537643465303030343363326136316433323932616265356331  
36656563303865343263613562366165646363363763356561323434336232353733613834303662  
65343566613565663631
```

Pour visualiser le contenu du fichier crypté:

```
# ansible-vault view fic_mdp.yml --vault-id mdp_vault.key  
---  
mdp_autre: "insolite_password"  
mot_de_passe: "secure!"  
mdp_bdd: "ansible123"  
...
```

Pour éditer un fichier crypté :

```
# ansible-vault edit fic_mdp.yml --vault-id mdp_vault.key
```

Notes

Fin de session de Formation

Je vous recommande de relire ce support de cours d'ici les deux semaines à venir, et de refaire des exercices.

Il ne vous reste plus qu'à mettre en œuvre ces nouvelles connaissances au sein de votre entreprise.

Merci, et à bientôt.

Jean-Marc Baranger

Theo Schomaker

