

# Formation Ansible – Python

Ihab ABADI / UTOPIOS

# SOMMAIRE - Ansible

## 1. Introduction

1. Ansible ?
2. Autres produits

## 2. Mise en œuvre de Ansible

1. Installation
2. Définition de l'inventaire
3. Les variables
4. Les outils / les accès

## 3. Utilisation des modules

1. Définition / syntaxe
2. Exemples

# SOMMAIRE - Ansible

## 4. Les Playbooks

1. Définition
2. Syntaxe
3. Exemples

## 5. Structures de contrôle

1. Définitions
2. Facts / Conditions / boucles / inclusions
3. Exemples

## 6. Templates / Rôles

1. Templates
2. Rôles

# Ansible - Introduction

Ansible est un projet récent (2012) qui a été développé entièrement en python et qui répond à des besoins de remplacement, de déploiement ou de changements de serveur. Ces actions peuvent être menées sur l'ensemble ou une partie des serveurs.

C'est un système agentless capable de piloter des systèmes Windows, Linux, Unix et aussi des équipements réseau tel que Cisco ou Juniper.

Pour fonctionner ansible n'a besoin que d'un accès ssh et de python ou des APIs. Il n'y a pas de serveur central, tout ordinateur ayant Ansible peut commander les autres.

- Autres produits ?

Les autres produits qui permettent l'orchestration de serveur :

- Puppet (client/serveur)  
C'est un projet qui a été créé en 2005 et qui est écrit en RUBY.
- Chef (client/serveur)  
Il a été créé en 2009 par des ex-employés de Puppetlabs. Il a été écrit en RUBY puis ré-écrit en Erlang.
- Saltstack (client/serveur ou agentless)  
Cette plateforme a vu le jour en 2011. Elle est écrite en python.

# Ansible – Mise en oeuvre

## 2. Mise en oeuvre de ANSIBLE

- Installation

Ansible s'installe facilement et uniquement sur l'ordinateur de management. Elle peut se faire via les paquets de la distribution (UBUNTU/DEBIAN) ou via des dépôts tiers EPEL (REDHAT/CENTOS). Elle peut se faire aussi via PIP.

Exemple : `pip install ansible`

- Via pip on peut installer une version spécifique d'ansible
- Python 2.5 au minimum doit être installé sur les machines cibles
- Pour les machines utilisant SELINUX il faudra installer le paquet python-selinux

- Définition de l'inventaire

Le fichier inventaire "*hosts*" se trouve dans "*/etc/ansible*" par défaut. Il définit la liste des machines qui pourrait répondre aux requêtes d'Ansible. Les ordinateurs définis dans le fichier "*hosts*" peuvent être organisés en groupe. On peut même créer des groupes de groupe. Ce fichier est modifiable dans la configuration d'ansible ou même spécifiable à l'exécution des commandes ansible.

## Exemple :

`[Ordonnanceur]`

10.2.2.5

`[file_slow]`

`[File_fast]`

10.2.2.10

# Ansible – Mise en oeuvre

- Les variables

On peut définir des variables arbitraires qui peuvent être associées à une machine ou un groupe de machines. Elles permettent de modifier le comportement d'Ansible (info d'authentification). Les variables se définissent soit sur la ligne correspondante à la machine soit dans une section dédiée.

- Il est commun d'utiliser la variable "*type*" qui sera utile non pas à ansible mais qui sera utilisée par la suite dans un rôle ou un playbook (ensemble de tâches à effectuer).

[file\_slow]

```
nlame1.example.com type=master  
nlame2.example.com type=slave
```

- Un autre exemple de variable est `ansible_ssh_user`. Elle est utilisé pour établir des connexions SSH avec le bon utilisateur.

