



Ansible, Automatiser la gestion des Serveurs

Mawaki PAKOUPETE

\$ whoami ?

- Mon nom est ***Mawaki PAKOUPETE***
- Ingénieur Cloud & DevOps
 - ◆ Débutant dans l'administration système : Unix/Linux, Virtualisation...
 - ◆ Ingénieur systèmes & DevOps (Docker, Kubernetes, Cloud AWS, GCP, Azure...)
 - ◆ Ingénieur cloud travaillant pour une entreprise cloud qui utilise fortement les technologies d'automatisation
- Certifications : Elasticsearch, CKS, CKA, AWS, RHCE, RHCSA, VMware LPIC-1, ITIL
- Passionné par la formation depuis 7 ans

Ice Breaker

- Prénom & Nom
- Entreprise actuelle & Rôle
- Attentes de la formation



\$ need --help

Comment demander de l'aide ?

- Cours qui se veut interactif
- Pour toute question, levez la main et posez là
- N'importe quel moment
- Vous pouvez utiliser le Chat également

Objectif de la formation

- Connaître les caractéristiques et le fonctionnement d'Ansible
- Mettre en oeuvre 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

Contenu du Cours

Ansible - Fondamentaux:

- Qu'est-ce que Ansible ?
- Pourquoi Ansible ?
- Fonctionnement de Ansible
- Ansible et DevOps
- Ansible VS Autres outils DevOps (Terraform, Puppet, Chef, Saltstack...)
- laC avec Ansible
- Introduction à YAML

Ansible - Prise en main

- Architecture Ansible
- Fichiers de configurations
- Inventory
- Notions : Playbook, Play, Module, Task, Role...
- Syntaxe Yaml.
- Ansible Ad-hoc
- Commandes de base d'Ansible
- Modules Ansible les plus utilisés
- Configuration des noeuds : clés ssh

Lab 1:

- Installation et prise en main d'Ansible
- Configuration des noeuds client
- Inventaire & Commandes Ad hoc

Ansible - Playbooks

- Modules
- Gestion des Variables
- Notifications et Handlers
- Exécution step by step
- Saut de tasks
- Gestion des erreurs, dry-run
- Conditions, boucles et blocks

Lab2 :

- Écriture de playbooks pour l'administration des serveurs

Ansible - Notions avancées

- Rôles Ansible
- Includes & tags
- Ansible-galaxy
- Templating et variables
- Les prompts et les facts
- Lookups, Plugins
- Ansible Vault

Lab3 :

- Créer un rôle ansible pour une administration avancée des serveurs

Ansible - Pour aller plus loin

- Développer ses propres modules
- Ansible Tower
- Automatisation CI avec Ansible
- Combinaison Ansible + Terraform

Ansible - Fondamentaux

- Qu'est-ce que Ansible ?
- Pourquoi Ansible ?
- Fonctionnement de Ansible
- Ansible et DevOps
- Ansible VS Autres outils DevOps (Terraform, Puppet, Chef, Saltstack...)
- IaC avec Ansible
- Introduction à YAML

Qu'est-ce que Ansible ?

→ Moteur d'automatisation IT qui automatise :

- ◆ Gestion des configurations
- ◆ Déploiement d'applications
- ◆ Orchestration intra-service
- ◆ Le provisionnement du cloud



Pourquoi Ansible ?

→ Simple

- ◆ Facile à écrire, à lire, à maintenir et à faire évoluer - sans écrire de scripts ou de code personnalisé.

→ Rapide à apprendre et à configurer

- ◆ Utilisation d'un langage très simple : **YAML**, sous la forme de Playbooks Ansible) qui vous permet de décrire vos **tâches d'automatisation** d'une manière qui se rapproche du langage humain.

→ Efficace

- ◆ Ne nécessite pas d'agent personnalisé ou de logiciel à installer.
- ◆ Ansible fonctionne en se connectant à vos nœuds et en y poussant de petits programmes, appelés "modules Ansible".

→ Sécurisé

- ◆ Pas d'agent
- ◆ Fonctionne avec OpenSSH

Pourquoi Ansible : Scripts vs Ansible Playbook

```
#!/bin/bash
# Script to add a user to Linux system
if [ $(id -u) -eq 0 ]; then
    $username=johndoe
    read -s -p "Enter password : " password
    egrep "^$username" /etc/passwd >/dev/null
    if [ $? -eq 0 ]; then
        echo "$username exists!"
        exit 1
    else
        useradd -m -p $password $username
        [ $? -eq 0 ] && echo "User has been added
to system!" || echo "Failed to add a user!"
    fi
fi
```

VS

```
- hosts: all_my_web_servers_in_DR
tasks:
  - user:
      name: johndoe
```

Place de Ansible dans l'Écosystème DevOps

- **DevOps** ⇒ vise à améliorer la collaboration et l'efficacité entre les équipes de développement (Dev) et d'exploitation (Ops).
- **Rôle essentiel de Ansible** au sein de cet écosystème :
 - ◆ Facilitant le déploiement, la configuration et la gestion des infrastructures.
 - ◆ Intervient dans la chaîne d'approvisionnement en automatisant la mise en place et la configuration des environnements, des serveurs, et des applications.
 - ◆ Permet une approche "Infrastructure as Code" (IaC) où les configurations sont traitées comme du code, assurant la cohérence et réduisant les erreurs manuelles.
 - ◆ Permet de définir et maintenir les configurations des systèmes.



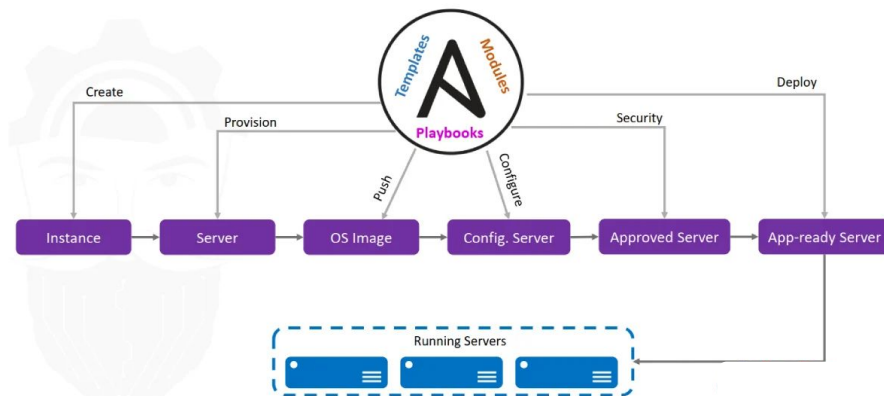
Place de Ansible dans l'Écosystème DevOps

→ **Rôle essentiel de Ansible** au sein de cet écosystème (Suite) :

- ◆ Ansible facilite l'intégration continue (CI) en automatisant les tests, la mise en place d'environnements de développement, et les déploiements pour assurer la qualité du code.
- ◆ Dans le cadre de la livraison continue (CD), Ansible permet des déploiements automatisés et cohérents, réduisant les délais et risques associés.
- ◆ Les "playbooks" d'Ansible décrivent les étapes nécessaires pour amener un système dans un état désiré, ce qui simplifie la gestion et la maintenance.
- ◆ Offre des capacités d'orchestration (comparable à Terraform à certains points)
- ◆ Les "playbooks" peuvent coordonner des tâches complexes impliquant plusieurs systèmes et services, assurant une exécution cohérente et ordonnée.

Avantages d'Ansible dans DevOps

- **Automatisation** : Réduit les tâches manuelles répétitives, minimisant les erreurs humaines.
- **Cohérence** : Assure une configuration uniforme et évite les déviations non intentionnelles.
- **Collaboration** : Facilite la collaboration entre équipes Dev et Ops en fournissant des processus standardisés.
- **Scalabilité** : S'adapte à des environnements diversifiés et à croissance rapide.



Ansible VS Terraform



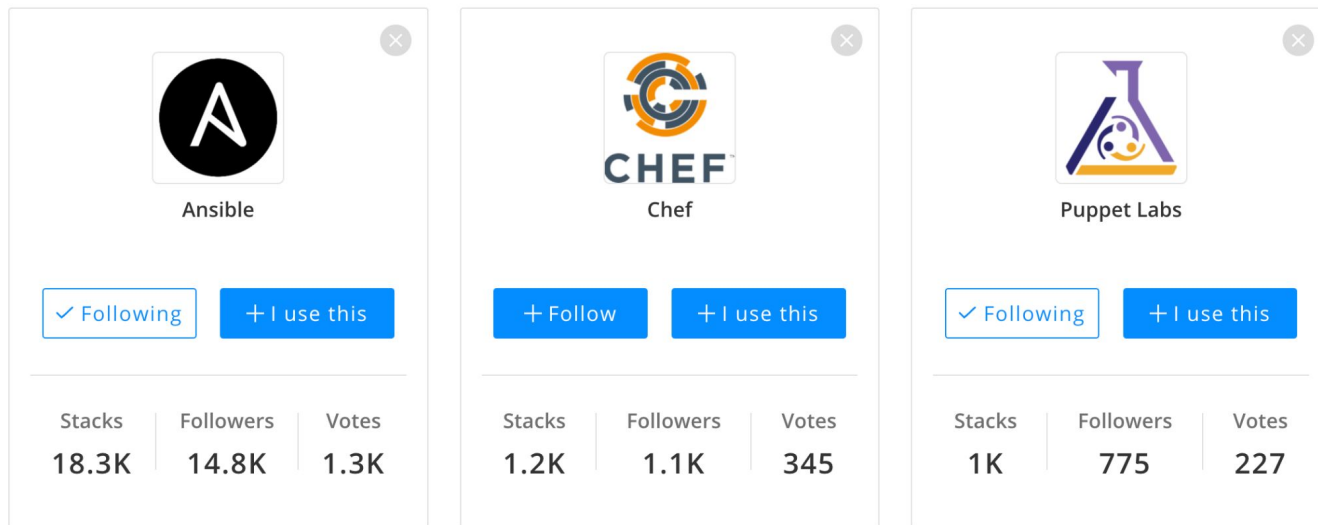
Caractéristiques	Ansible	Terraform
Type d'outil	Automatisation et gestion de configuration	Orchestration - Infrastructure as Code (IaC)
Langage de Description	YAML (Playbooks)	HashiCorp Configuration Language (HCL)
Approche	Agentless (pas d'agent requis sur les cibles)	Déclaratif et orienté état
Utilisation principale	Automatiser des tâches, configurer des systèmes	Déployer et gérer des infrastructures
Cible principale	Configuration système et applications	Infrastructures et ressources cloud
Dépendances	Dépend de l'état actuel des cibles	Crée et modifie les ressources indépendamment
Gestion de l'état	Prévoit l'état désiré et applique les changements	Gère l'état réel et le modifie pour correspondre à l'état désiré
Exemples de ressources	Packages, fichiers, services, utilisateurs	Machines virtuelles, réseaux, groupes de sécurité
Exécution en parallèle	Peut exécuter des tâches en parallèle sur plusieurs cibles	Crée des ressources en parallèle, mais les dépendances peuvent limiter certaines actions
Support des fournisseurs cloud	Peut gérer différentes plates-formes avec des modules. Extensible par des modules personnalisés	Fournit un large éventail de fournisseurs cloud. Egalement extensible par des modules personnalisés

Ansible VS Puppet































Caractéristiques	Ansible	Puppet
Type d'outil	Automatisation et gestion de configuration	Gestion de configuration et automatisation
Langage de Description	YAML (Playbooks)	Puppet DSL (Domain-Specific Language)
Approche	Agentless (pas d'agent requis sur les cibles)	Agent-based (requiert un agent sur les cibles)
Utilisation principale	Automatiser des tâches, configurer des systèmes	Configurer et gérer les états des systèmes
Cible principale	Configuration système et applications	Configuration et gestion des systèmes
Dépendances	Dépend de l'état actuel des cibles	Gère l'état cible pour correspondre à la définition
Gestion de l'état	Prévoit l'état désiré et applique les changements	Gère l'état réel et le modifie pour correspondre à l'état désiré
Exemples de ressources	Packages, fichiers, services, utilisateurs	Packages, fichiers, services, utilisateurs
Exécution en parallèle	Peut exécuter des tâches en parallèle sur plusieurs cibles	Peut exécuter des tâches en parallèle, mais avec des contraintes
Maintenance continue	Peut être utilisé pour gérer les configurations en continu	Peut être utilisé pour effectuer des modifications continues sur les systèmes
Complexité	Peut être utilisé pour des tâches simples ou complexes	Peut être complexe pour des infrastructures complexes

Positionnement de Ansible



→ <https://stackshare.io/stackups/ansible-vs-chef-vs-puppet>

Positionnement de Ansible

 linux Public	Contributeurs : +14.473	 Watch 8.2k	 Fork 49.8k	 Star 157k
 kubernetes Public	Contributeurs : +3.497	 Watch 3.2k	 Fork 37.7k	 Star 102k
 ansible Public	Contributeurs : +5.505	 Watch 1.9k	 Fork 23.7k	 Star 58.5k
 terraform Public	Contributeurs : +1.758	 Watch 1.2k	 Fork 9k	 Star 38.8k
 salt Public	Contributeurs : +2.424	 Watch 538	 Fork 5.5k	 Star 13.5k
 chef Public	Contributeurs : +657	 Watch 371	 Fork 2.6k	 Star 7.3k
 puppet Public	Contributeurs : +566	 Watch 468	 Fork 2.3k	 Star 7.1k

Introduction à YAML

- YAML (YAML Ain't Markup Language) est un format de sérialisation de données lisible par l'homme.
- Souvent utilisé pour les fichiers de configuration et l'échange de données entre les langages, les applications et les plateformes.
- YAML utilise l'**indentation** et des caractères spéciaux pour structurer les données, ce qui facilite la lecture et l'écriture.
- Utilisé par plusieurs outils DevOps Populaires :
 - ◆ Kubernetes
 - ◆ **Ansible**
 - ◆ Docker
 - ◆ Cloud Formation
 - ◆ Plusieurs Outils CI
 - ◆ ...

Syntaxe YAML

→ Caractéristiques principales :

- ◆ Lisible par l'homme
- ◆ Structure basée sur l'indentation
- ◆ Couramment utilisé pour les fichiers de configuration

→ Structure YAML

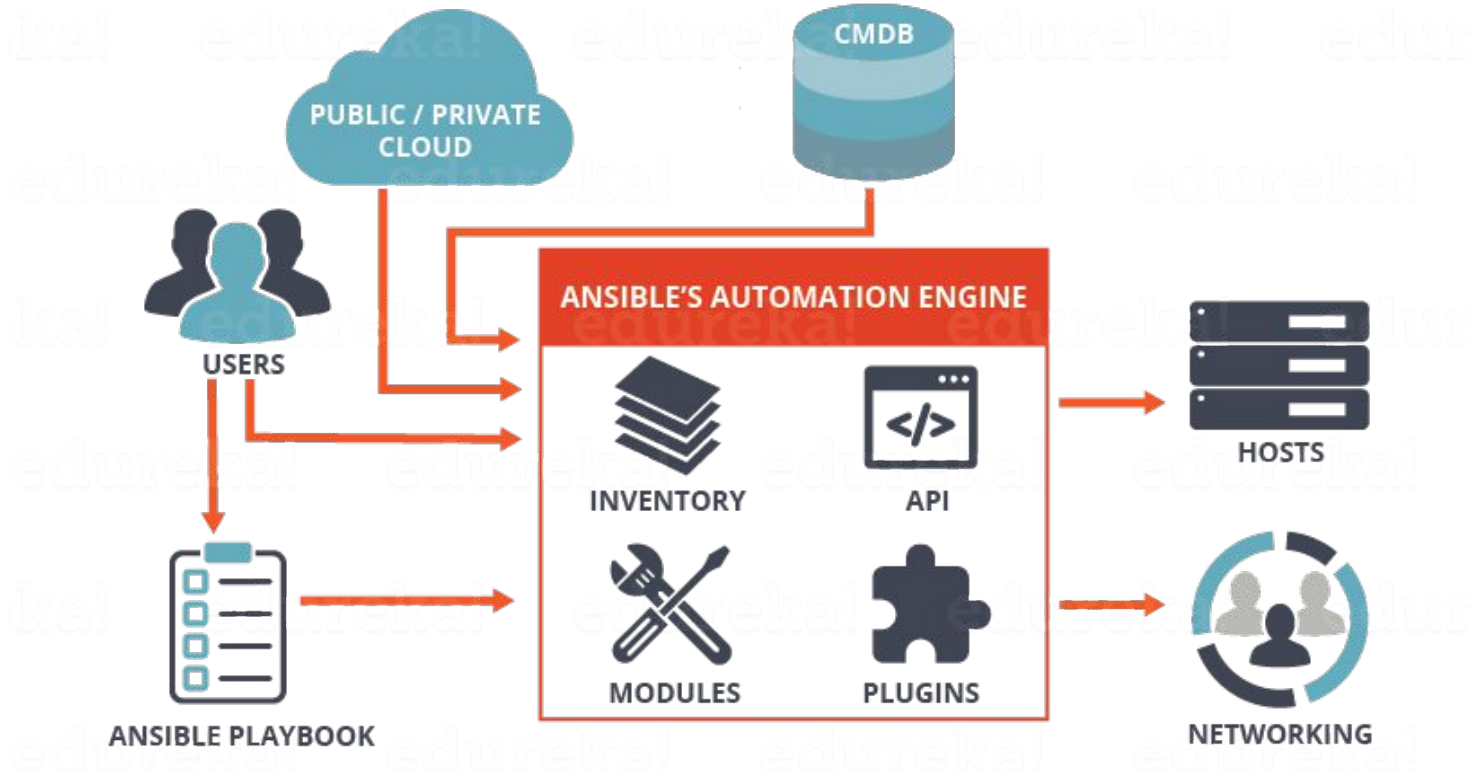
- ◆ YAML utilise l'indentation pour définir les hiérarchies et l'imbrication des données.
- ◆ Les données en YAML sont représentées par des paires clé-valeur.
- ◆ Les listes et les dictionnaires sont exprimés avec la syntaxe appropriée.
- ◆ Les commentaires commencent par le symbole "#"

```
personne:  
  nom: Jean Duppon  
  age: 30  
  interets:  
    - lecture  
    - randonnée  
    - cuisine
```

Ansible - Prise en main

- Architecture Ansible
- Fichiers de configurations
- Inventory
- Notions : Playbook, Play, Module, Task, Role...
- Syntaxe Yaml.
- Ansible Ad-hoc
- Commandes de base d'Ansible
- Modules Ansible les plus utilisés
- Configuration des noeuds : clés ssh

Architecture Ansible



Ansible : Fichiers de configurations

- Ansible dispose de plusieurs sources pour configurer son comportement, notamment :
 - ◆ Un fichier ini nommé **ansible.cfg**
 - ◆ des variables d'environnement
 - ◆ des options de ligne de commande
 - ◆ des mots-clés de playbook
 - ◆ et des variables.
- L'utilitaire **ansible-config** permet aux utilisateurs de voir tous les paramètres de configuration disponibles,

Ansible : Fichiers de config ini - ansible.cfg

- Si Ansible est installé à partir d'un gestion de paquets, le dernier fichier de config doit être à cet emplacement **/etc/ansible/ansible.cfg**
- Des modifications peuvent être apportées et utilisées dans un fichier de configuration qui sera recherché dans l'ordre suivant :
 - ◆ **ANSIBLE_CONFIG** (variable d'environnement si définie)
 - ◆ **ansible.cfg** (dans le répertoire courant)
 - ◆ **~/.ansible.cfg** (dans le répertoire personnel)
 - ◆ **/etc/ansible/ansible.cfg**

Ansible : Fichiers de config ini - Ex : ansible.cfg

```
[defaults]
# Spécifie le fichier d'inventaire des hôtes
inventory = /path/to/your/inventory

# Spécifie user distant qu'Ansible doit utiliser lors de
la connexion aux hôtes gérés
remote_user = admin

# Indique l'emplacement de clé privée SSH.
private_key_file = /path/to/your/ssh/key.pem

# Nombre de threads parallèles à utiliser (Par défaut 5)
forks = 5

# Contrôler si Ansible doit utiliser l'élévation de
privileges (become) lors de l'exécution des tâches.
become = True
```

```
# Spécifie la méthode d'escalade des privilèges.
become_method = sudo

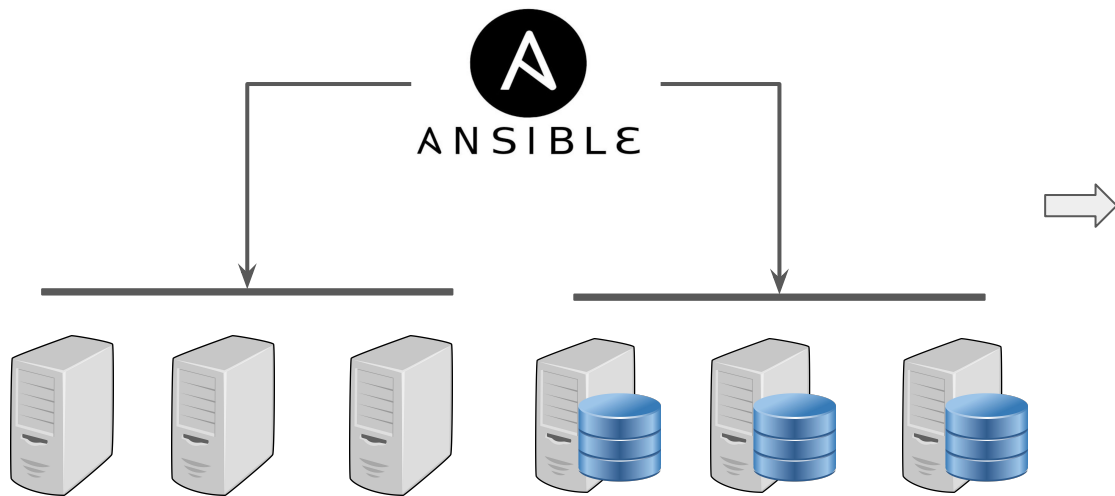
# Contrôle si Ansible doit demander un mot de passe pour
l'élévation de privilège
ask_become_pass = True

# Détermine si Ansible doit enregistrer et afficher des
informations sur les hôtes gérés.
gather_facts = True

# Spécifiez l'emplacement des rôles dans le répertoire
de votre projet Ansible.
roles_path = /path/to/your/ansible/roles
```


Inventaire

- Ansible travaille simultanément sur plusieurs systèmes de votre infrastructure. Pour ce faire, il sélectionne des portions de systèmes répertoriées dans le fichier d'**inventaire** d'Ansible
- Par défaut : **/etc/ansible/hosts**.



[Web]

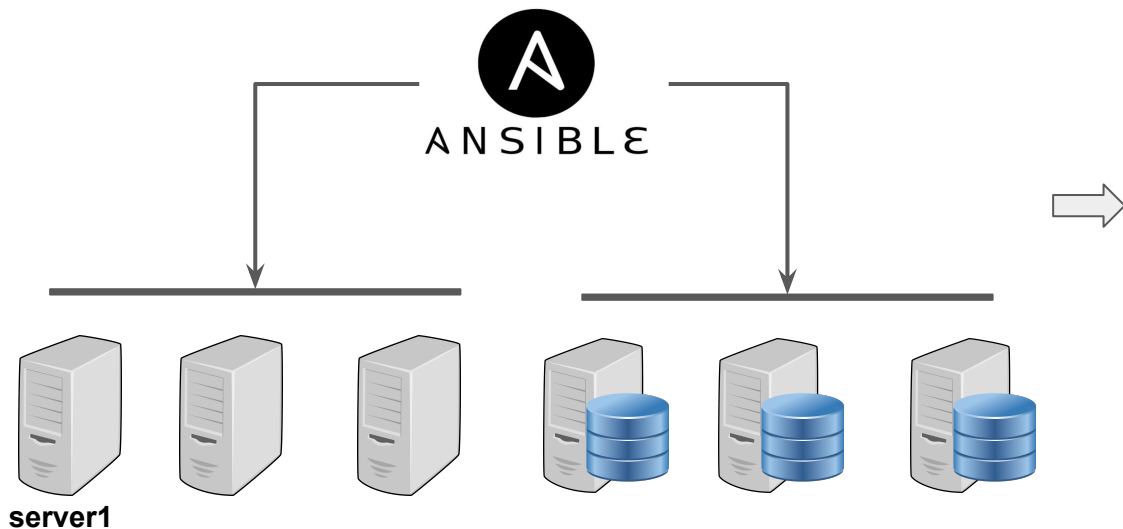
```
server1.wizetraining.local  
server2.wizetraining.local  
server3.wizetraining.local
```

[DB]

```
db1.wizetraining.local  
db2.wizetraining.local  
db3.wizetraining.local
```

Inventaire : Hôte

- Un **hôte** est simplement une machine distante gérée par Ansible.
 - ◆ **Ex** : server1.wizetraining.local
- Des variables individuelles peuvent leur être attribuées, et ils peuvent également être organisés en **groupes**.



[Web]

server1.wizetraining.local
server2.wizetraining.local
server3.wizetraining.local

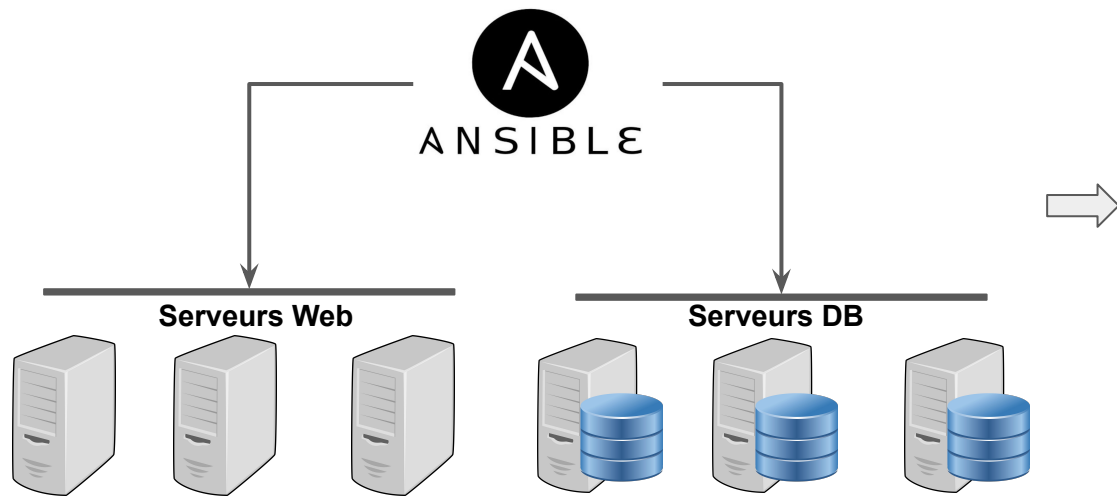
[DB]

db1.wizetraining.local
db2.wizetraining.local
db3.wizetraining.local

Inventaire : Groupe

→ Un **groupe** est constitué de plusieurs hôtes assignés à un pool qui peuvent être ciblés ensemble de manière pratique et auxquels sont attribuées des variables qu'ils partagent en commun.

◆ Ex : Web, DB



[Web]

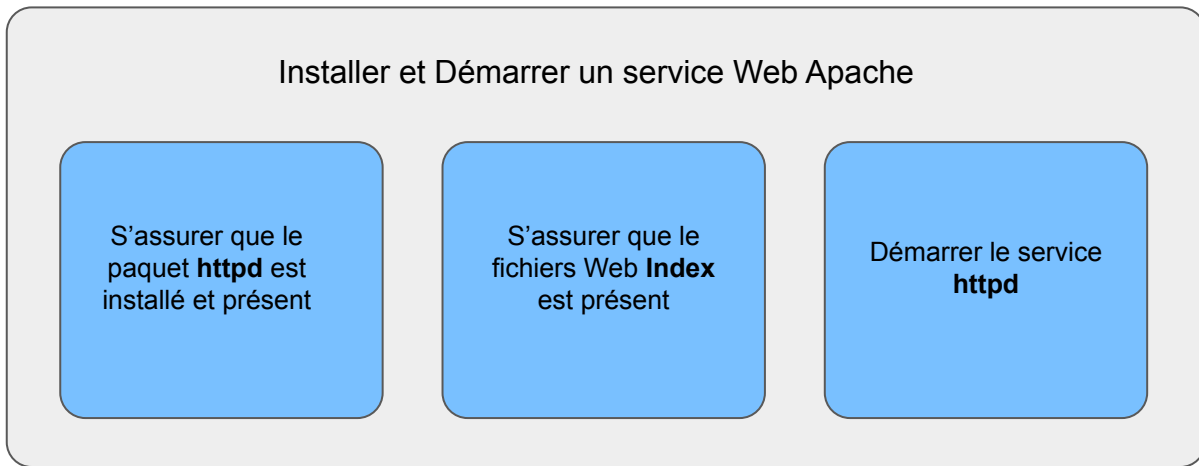
```
server1.wizetraining.local  
server2.wizetraining.local  
server3.wizetraining.local
```

[DB]

```
db1.wizetraining.local  
db2.wizetraining.local  
db3.wizetraining.local
```

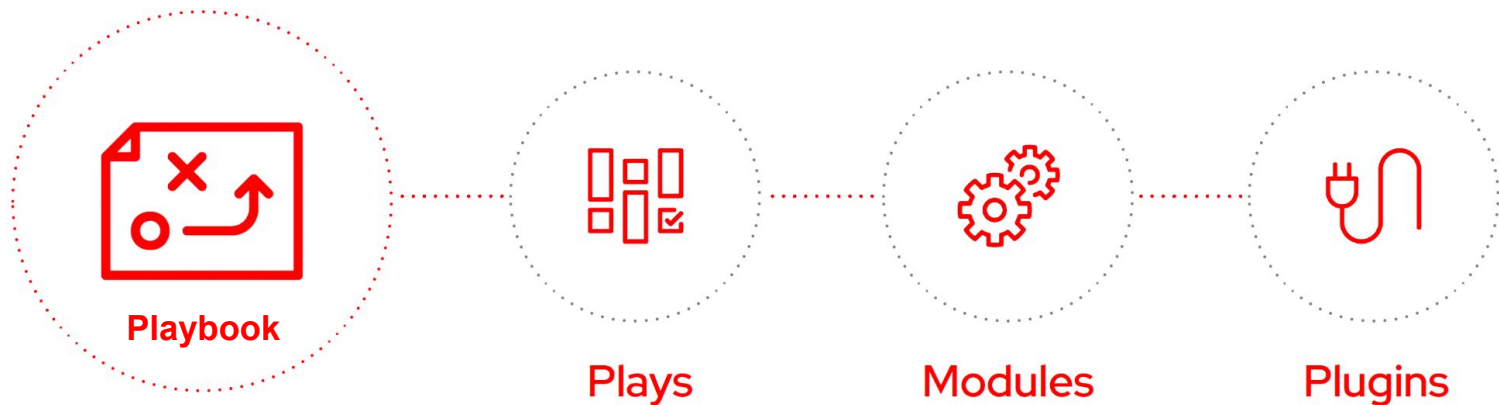
Ansible Playbook

- Les **Playbooks** sont le langage par lequel Ansible orchestre, configure, administre ou déploie des systèmes.
- Les **Playbooks** contiennent des **Plays**.



Ansible Playbook

→ De quoi se compose un playbook Ansible ?



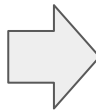
Ansible Playbook

Installer et Démarrer un
service Web Apache

S'assurer que le paquet **httpd** est installé et
présent

S'assurer que le fichiers Web **Index** est
présent

Démarrer le service **httpd**



```
---  
- name: Installer et démarrer Apache  
  hosts: web  
  become: yes  
  tasks:  
    - name: Le paquet httpd est présent  
      yum:  
        name: httpd  
        state: latest  
    - name: Le fichier index.html est présent  
      template:  
        src: files/index.html  
        dest: /var/www/html/  
    - name: httpd est démarré  
      service:  
        name: httpd  
        state: started
```

Ansible : Plays

- **Play** : Spécification de haut niveau pour un groupe de tâches, qui indique à la pièce les hôtes sur lesquels elle s'exécutera et contrôle le comportement, comme la collecte de données ou le niveau de privilège.
- Il peut y avoir plusieurs **Plays** dans un **Playbook** qui s'exécutent sur différents hôtes.

```
---  
- name: Installer et démarrer Apache  
  hosts: web  
  gather_facts: no  
  become: yes
```

Ansible : Modules & Tâche

- **Module** : Composants paramétrés avec une logique interne, représentant une étape unique à réaliser.
- Les **Tâches** combinent une **action** avec un **nom** et éventuellement d'autres mots clés (comme des directives de boucle).
- Les tâches font appel à des "modules" .
- Language : Généralement Python, ou Powershell pour les installations Windows. Mais il peut s'agir de n'importe quel langage.
- Liste de tous les modules :

https://docs.ansible.com/ansible/latest/collections/index_module.html

```
- name: Le paquet httpd est présent
yum:
  name: httpd
  state: latest
```


Ansible : Plugin

- Les plugins sont des éléments de code qui augmentent les fonctionnalités de base d'Ansible.
- Ansible utilise une architecture de plugins pour permettre un ensemble de fonctionnalités riches, flexibles et extensibles.

```
---  
- name: Installer et démarrer Apache  
  hosts: web  
  become: yes
```

Ansible : Roles

- Les **Role** sont des structures réutilisables : le principe est de regrouper les tâches et les variables de votre automatisation dans une structure réutilisable appelée Role.
- Rédigez les Role une seule fois et partagez-les avec d'autres personnes confrontées à des défis similaires.

```
---  
- name: Installer les composants serveurs K8s  
  hosts: master-nodes  
  roles:  
    - common  
    - masters
```

Ansible : Collections

- Diffusion simplifiée et cohérente du contenu - <https://galaxy.ansible.com/>
- Les collections sont une structure de données contenant des contenus d'automatisation :
Modules, Playbooks, Rôles, Plugins, Docs, Tests

```
nginx_core
├── MANIFEST.json
├── playbooks
│   └── deploy-nginx.yml
│   └── ...
├── plugins
├── README.md
├── roles
│   ├── nginx
│   │   ├── defaults
│   │   ├── files
│   │   │   └── ...
│   │   ├── tasks
│   │   └── templates
│   │       └── ...
│   ├── nginx_app_protect
│   └── nginx_config
```

deploy-nginx.yml

```
---
- name: Install NGINX Plus
  hosts: all
  tasks:
    - name: Install NGINX
      include_role:
        name: nginxinc.nginx
      vars:
        nginx_type: plus

    - name: Install NGINX App Protect
      include_role:
        name: nginxinc.nginx_app_protect
      vars:
        nginx_app_protect_setup_license: false
        nginx_app_protect_remove_license: false
        nginx_app_protect_install_signatures: false
```



Ansible : installation

→ Installation de la Dernière version via Yum

```
# install the epel-release RPM if needed on CentOS, RHEL, or Scientific Linux
$ sudo dnf install ansible
```

→ Installation et mise à jour d'Ansible avec pip

```
# Pour vérifier si pip est déjà installé pour votre Python préféré
$ python3 -m pip -V

# Utilisez pip dans l'environnement Python que vous avez choisi pour installer le paquetage
Ansible de votre choix pour l'utilisateur actuel
$ python3 -m pip install --user ansible

# Vous pouvez également installer une version spécifique d'ansible-core dans cet environnement
Python :
$ python3 -m pip install --user ansible-core==2.12.3
```

Ansible : Command Ad Hoc

→ Effectuer un Ping des hôtes et des groupes définis

```
# Ping d'un hôte spécifique
$ ansible -m ping server1.wizetraining.local

# Ping du groupe d'hôtes Web Server1, Server2 & Server3
$ ansible -m ping Web

# Ping de tous les hôtes contenus dans l'inventaire
$ ansible -m ping all
```

Ansible : Command Ad Hoc

- Quelques exemples de commandes Ad Hoc exécutés sur les cibles
- L'option “-a” est utilisée pour passer des arguments au module.
 - ◆ **Module** : Shell;
 - ◆ **Argument** : ls -al

```
# Lister le contenu du répertoire courant de l'utilisateur sur la cible
$ ansible -m shell -a 'ls -al' server1.wizetraining.local

# Exécution de la commande "whoami" sur les hôtes Web
$ ansible -m shell -a 'whoami' web

# Exécution de la commande "ifconfig" sur tous les hôtes de l'inventaire
$ ansible -m shell -a 'ifconfig' all
```

Ansible : Tâches en Ad Hoc

→ Ansible peut exécuter des tâches uniques sur des ensembles d'hôtes en mode ad hoc.

- Module “file” pour la création de fichiers/répertoires

```
$ ansible webservers -m file -a "path=/var/www/html/assets state=directory"
```

- Module “apt” pour l’installation des paquets

```
$ ansible webservers -m apt -a "name=nginx state=present"
```

- Module “service” pour la gestion des services sur Linux

```
$ ansible webservers -m service -a "name=nginx enabled=yes state=started"
```

Ansible : Clé SSH

- Pour configurer l'agent SSH afin d'éviter de retaper les mots de passe, vous pouvez ajouter la clé privée

```
# Création d'une nouvelle paire de clés SSH
$ ssh-keygen
```

- Exporter la clé vers tous les hôtes

```
# Création d'un nouvel utilisateur "admin" sur les hôtes distants servant d'administration
$ ansible all -u root -m user -a "name=admin state=present"

# Exportation de notre clé Publique vers les hôtes administrés
$ ansible all -u root -m authorized_key -a "user=admin state=present key='{{ lookup('file',
'/home/vagrant/.ssh/id_rsa.pub') }}'"

# Modification du fichier "/etc/sudoers" afin que l'escalade de privilège ne requiert pas de mot de passe
$ ansible all -u root -m lineinfile -a "path=/etc/sudoers insertafter='^root' line='admin    ALL=(ALL)
NOPASSWD:ALL' "
```


Ansible : Modules

→ Lister les modules et la documentation avec la commande **ansible-doc**

- Lister les modules

```
$ ansible-doc -l
```

- Consulter la documentation du module package

```
$ ansible-doc package
```

→ Liste de tous les modules : https://docs.ansible.com/ansible/latest/collections/index_module.html

Les modules Ansible les plus courant

- **ping** : Vérifie la connectivité aux hôtes cibles.
- **command** : Exécute une commande sur les hôtes distants.
- **shell** : Exécute une commande shell sur les hôtes distants avec plus de contrôle que le module `command`.
- **copy** : Copie des fichiers vers les hôtes distants.
- **template** : Copie un modèle Jinja2 sur les hôtes distants, en permettant la substitution de variables.
- **apt/yum** : Gère les packages (installation, suppression, mise à jour) sur les systèmes basés sur Debian ou Red Hat.
- **service** : Gère les services système (démarrage, arrêt, redémarrage) sur les hôtes.
- **user** : Gère les utilisateurs et les groupes d'utilisateurs sur les hôtes.
- **file** : Gère les fichiers et les répertoires sur les hôtes distants.
- **git** : Clone ou met à jour des dépôts Git sur les hôtes distants.

```
$ ansible-doc <Nom_Module> ==> Voir des exemples d'utilisation à la fin
```

Les modules Ansible les plus courant

- **lineinfile** : Modifie les lignes dans un fichier en fonction d'une expression régulière.
- **yum/apt_repository** : Gère les dépôts de packages sur les systèmes Red Hat ou Debian.
- **debug** : