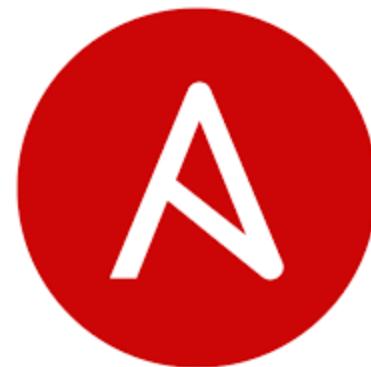


# Ansible, automatiser la gestion des serveurs



**Brahim HAMDI**

Consultant/Expert en DevOps & Cloud

Contact: [brahim.hamdi.consult@gmail.com](mailto:brahim.hamdi.consult@gmail.com)

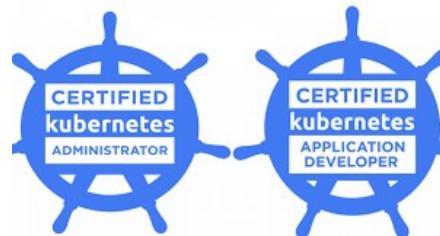
*Novembre 2023*

# Présentation du formateur

---

Brahim HAMDI

- Consultant/Formateur DevOps & Cloud
- Expert DevOps, Cloud, CyberOps et Linux



# Objectifs pédagogiques

---

- Connaître les caractéristiques et le fonctionnement d'Ansible
- Mettre en œuvre les playbooks, modules, rôles, tâches...
- Comprendre comment optimiser le pilotage d'un parc de serveurs et le déploiement d'applications
- Maîtriser les bonnes pratiques sous Ansible

- Administrateurs
- Développeurs
- Architectes

- Connaître l'administration des systèmes Linux et un langage de développement de scripts

- 1 Introduction à Ansible
- 2 Bases d'Ansible
- 3 Fichier inventaire
- 4 Commandes Ad-hoc
- 5 Playbook
- 6 Variables
- 7 Structures de contrôle
- 8 Étiquetage : Tags
- 9 Les rôles
- 10 Ansible Vault

---

# 1. Introduction à Ansible

---

- Qu'est-ce qu'Ansible
- Principes sur les outils de gestion de configuration
- Exemples d'Ansible
- Rapport avec le devops ?

# Qu'est-ce qu'Ansible

---

- Outil d'automatisation informatique sans agent développé en 2012 par Michael DeHaan, ancien employé de chez Red Hat.
- Objectifs: un code minime, cohérent, sécurisé, très fiable et facile à apprendre.
- La société Ansible a récemment été rachetée par Red Hat.
- Ansible fonctionne principalement en mode push en utilisant SSH.

# Qu'est-ce qu'Ansible

---

## ■ Automatisation des tâches d'admin

- Sur un parc de machines

## ■ Déployer des applications, configurer, gérer les services

## ■ Développement :

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

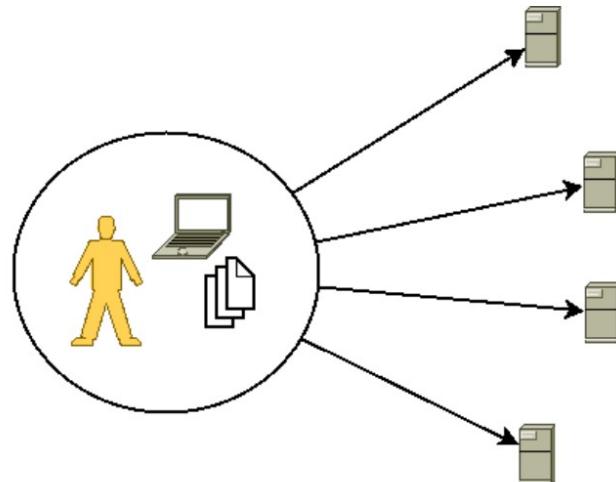
## ■ Écrit en python (les utilisateurs manipulent surtout du YAML)

GPLv3.

# Principes sur les outils de gestion de configuration

---

- “Recettes” de configuration centralisées, appliquées sur les machines (= noeuds)
  - Pas de config “à la main”



- Fournit des modules de haut niveau pour manipulations courantes (vs. script shell) :
  - Installation de paquets
  - gestion de services

# Principes sur les outils de gestion de configuration

---

- On décrit l'état souhaité plutôt que les étapes pour y arriver (vs. script shell)
  - Idempotence : une recette aboutit au même résultat qu'on l'applique une ou plusieurs fois
  - Les recettes doivent être écrites de cette façon pour que l'état soit stable

# Exemple ANSIBLE - 1

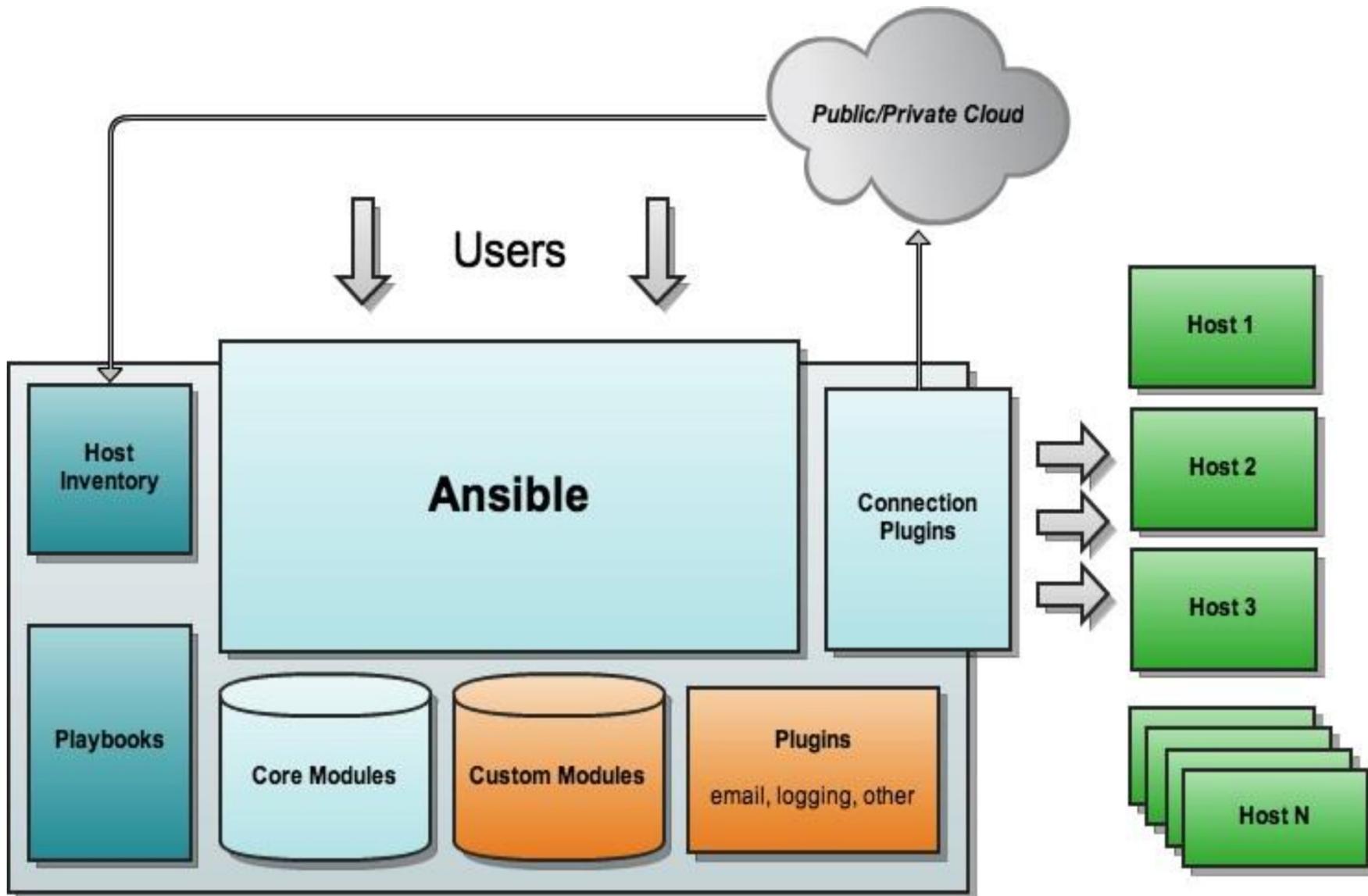
---

- L'objectif de ce logiciel est de maintenir une configuration identique sur différents serveurs/machines virtuelles.

Concrètement, Ansible permet :

- Installation des paquets.
- Installation des fichiers de configuration.
- Configuration des services.
- Réalisation de toutes les tâches d'administration possible par ssh.
- Couplage avec un outil de versioning pour, par exemple, suivre les évolutions du cluster.
- Notion de playbooks qui définit un ensemble de règles/scripts à appliquer/valider.

# Exemple ANSIBLE - 2



# Avantages d'Ansible

---

## ■ Pourquoi Ansible est-il populaire?

- Efficace : sans agent, installation minimale, état désiré (aucun changement non nécessaire), architecture basée sur la technologie de diffusion personnalisée, ciblage facile basé sur des faits
- Rapide : Facile à apprendre/à se rappeler, langage déclaratif simple
- Évolutif : Peut gérer des milliers de noeuds, architecture modulaire extensible
- Sécuritaire : Transport au travers SSH
- Vaste communauté : des milliers de rôles sur Ansible Galaxy

# Ce qu'on a couvert

---

- Objectifs d'Ansible.
- Concepts et bases de l'automatisation.

---

## 2. Bases d'ansible

---

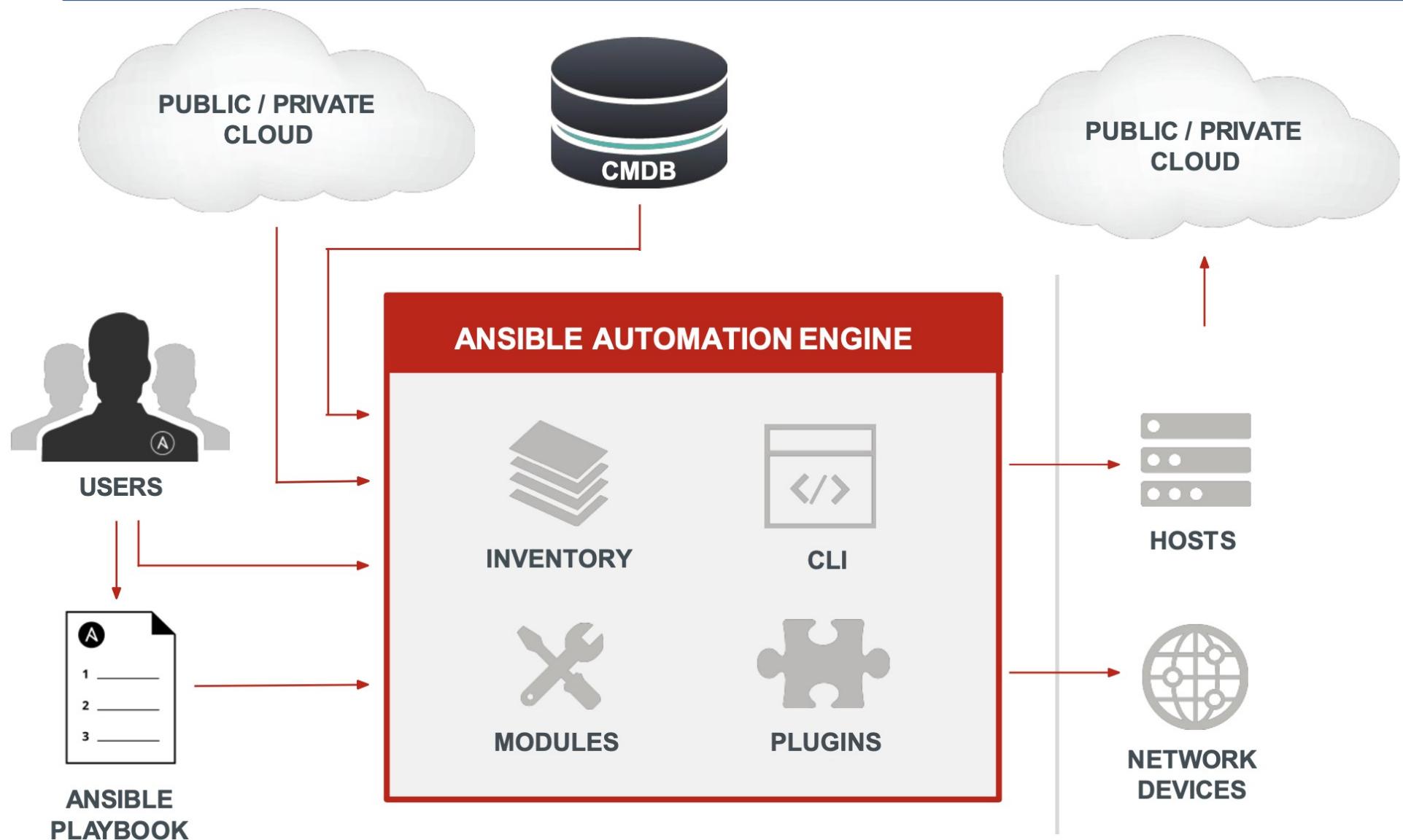
- Le vocabulaire d'Ansible
- Mécanisme d'automatisation d'Ansible
- Installation d'Ansible sur le serveur
- Configuration de Ansible
- Outils d'ansible
- Les différentes offres Ansible
- Les modules
- Les modules fréquents
- Documentation des modules
- Principe de l'idempotence

# Le vocabulaire d'Ansible

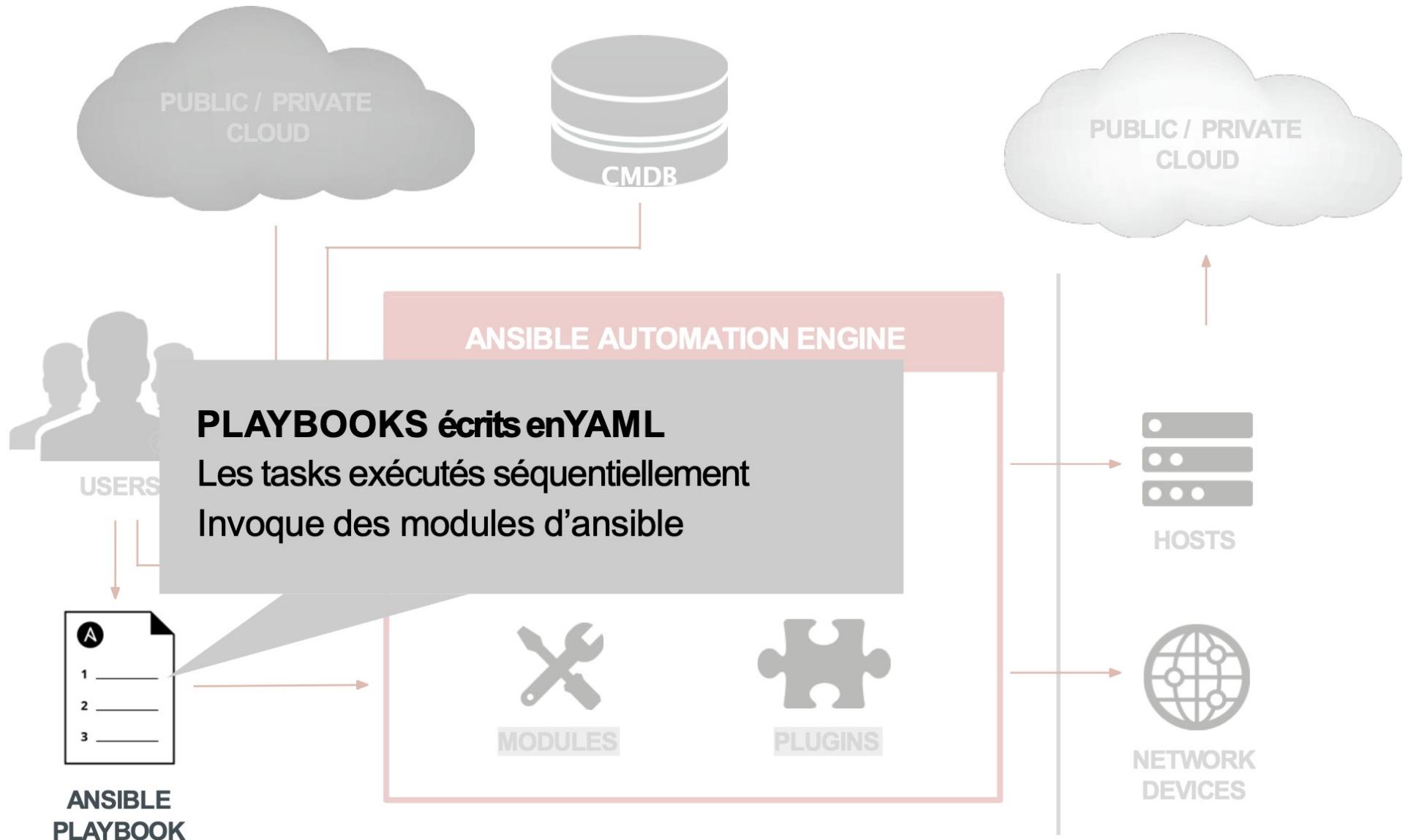
---

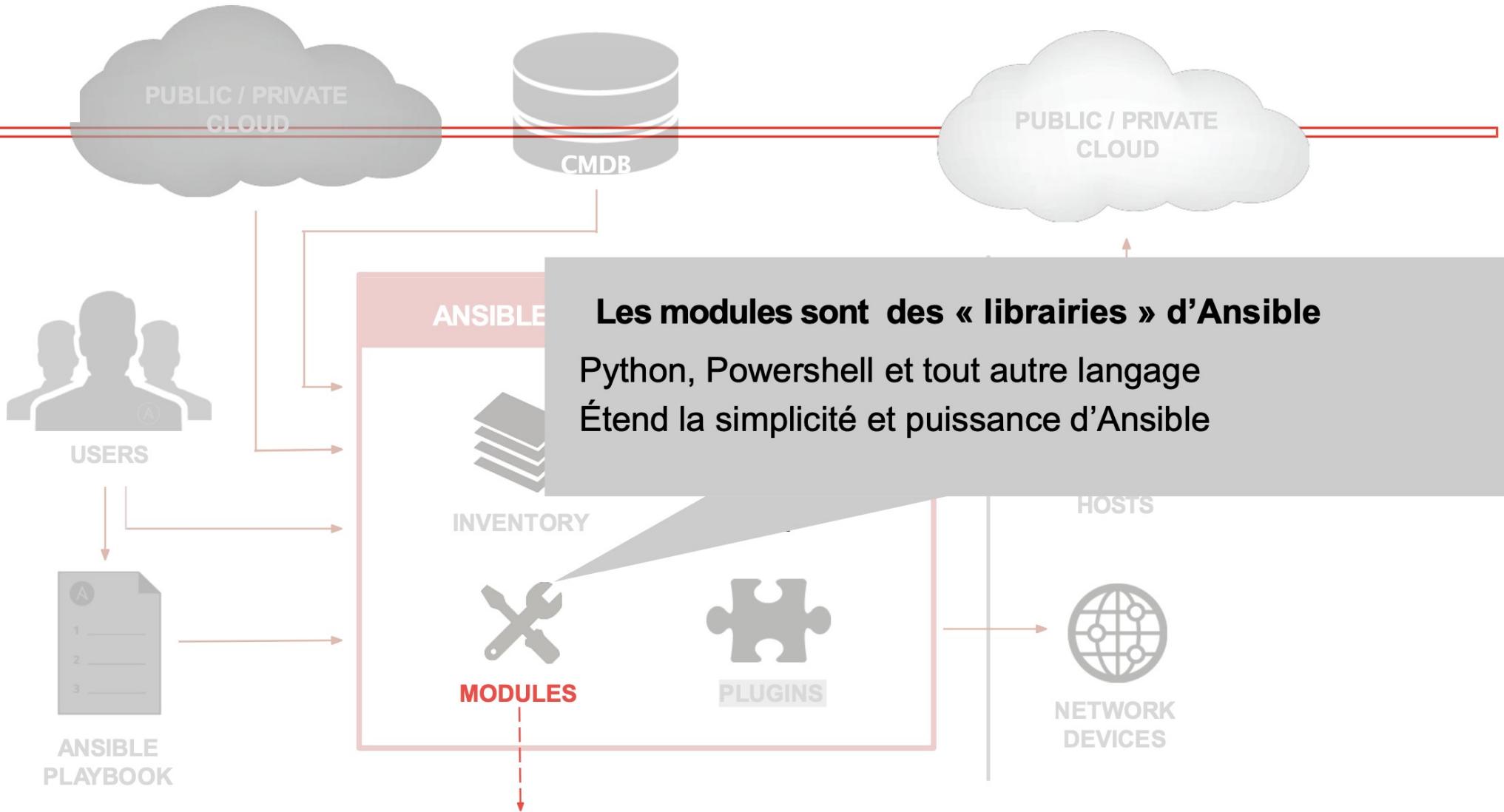
- Inventories
- Modules
- Variables
- Facts
- Playbooks and plays
- Configuration files
- Templates
- Handlers
- Rôles
- Ansible Vault

# Le vocabulaire d'Ansible



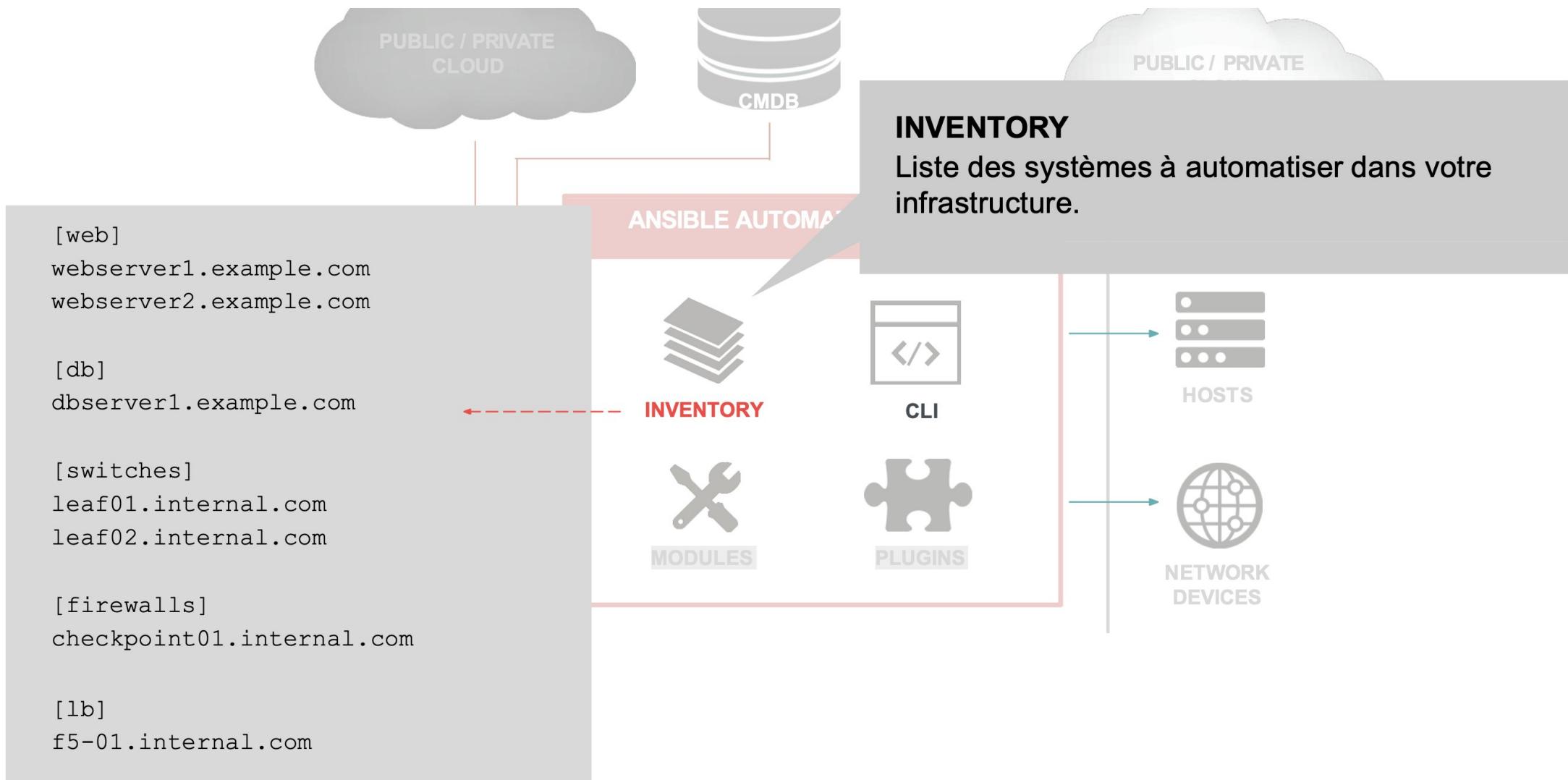
# Le vocabulaire d'Ansible



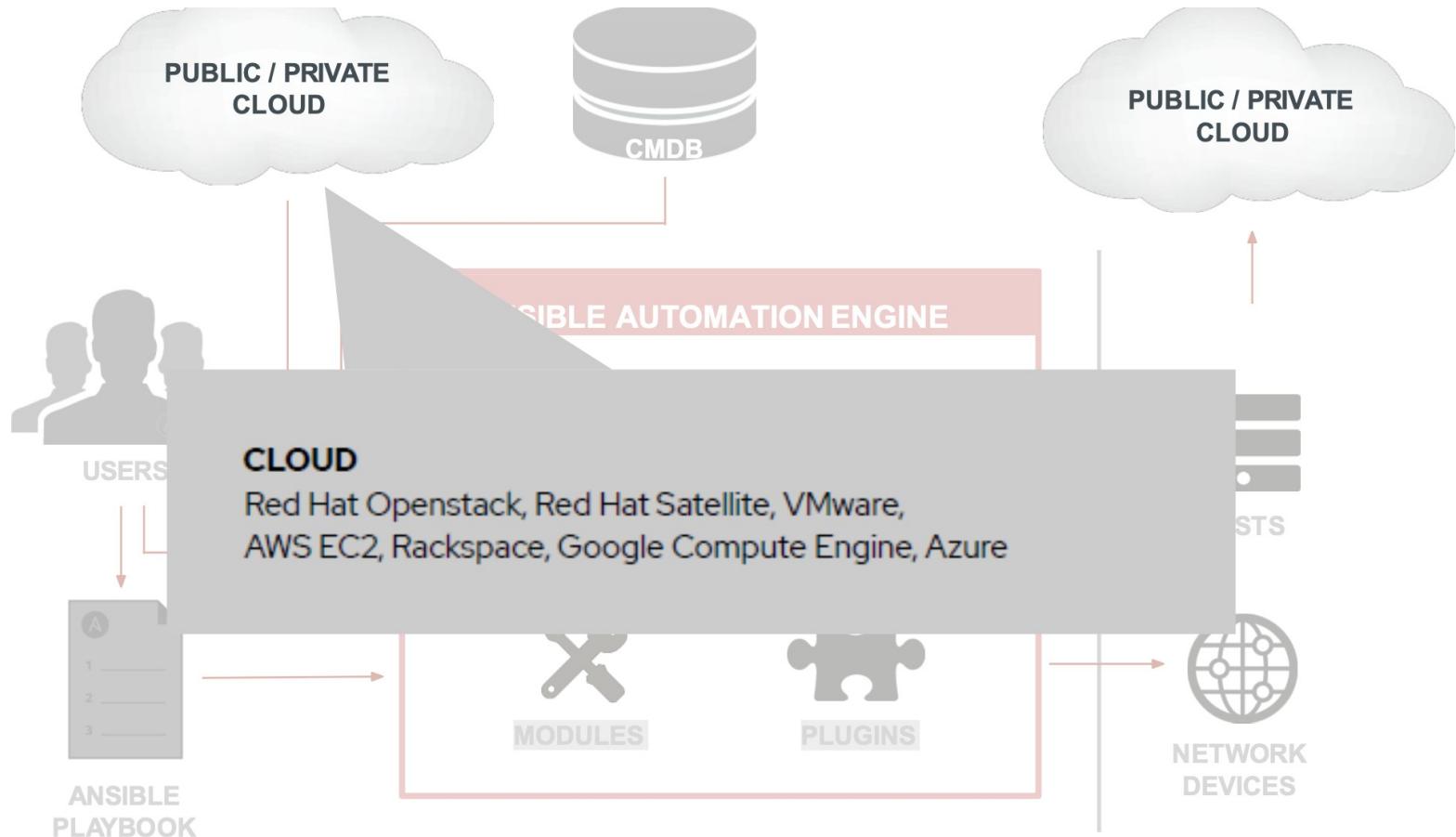


```
- name: copie index.html
copy:
  src: files/index.html
  dest: /var/www/html/
```

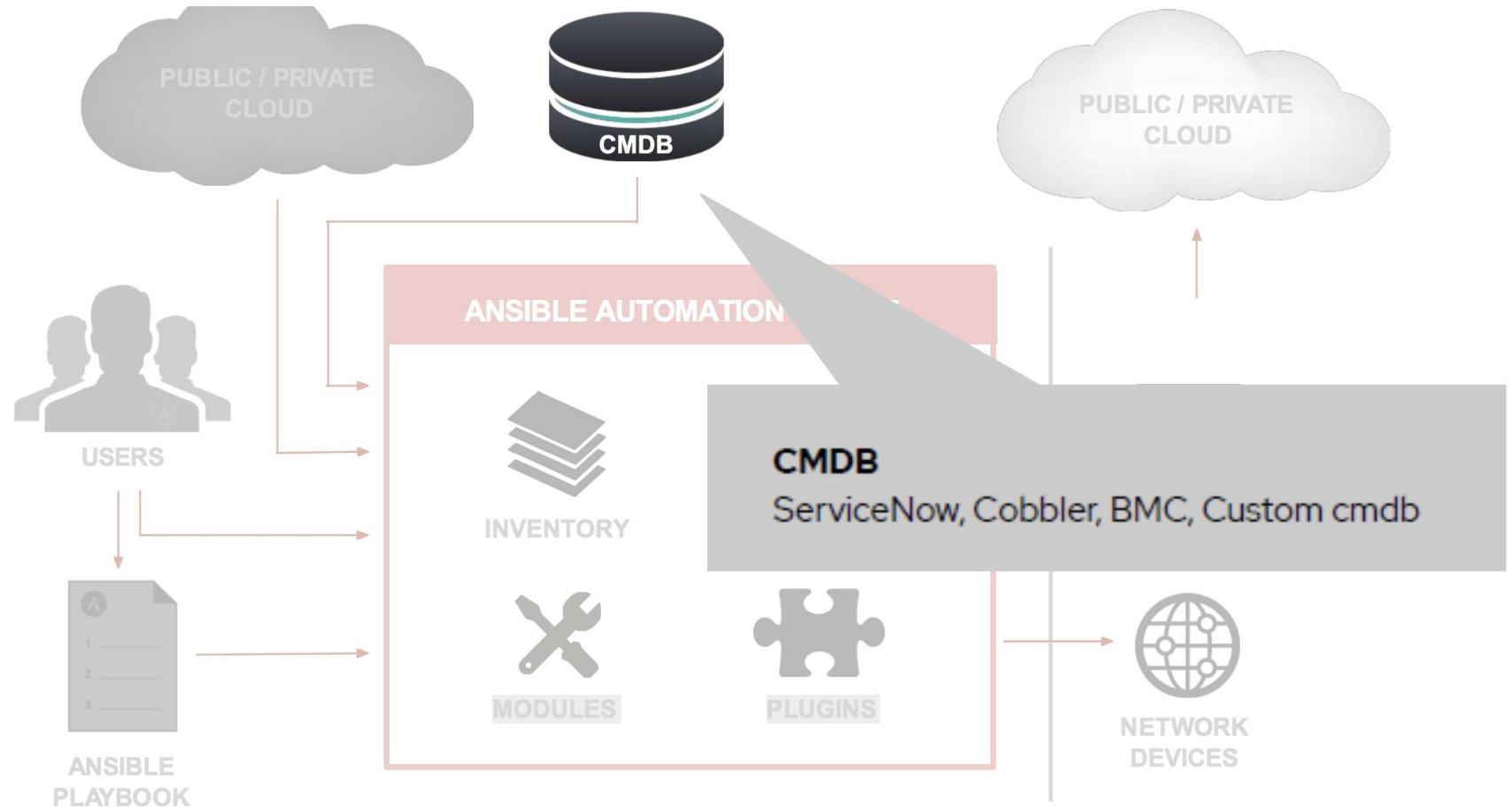
# Le vocabulaire d'Ansible



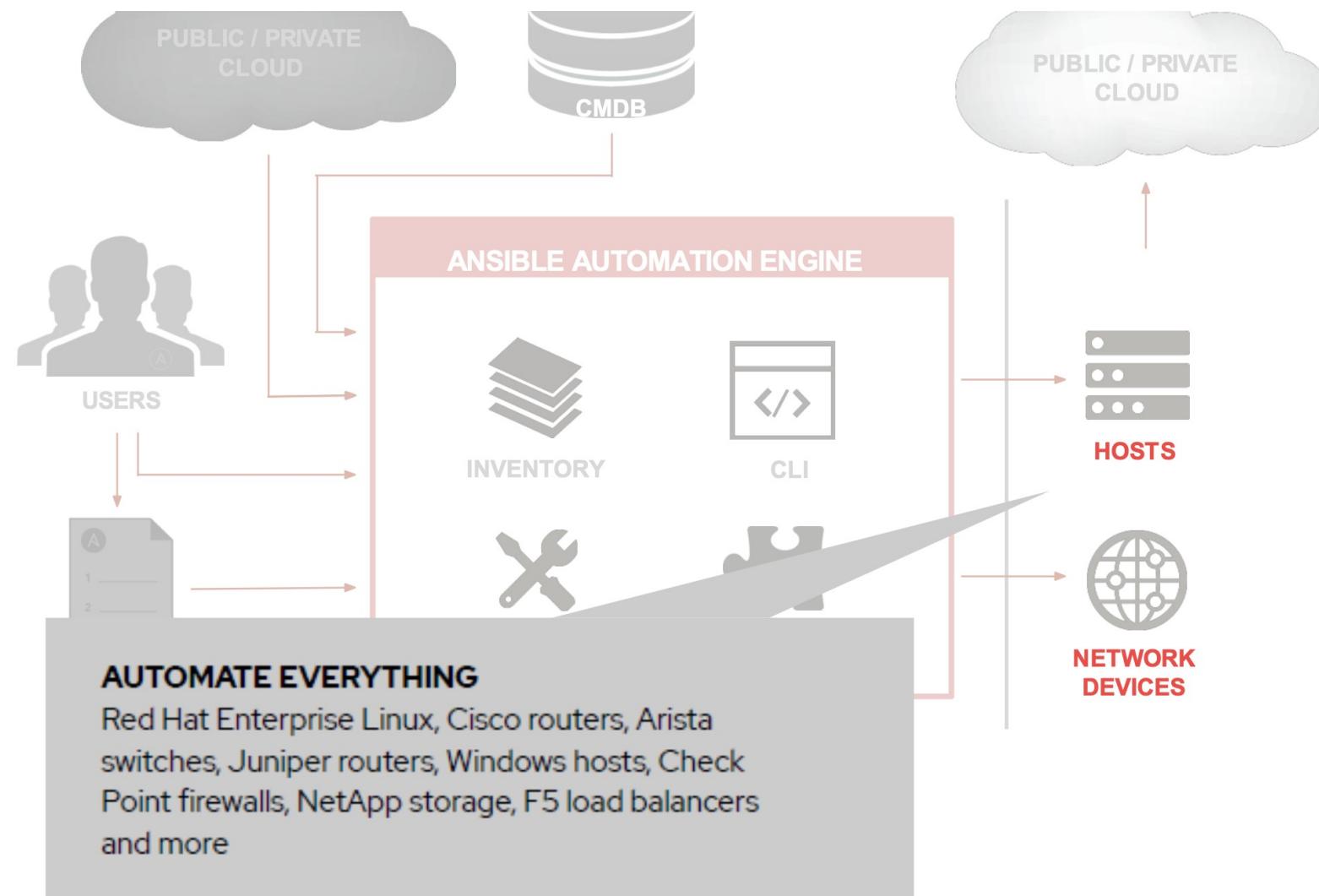
# Le vocabulaire d'Ansible



# Le vocabulaire d'Ansible

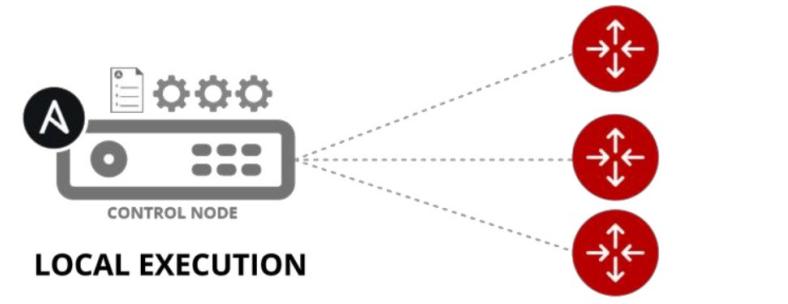


# Le vocabulaire d'Ansible



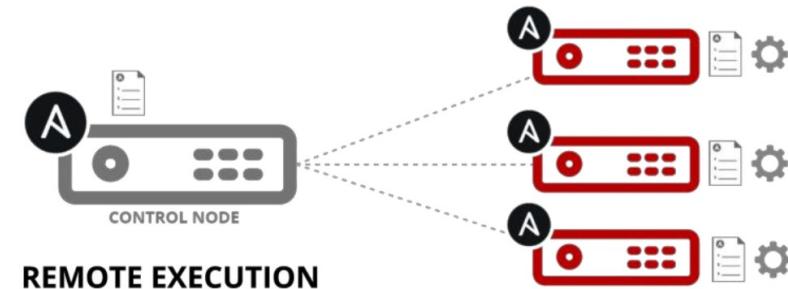
# Mécanisme d'automatisation d'Ansible

Le code du module est exécuté localement sur le nœud de contrôle



Équipements du réseau

Le code du module est copié sur le nœud cible, exécuté puis supprimé



Hôtes  
LINUX/WINDOWS

# Installation - Serveur avec apt

---

- Ubuntu/Debian:
  - \$ sudo apt install ansible
- RHEL
  - \$ sudo yum install ansible
- CentOS
  - \$ sudo yum install epel-release
  - \$ sudo yum install ansible
- Depuis les sources
  - \$ git clone <https://github.com/ansible/ansible.git>
  - \$ cd ./ansible
  - \$ source ./hacking/env-setup

# Configuration de Ansible

---

- Affichage de la configuration d'ansible:
  - *ansible-config list*
- On peut changer ces variables de configuration en renseignant un fichier de configuration. Ansible cherchera dans l'ordre (le premier trouvé sera utilisé et les autres seront ignorés) :
  - ANSIBLE\_CONFIG (Si la variable d'environnement est valorisée)
  - ./ansible.cfg (dans le dossier courant, le répertoire de travail)
  - ~/.ansible.cfg (à la racine du dossier utilisateur comme fichier caché)
  - /etc/ansible/ansible.cfg (dans le dossier de configuration du logiciel)

# Outils d'ansible

- ansible fournit plusieurs outils en ligne de commande

Util	Description
ansible	Execution d'une commande unique
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

Exemple :

**ansible -i hosts all -m ping**

- Bouts de code copiés sur le système cible.
- Exécutés pour satisfaire à la déclaration de tâche.
- Personnalisables.
- Les modules qui expédient avec Ansible sont tous écrit en Python, mais les modules peuvent être écrits en n'importe quel langage.

Vaste choix /force secrète d'Ansible...

Type	Exemples
Gestion du système	<code>user</code> (création des utilisateurs), <code>group</code> (gestion des groupes), etc.
Gestion des logiciels	<code>yum</code> , <code>apt</code> , <code>pip</code> , <code>npm</code>
Gestion des fichiers	<code>copy</code> , <code>fetch</code> , <code>lineinfile</code> , <code>template</code> , <code>archive</code>
Gestion des bases de données	<code>mysql</code> , <code>postgresql</code> , <code>redis</code>
Gestion du cloud	<code>amazon S3</code> , <code>cloudstack</code> , <code>openstack</code>
Gestion d'un cluster	<code>consul</code> , <code>zookeeper</code>
Envoyer des commandes	<code>shell</code> , <code>script</code> , <code>expect</code>
Gestion des messages	
Gestion du monitoring	
Gestion du réseau	<code>get_url</code>
Gestion des notifications	
Gestion des sources	<code>git</code> , <code>gitlab</code>

# Les modules fréquents

Outil	Description
ping	validation de l'inventaire
setup	retourne une liste d'informations matériels de l'hôte
shell /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/activation ou pas au boot
yum/apt/ zypper	permet de gérer l'installation, la mise à jour et la suppression de paquets.

La syntaxe est :

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

Exemple :

\$ ansible all –m yum –a "name=httpd state=latest"

# Documentation des modules

---

```
# AFFICHE TOUS LES MODULES  
ansible-doc -l
```

```
# ACCÉDER À LA DOCUMENTATION D'UN MODULE  
ansible-doc <module_name>
```

# Principe de l'idempotence

- L'idempotence signifie que vous pouvez faire quelque chose plusieurs fois et le résultat sera toujours le même.
- Un playbook est considéré comme idempotent si vous pouvez l'exécuter plusieurs fois et après la première exécution, la machine est dans un certain état qui ne change pas même si vous ré-exécutez le même playbook.

```
==> default: Running provisioner: ansible...
    default: Running ansible-playbook...

PLAY [all] ****
TASK [Gathering Facts] ****
ok: [default]

TASK [Make sure that we can connect to the machine] ****
ok: [default]

TASK [Install PHP] ****
ok: [default] => (item=[u'php5-cli', u'nginx', u'mysql-server-5.6'])

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

# Ce qu'on a couvert

---

- Installation et configuration d'Ansible.
- Composants de base d'Ansible.

---

### 3. Fichier inventaire

---

- Introduction
- Notion d'inventaire
- Inventaire – variable de communication
- Inventaire – variable de communication
- Inventaire – groupement de variables
- Inventaire – groupes de groupes
- Motifs utilisables

- Automatisation des tâches sur des machines clientes.
- Nécessité d'indiquer les informations et adresses des clients distants.

# Notion d'inventaire

---

- Peut être appelé à partir du répertoire courant ou via /etc/ansible/hosts
- Peut être appelé avec le flag -i <nom\_du\_fichier>
- Peut être dynamique, c'est à dire créé par programmation en injectant les résultats de programme d'inventaire dans un format JSON
- Variable **Inventory** : dans ansible.cfg

# Inventaire - variable de communication

---

- **ansible\_connection** : local, ssh
- **ansible\_ssh\_host** : Le nom de l'hôte distant
- **ansible\_ssh\_port** : Le numéro du port ssh si différent du 22
- **ansible\_ssh\_user** : Le nom d'utilisateur de la machine distante.
- **ansible\_ssh\_pass** : Mot de passe ssh nécessaire (non sécurisé).
- **ansible\_ssh\_private\_key\_file** : fichier de clé privée utilisé pour ssh.

# Inventaire - hôtes et groupes

---

```
localhost ansible_connection=local
```

```
[webservers]
```

```
web[1:5].example.com ansible_connection=ssh ansible_ssh_user=webadmin
```

```
[dbservers]
```

```
db[1:2].example.com ansible_connection=ssh ansible_ssh_user=dbadmin
```

# Inventaire - groupement de variables

---

```
[webservers]
web[1:5].example.com ansible_connection=ssh ansible_ssh_user=webadmin
[webservers:vars]
http_port=80
```

# Inventaire - groupes de groupes

---

[atlanta]

host1

host2

[raleigh]

host2

host3

[southeast:**children**]

Atlanta

craleigh

# Motifs utilisables

---

- Tous les hôtes de l'inventaire (**all** ou \*)
- Un nom d'hôte ou groupe (host1, webservers)
- Wildcards (192.168.1.\*)
- OR (host1:host2, webservers:dbservers)
- NOT (webservers:dbservers:!production)
- AND (webservers:dbservers:&staging)
- REGEX (~(web|db).\*\.\example\.com)

- Principe des inventaires.
- Hôtes et groupes.
- Variables des inventaires.

---

## 4. Commandes Ad-hoc

---

- Syntaxe des commandes Ad-Hoc
- COMMANDE Ad-Hoc - exemples

# Syntaxe des commandes Ad-Hoc

---

- \$ ansible {pattern} -m {module} -a "{options}" {flags}
  - **pattern** : Quel hôte?
  - **module** : Quel module ansible? (**command** par défaut)
  - **options** : Quelles options du module?
  - **flags** : Flags de la commande ansible:
    - -u {username}: Exécuter la commande en tant qu'un autre utilisateur (l'utilisateur courant par défaut)
    - --sudo: exécuter la commande via sudo
    - -K: Demande interactive du mot de passe de sudo
    - -U {username}: l'utilisateur a utiliser avec sudo autre que root
    - -i {file}: fichier inventaire si différent du /etc/ansible/hosts

# Syntaxe des commandes Ad-Hoc

- Pour exécuter votre première commande Ansible...

```
(sur quoi)          (module)  (arguments)  #
ansible all -i inventory -m module -a "uptime"

192.168.250.13 | success | rc=0 >>
 18:57:01 up 11:03,  1 user,  load average: 0.00, 0.01, 0.05

192.168.250.11 | success | rc=0 >>
 18:57:02 up 11:03,  1 user,  load average: 0.00, 0.01, 0.05
```

# COMMANDÉ Ad-Hoc - exemples

```
#Transfert de fichier
```

```
$ ansible all -m copy -a "src=/etc/hosts dest=/tmp/hosts"
```

```
# INSTALLER LE PAQUETAGE HTTPD
```

```
ansible web -i ./hosts -m yum -a "name=httpd state=present"
```

```
# DÉMARRER ET ACTIVER LE SERVICE HTTPD
```

```
ansible web -i ./hosts -m service -a "name=httpd enabled=yes state=started"
```

```
#Collecte des faits
```

```
$ ansible all -m setup
```

# Ce qu'on a couvert

---

- Principe des commandes ad-hoc.
- Syntaxes et options

---

## 5. Playbook

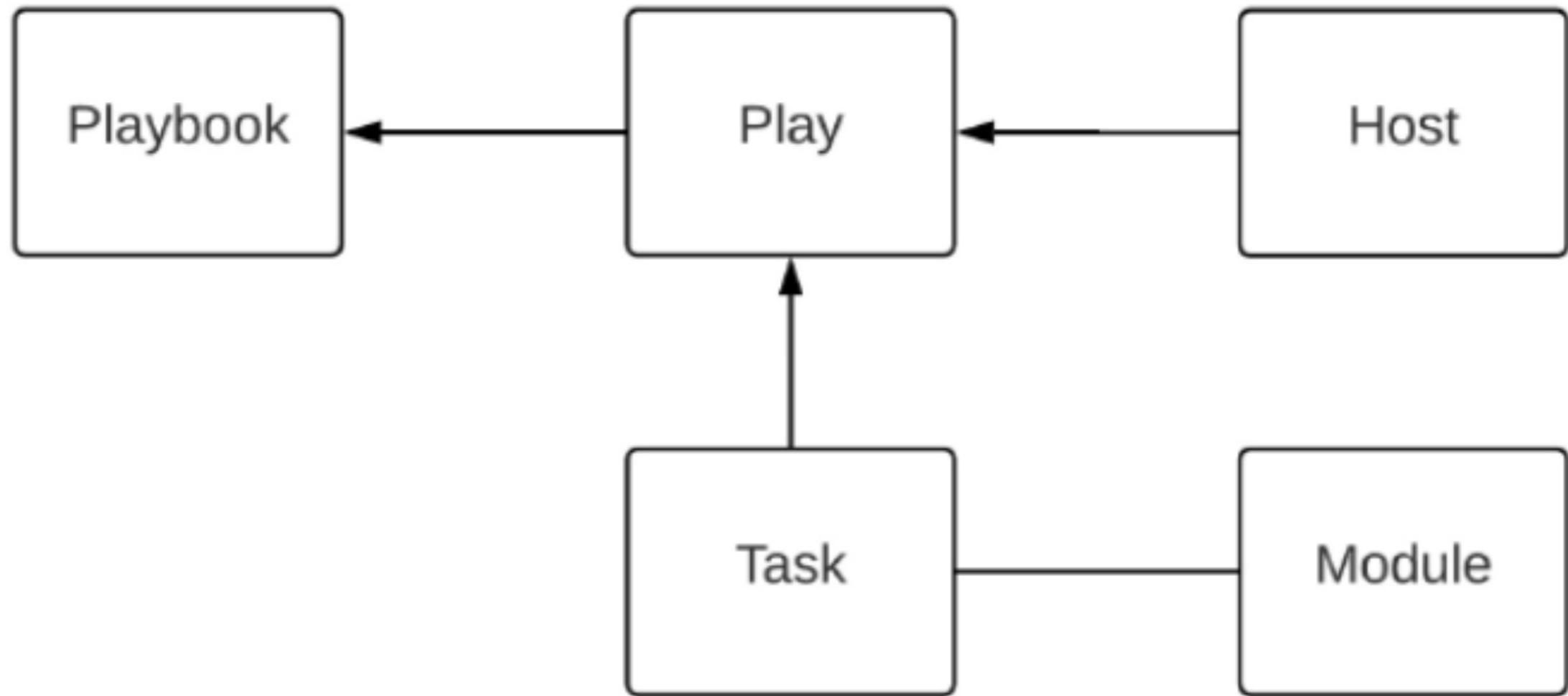
---

- Introduction
- Structure des playbooks
- Playbook – Inclusion de tâches
- Un « play » dans un « playbook »
- Eléments communs d'un play
- Exécution d'un playbook : codes couleurs
- Exécution d'un playbook
- Gestion des handlers
- Exécuter un playbook ansible

- Un playbook contient les instructions pour configurer vos hosts.
- Un playbook contient un ou plusieurs plays.
- Un play est une task.
- Les Playbooks sont au format YAML.
- Utilise un syntax minime.
- Fait pour être lisible.
- Doit être idempotent, c'est-à-dire exécutés plusieurs fois, le résultat sera le même.
- Peut-être divisés en template ou rôles.
- Plus efficaces pour être exécuté plusieurs tasks qu'en ligne de commande.

# Structure des playbooks

---



# Structure des playbooks



# Playbook - Inclusion de tâches

---

```
#playbook-with-tasks-included.yaml
- name: "PLAY 1: playbook with tasks included"
  hosts: localhost
  connection: local
  gather_facts: True
  vars:
    test: True
  tasks:
    - name: Include task list in play
      include_tasks: some-tasks.yaml
```

# Un « play » dans un « playbook »

Play 1

```
- hosts: db
  vars:
    software:
      - mariadb-server
  roles:
    - install_wordpress_db
```

Un autre  
play

```
- hosts: web
  vars:
    software:
      - httpd
      - php
  roles:
    - install_wordpress_web
```

# Un « play » dans un « playbook »

Play 1

```
- hosts: db
vars:
  software:
    - mariadb-server
roles:
  - install_wordpress_db

- hosts: web
vars:
  software:
    - httpd
    - php
roles:
  - install_wordpress_web
```

Un autre  
play

# Un « play » dans un « playbook »

Play 1

Un autre  
play

```
- hosts: db
vars:
  software:
    - mariadb-server
roles:
  - install_wordpress_db

- hosts: web
vars:
  software:
    - httpd
  - php
roles:
  - install_wordpress_web
```

# Eléments communs d'un play

- Ceci n'est pas une liste exhaustive mais contient les éléments couramment utilisés dans un play

<b>hosts</b>	La déclaration de la liste des hôtes/ groups concernés par le play
<b>connection</b>	Permet de changer le plugin de connexion utilisé pour les tasks à exécuter
<b>port</b>	Permet de surcharger le port par défaut de la connexion
<b>remote_user</b>	Utilisateur distant : surcharge l'utilisateur indiqué dans l'inventaire
<b>become</b>	Booléenne contrôlant le gain de privilège sur la machine distante lors de l'exécution du task.  (avec <code>become_user</code> , <code>become_method</code> )

# Eléments communs d'un play

## Traitement de l'information

<b>name</b>	Identifiant: peut être utilisé pour la documentation, tâches ou handlers
<b>gather_facts</b>	Booléenne permettant de bypasser la collecte des faits afin d'accélérer. Ce contenu peut être retrouvé avec le module setup
<b>no_log</b>	booléenne contrôlant la journalisation de l'information.
<b>ignore_errors</b>	Booléenne; si vrai, ignore les erreurs sauf celles bien fatales pour l'exécution du playbook.
<b>check_mode</b>	Connu sous le nom « dry run », évalue l'exécution sans exécuter réellement.

# Eléments communs d'un play

## ■ Traitement de l'inventaire

### **order**

Contrôle l'ordre des hôtes pour l'exécution du playbook.

## ■ Traitement des variables

### **vars**

Dictionnaire des variables

### **vars\_files**

Liste de fichiers contenant des variables à inclure.

### **vars\_prompt**

Liste de variables à saisir au clavier.

# Eléments communs d'un play

## ■ Traitement des tasks:

<b>pre_tasks</b>	Une liste de tâches à exécuter avec les tâches.
<b>roles</b>	Liste de rôles à importer dans le play.
<b>tasks</b>	Tâches principales à exécuter dans le play; elles s'exécutent après roles et avant post_tasks
<b>post_tasks</b>	Une liste de tâches qui s'exécutent après la section des tâches.
<b>handlers</b>	Des tâches s'exécutant suite à une notification par d'autres tâches.

# Eléments communs d'un play

```
- name: install a LAMP stack
  hosts: web,db,appserver01
  become: yes
  vars:
    my_greeting: Welcome to my awesome page
    favorite_food: fried pickles

  roles:
    - install_lamp_elements

  tasks:
    - name: write the index file
      copy:
        content: "{{ my_greeting }}. Enjoy some {{ favorite_food }}"
        dest: /var/www/html/index.html
      notify: reload_apache

  handlers:
    - name: reload_apache
      service:
        name: httpd
        state: reloaded
```

# Eléments communs d'un play

```
- name: install a LAMP stack
  hosts: web,db,appserver01
  become: yes
  vars:
    my_greeting: Welcome to my awesome page
    favorite_food: fried pickles

  roles:
    - install_lamp_elements

  tasks:
    - name: write the index file
      copy:
        content: "{{ my_greeting }}. Enjoy some {{ favorite_food }}"
        dest: /var/www/html/index.html
        notify: reload_apache

  handlers:
    - name: reload_apache
      service:
        name: httpd
        state: reloaded
```

# Eléments communs d'un play

```
tasks:  
- name: Ensure httpd package is present  
  yum:  
    name: httpd  
    state: latest  
  
- name: Ensure latest index.html file is present  
  copy:  
    src: files/index.html  
    dest: /var/www/html/  
  
- name: Restart httpd  
  service:  
    name: httpd  
    state: restart
```

# Eléments communs d'un play

```
tasks:  
- name: Ensure httpd package is present  
  
yum:  
    name: httpd  
    state: latest  
  
- name: Ensure latest index.html file is present  
copy:  
    src: files/index.html  
    dest: /var/www/html/  
  
- name: Restart httpd  
service:  
    name: httpd  
    state: restart
```

# Eléments communs d'un play

---

## tasks :

- **name: Ensure httpd package is present**  
**yum:**  
    **name:** httpd  
    **state:** latest
- **name: Ensure latest index.html file is present**  
**copy:**  
    **src:** files/index.html  
    **dest:** /var/www/html/
- **name: Restart httpd**  
**service:**  
    **name:** httpd  
    **state:** restart

# Eléments communs d'un play

```
tasks:  
- name: Ensure httpd package is present  
  yum:  
    name: httpd  
    state: latest  
  
- name: Ensure latest index.html file is present  
  copy:  
    src: files/index.html  
    dest: /var/www/html/  
  
- name: Restart httpd  
  service:  
    name: httpd  
    state: restart
```

# Exécution d'un playbook : codes couleurs

---

**Vert**: Une tâche exécutée correctement sans effectuer aucun changement.

**Jaune**: Une tâche exécutée correctement et introduit des changements.

**Blanc**: Information générale et entêtes.

**Bleu**: Une tâche conditionnelle ignorée.

**Violet**: Un bug ou avertissement obsolète.

**Rouge**: Une tâche échouée.

# Exécution d'un playbook

```
[user@ansible] $ ansible-playbook apache.yml

PLAY [webservers]
*****
TASK [Gathering Facts]
*****
ok: [web1]
ok: [web2]
ok: [web3]

TASK [Ensure httpd package is present]
*****
changed: [web1]
changed: [web2]
changed: [web3]

TASK [Ensure latest index.html file is present]
*****
changed: [web1]
changed: [web2]
changed: [web3]

TASK [Restart httpd]
*****
changed: [web1]
changed: [web2]
changed: [web3]

web2          ok=1    changed=3  unreachable=0  failed=0
:
web1          ok=1    changed=3  unreachable=0  failed=0
*****
```

# Exécution d'un playbook

```
[user@ansible] $ ansible-playbook apache.yml
```

```
PLAY [webservers]
```

```
*****  
TASK [Gathering Facts]
```

```
ok: [web1]  
ok: [web3]
```

```
TASK [Ensure httpd package is present]
```

```
changed: [web2]  
changed: [web1]  
changed: [web3]
```

```
TASK [Ensure latest index.html file is present]
```

```
changed: [web2]  
changed: [web1]  
changed: [web3]
```

```
TASK [Restart httpd]
```

```
changed: [web2]  
changed: [web1]  
changed: [web3]
```

```
PLAY RECAP
```

```
web2          ok=1      changed=3 unreachable=0 failed=0*****  
:  
web1          ok=1      changed=3 unreachable=0 failed=0
```

setup module

yum module

copy module

service module



# Gestion des handlers

- Un handler est exécuté lorsqu'une tâche le notifie suite à un changement.

```
tasks:  
- name: Ensure httpd package is present  
  yum:  
    name: httpd  
    state: latest  
    notify: restart_httpd  
  
handlers:  
- name: restart_httpd  
  service:  
    name: httpd  
    state: restarted
```

# Gestion des handlers

```
tasks:  
- name: Ensure httpd package is  
  present yum:  
  name: httpd  
  state: latest  
  notify: restart_httpd  
  
- name: Standardized index.html  
  file copy:  
  content: "This is my index.html file for {{ ansible_host }}"  
  dest: /var/www/html/index.html  
  notify: restart_httpd
```

Si **une** des tâches **notify** **un changement**, le handler sera notifié une seule fois.

```
TASK [Ensure httpd package is present]  
ok: [web2] ****  
ok: [web1] unchanged  
  
TASK [Standardized index.html file]  
changed: [web2] ****  
changed: [web1] changed  
  
NOTIFIED: [restart_httpd]  
***** changed: [web2]  
changed: [web1]
```

**handler runs once**

# Gestion des handlers

```
tasks:  
- name: Ensure httpd package is  
  present yum:  
    name: httpd  
    state: latest  
  notify: restart_httpd  
  
- name: Standardized index.html  
  file copy:  
    content: "This is my index.html file for {{ ansible_host }}"  
    dest: /var/www/html/index.html  
  notify: restart_httpd
```

Si les **deux** tâches notifie **un changement**, le handler sera notifié **une seule fois**.

```
TASK [Ensure httpd package is present]  
***** changed: [web2]  
changed: [web1] changed  
  
TASK [Standardized index.html file]  
***** changed: [web2]  
changed: [web1] changed  
  
NOTIFIED: [restart_httpd]  
***** changed: [web2]  
changed: [web1]
```

**handler runs once**

# Gestion des handlers

```
tasks:  
- name: Ensure httpd package is  
  present yum:  
  name: httpd  
  state: latest  
  notify: restart_httpd  
  
- name: Standardized index.html  
  file copy:  
  content: "This is my index.html file for {{ ansible_host }}"  
  dest: /var/www/html/index.html  
  notify: restart_httpd
```

Si **aucune** tâche  
notifie de  
**changement**, le  
handler ne sera pas  
notifié.

```
TASK [Ensure httpd package is present]  
*****  
ok: [web2] unchanged  
  
TASK [Standardized index.html file]  
*****  
ok: [web1] unchanged  
  
PLAY RECAP  
*****  
web2 : ok=2 changed=0 nreachable=0 failed=0 skipped=0 rescued=0 ignored=0  
web1 : ok=2 changed=0 nreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

# Execution d'un playbook Ansible

---

```
$ ansible-playbook play.yml -i inventory
```

# Execution d'un playbook Ansible

---

En mode vérification uniquement

```
$ ansible-playbook play.yml -i hosts -check
```

# Ce qu'on a couvert

---

- Principe des playbooks.
- Script d'exécution de tâches distantes.
- Différents parties et blocs.

---

## 6. Variables

---

- Introduction
- Saisie au clavier
- Variables internes et externes
- Récupération du résultat d'une commande
- Facts : Variables magiques

- Variables saisies au clavier.
- Variables définies dans le scripts.
- Variables incluses depuis des fichiers externes.
- Facts : Variables magiques

Définie dans le playbook par le module « vars\_prompt » :

```
- hosts: localhost
  vars_prompt:
    - name: "name"
      prompt: "Quel est votre prénom?"
      private: no
  tasks:
    - debug:
        msg: "Bonjour {{ name }} "
```

# Variables internes et externes

---

- Il est de bonne pratique d'organiser ses procédures avec des variables.
- Les variables Ansible peuvent être déclarées à différents endroits : dans
  - l'inventaire, dans les dossiers group\_vars/ ou host\_vars/ par exemple.
  - On peut aussi les définir via la ligne de commande : **ansible-playbook -e**
  - On peut les définir dans les playbook sous des forme diverses :
    - **en valorisation directe dans la tâche ou dans le jeu ou encore dans le rôle ( vars: )**
    - **en référence à un fichier ( vars\_files: )**
    - **en les "inclusant" ( include\_vars: )**
    - **en appelant des variables d'environnement**
    - **en important d'autres playbooks( import\_playbook: )**

# Variables internes et externes

---

- Il est de bonne pratique d'organiser ses procédures avec des variables.
- Les variables Ansible peuvent être déclarées à différents endroits : dans
  - l'inventaire, dans les dossiers group\_vars/ ou host\_vars/ par exemple.
  - On peut aussi les définir via la ligne de commande : **ansible-playbook -e**
  - On peut les définir dans les playbook sous des forme diverses :
    - **par génération ou récolte dynamique (gather\_facts: )**
    - **définies par des tâches du playbook ( set\_facts: )**
    - **définies à partir de la sortie standard d'une tâche ( register: )**
    - **définies à partir d'invites interactives ( vars\_prompt: ) • A partir des dossiers default/ ou variables/ d'un rôle**

# Variables internes et externes

---

- Définies dans le playbook
  - hosts: webservers
    - vars:**
      - http\_port: 80
- • Des variables par défaut des rôles: **{{ role }}/defaults/main.yml**
- • Variables incluses
  - 
  - hosts: all remote\_user: root vars:
    - favcolor: blue
  - vars\_files:**
    - /vars/external\_vars.yml

# Récupération du résultat d'une commande

Valeurs de retour	Signification de la valeurs de retour
changed (modifié)	Un booléen indiquant si la tâche a dû effectuer des modifications.
failed (échoué)	Un booléen qui indique si la tâche a échoué ou non.
skipped (évit�)	Un booléen qui indique si la tâche a �t� ignor�e ou non.
rc	Certains modules ex�utent des utilitaires en ligne de commande ou sont con�us pour ex�uter des commandes directement (raw, shell, commande, etc), ce champ contient le 'code de retour' de ces utilitaires.

# Récupération du résultat d'une commande

Valeurs de retour	Signification de la valeurs de retour
stderr	Certains modules exécutent des utilitaires en ligne de commande ou sont conçus pour exécuter des commandes directement (raw, shell, commande, etc), ce champ contient la sortie d'erreur de ces utilitaires.
stderr_lines	Lorsque <code>stderr</code> est retourné, nous fournissons aussi toujours ce champ qui est une liste de chaînes de caractères, un élément par ligne de l'original.
stdout	Certains modules exécutent des utilitaires en ligne de commande ou sont conçus pour exécuter directement des commandes (raw, shell, commande, etc). Cette zone contient l'édition normale de ces utilitaires.
stdout_lines	Lorsque <code>stdout</code> est retourné, Ansible fournit toujours une liste de chaînes de caractères, chacune contenant un élément par ligne de la sortie originale.

# Récupération du résultat d'une commande

---

```
- name: "print df -Th"
  hosts: localhost
  gather_facts: True
  tasks:
    - shell: df -h
      register: df_output
    - debug:
        msg: "{{ df_output.stdout_lines | list }}
```

# Facts : Variables magiques

---

- Les facts visent l'obtention des informations de vos hosts.
- Elles commencent par **ansible\_**
- Vous pouvez utiliser des Facts dans vos variables de playbooks.
- Le retour des informations fournies par les facts peut être bloqué pour améliorer les performances :
  - `hosts: mainserver`  
`gather_facts: no`

# Facts : Variables magiques

- Les facts sont fournies par le module setup:

```
---  
#print-out-operating-system.yml  
- name: print out operating system  
  hosts: localhost  
  gather_facts: False  
  tasks:  
    - name: "facts"  
      setup:  
      register: output  
    - name: "print output"  
      debug:  
      msg: "{{ output }}"
```

# Facts : Variables magiques

---

- Quelques variables magiques:

- ❑ ansible\_distribution
  - ❑ ansible\_distribution\_release
  - ❑ ansible\_distribution\_version
  - ❑ ansible\_fqdn ansible\_hostname
  - ❑ ansible\_os\_family
  - ❑ ansible\_pkg\_mgr
  - ❑ ansible\_default\_ipv4.address
  - ❑ ansible\_default\_ipv6.address

# Ce qu'on a couvert

---

- Types des variables utilisées par les playbooks.
- Variables saisies au clavier.
- Variables internes et externes.
- Variables magiques.

---

## 7. Structures de contrôle

---

- Introduction
- Structures conditionnelles
- Structures conditionnelles - exemple
- Condition selon un état précédent
- Structures des boucles
- Notion des blocs
- Gestion des exceptions
- Ignorer les tâches en échec

- Ansible intègre des structures complexes en plus des tâches simples.
- Des structures conditionnelles peuvent être écrites pour contrôler l'exécution d'une tâche spécifique.
- Des boucles sont fournies également pour parcourir une liste de valeurs.

# Structures conditionnelles

---

- Les conditions sont introduites à la suite de la tâche concernée via le mot clé « when »:
  - var == "Value" , var >= 5 , etc.
  - var , où var correspond à un booléen ( yes , true , True , TRUE )
  - var is defined , var is not defined , ! var is defined
  - <condition1> and <condition2>
  - <condition1> or <condition2>
  - (<condition1> and <condition2>) or (<condition3> and <condition4>)
  - Les facts peuvent être formulées comme éléments d'un tableau
  - **ansible\_facts:**  
`ansible_os_family : ansible_facts['os_family']`

```
- name: "playbook handler"
hosts: all
become: yes
vars_prompt:
  - name: "response"
    prompt: "tu veux executer?\n1-yes\n2-no"
    private: no
tasks:
  - name: "tache 1"
    command: "true"
    notify: print state
    when: response == "1"
  - name: "tache 2"
    debug:
      msg: "bye"
    when: response == "2"
handlers:
  - name: "print state"
    debug:
      msg: "parfait!"
```

# Condition selon un état précédent

---

- **name: Ensure httpd package is present**  
**yum:**  
    **name:** httpd  
    **state:** latest  
**register:** httpd\_results
  
- **name: Restart httpd**  
**service:**  
    **name:** httpd  
    **state:** restart  
**when:** httpd\_results.changed

# Structures des 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.

# Structures des boucles

- L'initiation de la boucle est effectuée en utilisant le mot clé «loop» :

```
- name: "loop illustration"
```

```
hosts: localhost
```

```
vars:
```

**fruits:**

```
- apple
```

```
- orange
```

```
- pineapple
```

```
tasks:
```

```
- name: "Print my Fruits"
```

```
debug:
```

```
msg: "{{ item }}"
```

**loop: "{{ fruits }}"**

# Structures des boucles

- L'initiation de la boucle est effectuée en utilisant le mot clé «loop» :

```
- name: "loop illustration"
hosts: localhost
vars:
fruits:
- apple
- orange
tasks:
- name: "loop with register"
shell: "echo {{ item }}"
loop: "{{ fruits }}"
register: echo
- name: "affichage resultats"
debug:
msg: "{{ echo }}
```

# Structures des boucles

- L'initiation de la boucle est effectuée en utilisant le mot clé «loop» :

```
- name: "loop illustration"
hosts: localhost
vars:
fruits:
- name: apple
color: green
- name: orange
color: orange
- name: pineapple
color: yellow
tasks:
- name: "Print my Fruits"
debug:
msg: "{{ item.name }} is {{ item.color }}"
loop: "{{ fruits }}"
```

# Structures des boucles

```
- name: "loop illustration"
hosts: localhost
vars:
  fruits:
    - id: 0
      name: apple
  colors:
    - id: 0
      name: green
tasks:
  - name: "Print my Fruits"
    debug:
      msg: "{{ item.0.name }} is {{ item.1.name }}"
    loop: "{{ fruits|product(colors)|list }}"
```

# Notion des blocs

---

- Les blocs permettent le regroupement logique des tâches et la gestion des erreurs dans le jeu.
- Tout ce que l'on peut appliquer à une tâche unique peut être appliqué au niveau du bloc ce qui facilite également la définition de données ou de directives communes aux tâches.
- Les directives seront appliquée aux tâches, mais pas au bloc lui-même.

```
tasks:
```

```
- name: Install Apache
```

```
block:
```

```
- yum:
```

```
    name:
```

```
        - httpd
```

```
        - memcached
```

```
    state: installed
```

```
- service:
```

```
    name: bar
```

```
    state: started
```

```
    enabled: True
```

```
when: ansible_distribution == 'CentOS'
```

```
become: true
```

```
become_user: root
```

# Gestion des exceptions

---

- Les blocs permettent également de gérer les sorties d'erreurs de manière similaire aux exceptions de la plupart des langages de programmation.
- Les tâches du block: s'exécutent normalement.
- En cas d'erreur, la section rescue: s'exécute avec tout ce que vous devez faire pour résoudre l'erreur précédente.
- La section always: est exécutée quelle que soit l'erreur précédente qui s'est produite ou non dans les sections de block: et rescue:

# Gestion des exceptions

tasks:

- name: Attempt and graceful roll back demo
- block:**
- debug:
    - msg: 'I execute normally'
  - command: /bin/false
  - debug:
    - msg: 'I never execute, due to the above task failing'
- rescue:**
- debug:
    - msg: 'I caught an error'
  - command: /bin/false
  - debug:
    - msg: 'I also never execute :-('
- always:**
- debug:
    - msg: "This always executes"

# Ignorer les tâches échouées

---

- Une tâche qui échoue (failed) arrête la lecture du livre de jeu.  
`ignore_errors` permet d'outrepasser ce comportement:
  - name: this will not be counted as a failure  
command: /bin/false  
**ignore\_errors: yes**

# Ce qu'on a couvert

---

- Boucles et structures conditionnelles.
- Blocks et gestion des erreurs.

---

## 8. Etiquetage : Tags

---

- Introduction
- Application des tags
- Exécution des tags
- Tags spéciaux
- Réutilisation des tags

- Si vous avez un grand playbook, il peut s'avérer utile de ne pouvoir en exécuter qu'une partie spécifique plutôt que de tout lire.
- Ansible prend en charge un attribut tags: pour cette raison.
- Lorsque vous exécutez un playbook, vous pouvez filtrer les tâches en fonction des "tags" de deux manières:
  - Sur la ligne de commande, avec les options **--tags** ou **--skip-tags**
  - Dans les paramètres de configuration Ansible, avec les options **TAGS\_RUN** et **TAGS\_SKIP**

# Application des tags

---

- Les "tags" peuvent être appliqués à de nombreuses structures dans Ansible :
  - tasks:
  - roles:
  - blocks:
  - import\_roles:
  - import\_tasks:

## ■ Exemple d'utilisation d'un tag

tasks:

```
- yum:  
    name={{ item }}  
    state=installed  
with_items:  
    - httpd  
    - memcached  
tags:  
    - packages  
  
- template:  
    src=templates/src.j2  
    dest=/etc/foo.conf  
tags:  
    - configuration
```

# Exécution des tags

## ■ Exécuter avec des tags

```
ansible-playbook example.yml --tags "configuration"
```

```
ansible-playbook example.yml --skip-tags "notification"
```

```
Ansible-playbook example.yml -tags "configuration,notification"
```

- "**always**" : toujours exécuté tant qu'il n'est explicitement évité ( --skip-tags always )
- "**never**" : jamais exécuté tant qu'il n'est explicitement mentionné ( --tags never )
- "**tagged**" : exécute uniquement les tâches balisées
- "**untagged**" : exécute uniquement les tâches non balisées.
- "**all**" : exécute toutes les tâches balisées ou non

```
ansible-playbook example.yml --tags "tagged"
```

```
ansible-playbook example.yml --tags "untagged"
```

```
ansible-playbook example.yml --tags "all"
```

# Réutilisation des tags

---

- Il est possible d'appliquer le même "tag" à plusieurs tâches.
- Lors de l'exécution d'une playbook à l'aide de l'option de ligne de commande --tags , toutes les tâches portant ce nom de "tag" seront exécutées.

# Réutilisation des tags

- Cet exemple balise plusieurs tâches avec un "tag" "ntp" :

```
---
# file: roles/common/tasks/main.yml

- name: be sure ntp is installed
  yum:
    name: ntp
    state: installed
  tags: ntp

- name: be sure ntp is configured
  template:
    src: ntp.conf.j2
    dest: /etc/ntp.conf
  notify:
    - restart ntpd
  tags: ntp

- name: be sure ntpd is running and enabled
  service:
    name: ntpd
    state: started
    enabled: yes
  tags: ntp
```

# Ce qu'on a couvert

---

- Etiquetage des blocs et tâches.
- Exécution du playbook par parties selon les tags indiqués.

## 9. Les rôles

- Gestion des rôles
- Ansible Galaxy

# Gestion des rôles

---

- Un playbook est un unique fichier qui permet à Ansible d'installer vos hosts.
- Un Rôle pour être vu comme un fichier playbook divisé en plusieurs fichiers:
  - un pour les tasks, un autre pour les variables, un autre pour les handlers et ainsi de suite.
- Les rôles ajoutent de la modularité dans vos playbooks, vous pouvez les réutiliser en faisant des includes de rôles dans vos playbooks.
- Ansible galaxy est un repository, dans le cloud, de rôles déjà définis par une communauté de développeurs.

Structure du répertoire

## rôles

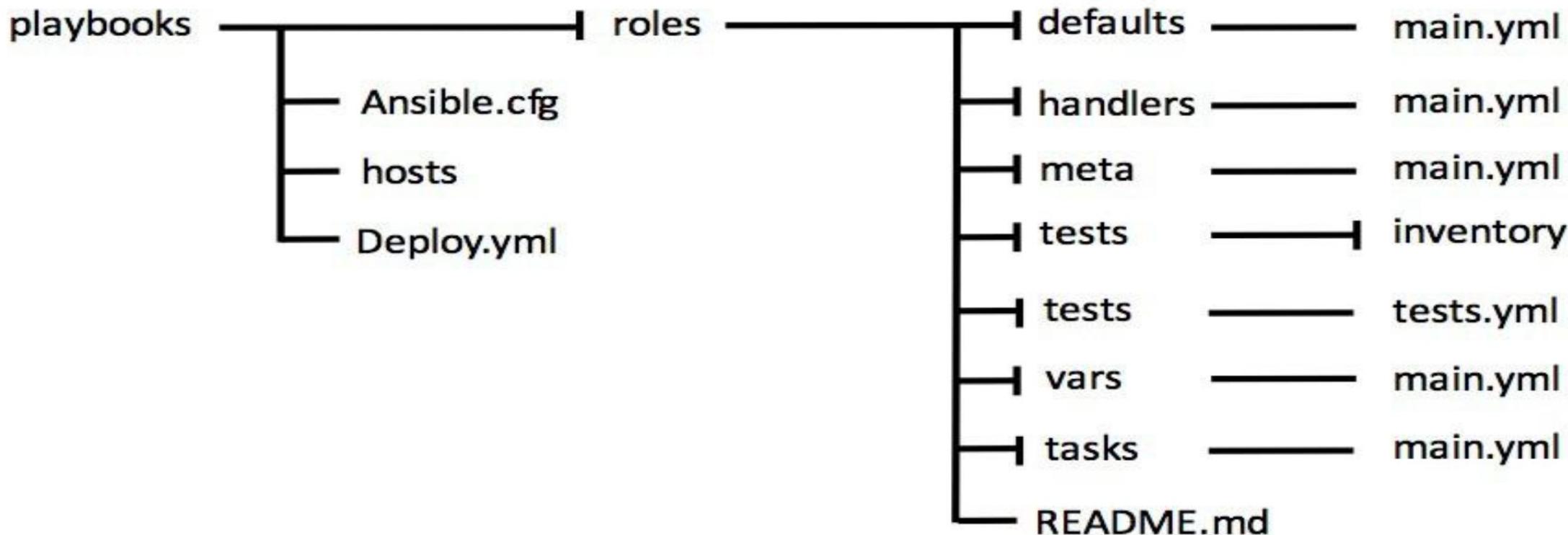
```
└── myapp
    ├── defaults
    ├── files
    ├── handlers
    ├── meta
    ├── tasks
    ├── templates
    └── vars
```

Pour créer automatiquement une structure de fichiers

```
ansible-galaxy init <role_name>
```

# Gestion des rôles

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



- Grâce à cette arborescence il ne sera pas nécessaire de préciser les directives vars, tasks ou handlers dans les fichiers vars/main.yml, tasks/main.yml et handlers/main.yml
- cette information sera obtenue à partir du nom du répertoire du rôle.

- Par exemple un fichier *tasks/main.yml* ne contiendra qu'une liste de tâches.
- De la même façon, les modules *copy* et *template* iront chercher les fichiers sources directement dans les répertoires *files* et *templates*, sans besoin de les préciser dans le path.

Exemple : un fichier *tasks/main.yml* ne contiendra qu'une liste de tâches

```
- name: Install Apache
  apt:
    name=apache2
    state=present
- name: Ensure service is registered and running service:
  name=apache2
  enabled=true
  state=started
```

## ■ Exemples de playbook

```
---
```

- **hosts: webservers**
- roles:**
  - **common**
  - **webservers**

## ■ Exemples de playbook

```
---  
- hosts: webservers  
  roles:  
    - common  
    - { role: myapp, dir: '/opt/a', port: 5000 }  
    - { role: myapp, dir: '/opt/b', port: 5001 }
```

## ■ Exemples de playbook

```
---
- hosts: webservers
  roles:
    - { role: foo, when: "ansible_os_family == 'RedHat'" }
```

# Ansible Galaxy

---



<https://galaxy.ansible.com>



ABOUT EXPLORE

BROWSE ROLES

BROWSE AUTHORS

SIGN IN

BROWSE ROLES

Keyword ▾



SORT Relevance ▾

<b>mysql</b> <span style="float: right;">1506</span> ansible role for mysql Type: Ansible Author: bennojoy Platforms: Enterprise_Linux, Fedora, Ubuntu Tags: database, sql Last Commit: NA Last Import: NA <a href="#">Watch 21</a> <a href="#">★ Star 117</a>	<b>nginx</b> <span style="float: right;">1299</span> ansible role nginx Type: Ansible Author: bennojoy Platforms: Enterprise_Linux, Fedora, Ubuntu Tags: web Last Commit: NA Last Import: NA <a href="#">Watch 17</a> <a href="#">★ Star 87</a>	<b>network_interface</b> <span style="float: right;">609</span> role for system network configuration Type: Ansible Author: bennojoy Platforms: Enterprise_Linux, Fedora, Ubuntu Tags: development, networking, system Last Commit: NA Last Import: NA <a href="#">Watch 12</a> <a href="#">★ Star 55</a>	<b>ntp</b> <span style="float: right;">9761</span> ansible role ntp Type: Ansible Author: bennojoy Platforms: Enterprise_Linux, Fedora, Ubuntu Tags: development Last Commit: NA Last Import: NA <a href="#">Watch 8</a> <a href="#">★ Star 23</a>
<b>memcached</b> <span style="float: right;">600</span> ansible role memcached Type: Ansible Author: bennojoy Platforms: Enterprise_Linux, Fedora, Ubuntu Tags: web Last Commit: NA Last Import: NA <a href="#">Watch 5</a> <a href="#">★ Star 10</a>	<b>redis</b> <span style="float: right;">175</span> ansible role for configuring redis Type: Ansible Author: bennojoy Platforms: Enterprise_Linux, Ubuntu Tags: web Last Commit: NA Last Import: NA <a href="#">Watch 0</a> <a href="#">★ Star 0</a>	<b>openldap_server</b> <span style="float: right;">1069</span> ansible role openldap server Type: Ansible Author: bennojoy Platforms: Enterprise_Linux, Fedora, Ubuntu Tags: development, system Last Commit: NA Last Import: NA <a href="#">Watch 8</a> <a href="#">★ Star 0</a>	<b>kerberos_server</b> <span style="float: right;">59</span> ansible role for configuring kerberos server Type: Ansible Author: bennojoy Platforms: Enterprise_Linux, Fedora, Ubuntu Tags: development Last Commit: NA Last Import: NA <a href="#">Watch 2</a> <a href="#">★ Star 4</a>
<b>kerberos_client</b> <span style="float: right;">56</span> ansible role for configuring kerberos client Type: Ansible Author: bennojoy Platforms: Enterprise_Linux, Fedora, Ubuntu Tags: development Last Commit: NA <a href="#">Watch 0</a> <a href="#">★ Star 0</a>	<b>cowsay</b> <span style="float: right;">152</span> Install Ansible-most-important-prerequisite software. This can be used from ansible-galaxy. Type: Ansible Author: r_rudi Platforms: Ubuntu Tags: development Last Commit: NA <a href="#">Watch 0</a> <a href="#">★ Star 0</a>		

POPULAR TAGS

system	4110
development	2143
web	1831
monitoring	839
networking	736
database	715
cloud	622
packaging	604
ubuntu	336
docker	299
centos	281

```
# ansible-galaxy search 'install git' --platforms el
Found 176 roles matching your search:
Name...
# ansible-galaxy install davidkarban.git -p roles
# ansible-galaxy list -p roles
# ansible-galaxy remove -p roles
```

# Ce qu'on a couvert

---

- Notion des rôles.
- Manipulation des rôles.
- Ansible Galaxy.

# 10. Ansible Vault

- Introduction
- ansible-vault
- Cryptage avec ansible-vault
- Exécution d'Ansible avec des variables cryptées
- Ansible-vault – autres commandes
- Remarques supplémentaires

- Les playbooks accèdent souvent à des informations sensibles : mots de passe des administrateurs et bases de données, clés privées,...
- Ansible vault permet de sécuriser toute donnée sensible tout en permettant l'exécution du playbook.
- Si la clé est fournie, Ansible décrypte automatiquement les mots de passe cryptés par vault.

- Tout contenu crypté est incorporé de manière transparente dans Ansible avec vault.
- Ansible-vault permet le cryptage de toute donnée sensible sur le disque.
  - Toute commande de ansible et ansible-playbook supporte le décryptage du contenu crypté lors de l'exécution.

# Cryptage avec ansible-vault

---

- Ansible vault est typiquement utilisé pour crypter les fichiers des variables.
- Cryptage d'un fichier existant:
  - `ansible-vault encrypt defaults/main.yml`
- Création d'un fichier crypté:
  - `ansible-vault create defaults/extrayml`
- Ces commandes demanderont la saisie et la confirmation d'un mot de passe.

# Exécution d'Ansible avec des variables cryptées

---

- Demande du mot de passe de vault:

```
ansible-playbook --ask-vault-pass -i inventory_file  
some_playbook.yml
```

- Utilisation d'un fichier vault contenant le mot de passe:

```
echo "secret_password" > vault_password
```

```
ansible-playbook --vault-password-file=vault_password -i  
inventory_file some_playbook.yml
```

# Ansible-vault - autres commandes

---

- Modifier un fichier crypté:

```
ansible-vault edit defaults/main.yml
```

- Déchiffrer un fichier:

```
ansible-vault decrypt defaults/main.yml
```

- Afficher le contenu d'un fichier crypté (lecture seule):

```
ansible-vault view defaults/main.yml
```

# Ansible-vault - autres commandes

---

- Changer le mot de passe des fichiers cryptés:

```
ansible-vault rekey defaults/main.yml
```

- Vous serez appelés à indiquer le mot de passe courant avant de le modifier.
- Un message sera affiché pour indiquer le succès du processus du nouveau cryptage.

# Remarques supplémentaires

---

- Si le mot de passe est perdu, les données cryptées sont définitivement perdue.
- Si vous gérez les rôles d'Ansible avec git, vous remarquerez que seule l'ouverture d'un fichier crypté, sans le changer, nécessitera un nouveau git commit.
- Faire en sorte de ne crypter que les fichiers des variables pour une bonne optimisation.

# Ce qu'on a couvert

---

- Protection des données sensibles.
- Cryptage des données sensibles utilisées par Ansible.
- Ansible Vault: chiffrement et déchiffrement des fichiers sensibles.