Exemple :

[file\_slow]

```
nlame1.example.com ansible_ssh_user=root
```

- Enfin il est possible de définir des variables dans les dossiers `group_vars` et `host_vars` (Format YML)

Exemple :

[file\_slow]

```
nlame1.example.com type=master  
nlame2.example.com type=slave
```

Contenu de : `/etc/ansible/group_vars/file_slow`  
`ansible_ssh_user : ansible`

# Ansible – Mise en oeuvre

Contenu de : /etc/ansible/host\_vars/  
n1ame1.example.com  
type : **master**

Contenu de : /etc/ansible/host\_vars/  
n1ame2.example.com  
type : **slave**

- Les outils / les accès

Les outils :

ansible fournit plusieurs outils en ligne de commande :

Outil	Description
ansible	Execution d'une tâche
ansible-playbook	Execution de playbook (ensemble de tâches à effectuer)
ansible-doc	Accès au listing + documentation des serveurs
ansible-vault	Gestion de fichiers chiffrés (stockage variable - mot de passe)
ansible-galaxy	Accès au dépôt des rôles d'ansible

Le module **ping** permet de valider les accès d'un inventaire. Les machines cibles répondront par pong.

Exemple :

**ansible -i hosts file\_slow -m ping**

nota :

- L'option -i spécifie le chemin vers l'inventaire
- L'option -m spécifie le module ansible à utiliser
- L'option all mis à la place de file\_slow aurait impacté toutes les machines présentes dans l'inventaire

# Ansible - Mise en œuvre

Atelier

- En utilisant deux machines virtuelles, exécuter le module ping d'ansible.



# Ansible – Mise en oeuvre

## Les accès :

L'accès SSH aux hôtes peut être explicité de plusieurs manières :

- En utilisant les informations de connexion dans l'inventaire
- En précisant toutes les informations sur la ligne de commande ansible
- En configurant la machine de management pour une connexion transparente

En pratique la méthode "*configurer la machine de management*" est la plus pratique à utiliser.

Pour cela il faudra :

- Créer une paire de clé SSH (ssh-keygen)
- Déployer la clé publique sur chacun des serveurs (ssh-copy-id)
- Créer une configuration sudo sans mot de passe sur les hôtes distants ou un login en tant que root. Le login utilisateur pour chaque hôte peut être spécifié dans l'inventaire.

## Exemple :

### `$ssh-keygen`

Generating public/private rsa key pair.

Enter file in which to save the key

(/home/user/.ssh/id\_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

...

`$ssh-copy-id ansible@nlame1.example.com`

Nota: sudo peut être configuré pour ne pas demander de mot de passe lors de son utilisation

`$cat /etc/sudoers.d/ansible`

ansible ALL=(ALL) NOPASSWD : ALL

# Ansible - Mise en œuvre

Atelier

- Relancer le module ping après avoir générer des clés ssh.

# Ansible – Utilisation des modules

## 3. Utilisation des modules

- Définition

Les modules sont la base des actions exécutées par Ansible. Chaque module est lié à une action spécifique. Les modules acceptent des arguments. La liste des modules proposé par l'application ansible est accessible grâce à la commande "*ansible-doc --list*". On peut retrouver la liste des modules à cette adresse : [http://docs.ansible.com/ansible/modules\\_by\\_category.html](http://docs.ansible.com/ansible/modules_by_category.html)

La syntaxe est :

ansible (hote/groupe/all) -m MODULE [-a "arg1=val1"]

- Exemple :

```
$ ansible all -m yum -a "name=httpd state=latest"
```

Cette commande permet de vérifier si la dernière version d'apache est installée, si non, alors elle installe la dernière version.

Voici une liste de modules couramment utilisés :

- *ping* : validation de l'inventaire
- *setup* : retourne une liste d'informations matériels de l'hôte
- *shell* et *command* : permettent d'exécuter des commandes sur les hôtes
- *user* : permet de gérer des utilisateurs sur les hôtes
- *file* : permet de gérer des droits sur des fichiers
- *service* : permet de gérer les services systèmes tel que : arrêt/démarrage/redémarrage ou activation et désactivation au boot
- *yum/apt/zypper* : permet de gérer l'installation, la mise à jour et la suppression de paquets

# Ansible - Mise en œuvre

Atelier

- À l'aide du module `command`, installer python sur les deux machines.
- À l'aide du module `copy` copier un script python vers les deux machines.
- À l'aide du module `command`, exécuter votre script python sur les deux machines.

# Ansible – Playbook

## 4. Les Playbooks :

### 1. Définition :

Les playbooks permettent d'exécuter un ensemble de tâches à effectuer sur les machines hôtes de manière séquentielle sur tout ou partie de l'inventaire.

Chaque tâche utilise un module de Ansible. On utilise le langage YAML pour écrire un playbook. Un playbook est constitué au minimum :

- D'une variable *hosts* qui désigne les machines cibles
- D'une variable *task* qui définit une action à accomplir.

Les playbooks peuvent aussi :

- Emettre des notifications, qui seront utiles pour déclencher une action dans certaines conditions
- Effectuer des actions conditionnelles selon leur valeur (par exemple le type de distribution du système)
- Utiliser des templates pour créer ou modifier des fichiers

On trouve aussi dans les playbooks :

- Des rôles à utiliser
- Des handlers (tâches spécifique se déclenchant grâce à une notification)
- Des éléments de configuration pour Ansible (par exemple l'utilisation de sudo)
- Des variables spécifiques qui ne sont pas relié à l'inventaire.

# Ansible – Playbook

## Exemple :

- hosts: file\_slow
- tasks:
  - name: Install apache
  - yum:
    - name: httpd
    - state: present
  - tags:
    - install\_httpd

L'exécution du playbook install\_apache.yml se fait grâce à la commande ansible-playbook :

```
$ansible-playbook install_apache.yml
```

Par défaut, un playbook s'exécute sur tous les hôtes concernés. Il est possible de cibler ces actions grâce aux options :

- Tags : cette option se définit dans le playbook et on l'appelle --tags nom\_du\_tag ou --skip-tags
- L'option -I qui permet de cibler une machine en particulier

## Exemple :

```
$ansible-playbook install_apache.yml --tags  
install_httpd -I nlame1.example.com
```

## La syntaxe YAML (Yet Another Markup Language)

C'est un langage facile à lire et à écrire. On peut y inclure une syntaxe JSON. On peut y manipuler du texte, des listes et des dictionnaires. L'indentation est très stricte (2 espaces / indentation). On démarre un playbook toujours avec --- puis on va à la ligne. Enfin les commentaires sont définis grâce au #

# Ansible - Playbook

Atelier

- Créer un premier playbook qui permet de:
- Installer Python
- Copier les scripts python d'une application.
- Exécuter le script python

# Ansible – Playbook

## Les tâches :

Chaque tâche est définie par un nom (rubrique *name*) pour commenter l'action et d'un module à appeler.  
Des attributs supplémentaires permettent de rendre les playbooks plus interactifs :

- *notify* : défini un Handler à appeler
- *register* : sauve le résultat d'une action dans une variable
- *include* : inclusion d'un fichier externe
- ...

## Notification et handlers :

Les handlers permettent de définir des actions qui ne seront exécutées qu'au déclenchement d'une notification. Les handlers ne sont pris en compte qu'après la fin de toutes les tâches présentes dans le playbook.

## Exemple :

- hosts: `file_slow`  
become: no tasks:  
- name: Action qui fait une modification file:  
  path: /tmp/toto state: touch mode: 0644  
  notify:  
  -handler1

Attention l'argument de *notify* et le nom du *handler* doivent être rigoureusement identique.



# Ansible - playbook

Atelier

- Créer un playbook qui permet
  - D'installer postgresql sur les machines.
  - De démarrer postgresql après l'installation uniquement.

# Ansible – Playbook

## Les variables :

Un playbook permet de définir des variables à utiliser pendant l'exécution des tasks.

La définition des variables peut se faire :

- Dans le block vars
- Dans un fichier séparé.

## Exemple:

```
vars_files: vars/main.yml
```

```
mysql_user: "admin "  
mysql_password: "secret "  
mysql_dbname: "blog "  
db_host: "192.168.0.22"  
webserver_host: "192.168.0.21"
```

# Ansible – Playbook

## Les variables :

Un playbook permet de d'enregistrer le résultat de l'exécution d'un module dans une variable à l'aide de l'argument register

# Ansible – Playbook

## 5. Structures de contrôle

- Définition :

Les structures de contrôle sont probablement la partie la plus utile. En effet grâce aux structures de contrôle on peut manipuler les données provenant des hôtes de façon très flexible et puissante.

- Facts / Conditions / boucles / inclusions :

- Les facts : La première action effectuée lors de l'exécution d'un playbook est de récolter des informations de chacun des hôtes distants. Ces *facts* alimentent des variables (préfixées par `Ansible`). Ces variables seront utilisables dans les filtres des tests et des boucles du playbook. Pour avoir une liste de tous les *facts* qu'Ansible met à votre disposition il faut utiliser le module `setup`

- Les conditions : Il est utile que certaines tâches du playbook soient effectuables seulement si une condition spécifique est remplie ou non. La directive **when** permet d'effectuer ce test

Exemple :

- **name: Installation apache (Debian)**

```
apt:
  name: apache2path: /tmp/toto
  state: present
  when: ansible_os_family == 'Debian'
```

- **name: Installation apache (REHL)**

```
yum :
  name: httpd
  state: present
  when: ansible_os_family == 'RedHat'
```

# Ansible - playbook

Atelier

- Ecrire un playbook qui permet d'installer et configurer les composants suivants à la fois sur un environnement debian et redhat.
  - Postgres
  - Nginx
  - Python

# Ansible – Playbook

- Les boucles : Les boucles permettent d'itérer sur les variables de type liste et dictionnaire. Cette technique permet d'exécuter plusieurs fois la même action sur un nombre d'éléments indéfini lors de l'écriture du rôle ou playbook. La liste complète des boucles est disponible dans la [documentation ansible](#).

## Exemple :

*with\_items* permet de boucler sur une liste. A chaque itération une variable item prend la valeur suivante de la liste :

- name: Installation d'éditeurs
- yum:
  - name: "{{ item }}"
  - with\_items:
    - vim
    - emacs
    - nano

# Ansible - Playbook

Atelier

- Ecrire un playbook qui permet de copier plusieurs scripts python en même temps et qui les exécute.

# Ansible – Playbook

- Les inclusions : La directive *include* permet d'importer une liste de tâches ou de handlers. Cette technique permet de rendre génériques certaines actions.

Exemple :

hosts: all

tasks:

- include: common-setup.yml

- name : something else

- ...



- En utilisant Ansible, nous souhaitons déployer notre application api rest product avec base de données sur deux environnements différents, exécuter les tests et récupérer le résultat des tests.
- Créer un playbook qui permet de :
- Installer les dépendances sur chaque environnement.
- Déployer et démarrer l'application.
- Exécuter les tests et récupérer le résultat.

# Ansible –Rôles

## Définition des rôles :

Les rôles sont des playbooks génériques, qui peuvent être intégrés dans d'autres playbooks. Cette notion est essentielle afin de créer des tâches complexes. En effet afin de rendre des playbooks lisibles il vaut mieux construire des rôles que l'on peut combiner ensemble plutôt que créer un rôle complexe qui sera difficile à maintenir et complexe à utiliser et peu lisible. Pour chaque élément d'un rôle (tâche, handlers, ...) un fichier nommé main.yml sert de point d'entrée.

## Exemple d'arborescence :

Dans le répertoire playbooks, où se trouvent ansible.cfg, hosts et deploy.yml, on va créer un dossier « roles » et on lancera la commande :

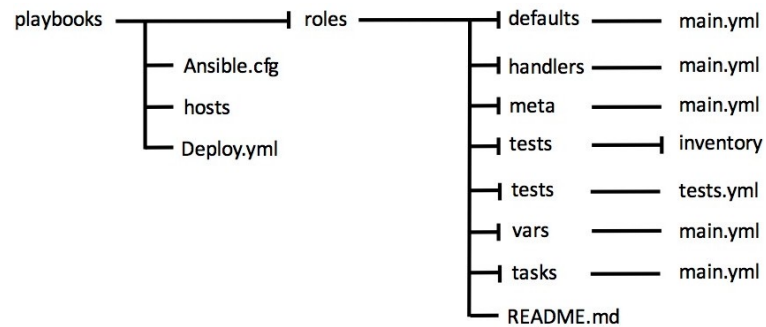
```
$ ansible-galaxy init -p roles/ test
```

L'argument **init** permet d'initier la création d'un rôle et il est suivi de l'option **-p** pour indiquer le chemin où l'on désire créer le rôle. Si l'option **p** n'est pas définie le rôle sera créé dans le répertoire courant. Ici on crée un rôle test.

- test was created successfully

# Ansible – Templates / Rôles

Notre rôle est bien créé et il contient les répertoires suivants :



Grâce à cette arborescence il ne sera pas nécessaire de nommer les directives vars, tasks ou handlers dans les fichiers vars/main.yml, tasks/main.yml et handlers.yml. Cette information sera obtenue à partir du nom du répertoire. De la même façon les modules copy et template iront chercher les fichiers sources directement dans les répertoires files et templates, sans avoir besoin de préciser le path.

## Obtenir des rôles :

Ansible fournit une bibliothèque de rôles en ligne : [Galaxy](#).

Les rôles disponibles sur Galaxy sont soumis par la communauté Ansible, et leur qualité varie. Un système de notation permet de trouver les meilleurs rôles.

## Exemple : Installation

```
$ ansible -galaxy install geerlingguy.apache
```

Suppression

```
$ ansible -galaxy remove geerlingguy.apache
```

# Ansible - TP

TP

- Créer un rôle pour le TP du slide 25

# Module Ansible

- Ansible nous permet de développer nos propres modules.
- Un module Ansible est développé en Python à l'aide du package Ansible.
- Un module Ansible custom est exécuté de la même façon que les modules builtin.
- Un module Ansible sera exécuté sur chaque node.

# Module Ansible

- Pour créer un module Ansible, nous utiliserons la classe AnsibleModule fournit par le package ansible.module\_utils.basic.
- Exemple :

```
from ansible.module_utils.basic import *

def main():

    module = AnsibleModule(argument_spec={})
    response = {"result" : "hello world"}
    module.exit_json(changed=False, meta=response)

if __name__ == '__main__':
    main()
```

# Module Ansible

- Un module ansible peut définir ses propres arguments

```
fields = {  
    "name": {"default": "unknown", "type": "str"},  
    "description": {"default": "empty", "required":  
False, "type": "str"},  
    "persons": {"required": True, "type": "list"},  
}  
module = AnsibleModule(argument_spec=fields)
```

# Ansible - Module

Exercice

- Ecrire un module python qui permet de tester si une liste de ports est ouvert sur une machine.



# Plugin Ansible

- Les plugins sont des morceaux de code qui augmentent les fonctionnalités de base d'Ansible.
- Ansible utilise une architecture de plugin pour permettre un ensemble de fonctionnalités riche, flexible et extensible

# Plugin Ansible

- Ansible est livré avec un certain nombre de plugins pratiques, et vous pouvez facilement écrire le vôtre.
  - Action Plugins
  - Become Plugins
  - Cache Plugins
  - Callback Plugins
  - Cliconf Plugins
  - Connection Plugins
  - Httpapi Plugins
  - Inventory Plugins
  - Lookup Plugins
  - Netconf Plugins
  - Shell Plugins
  - Strategy Plugins
  - Vars Plugins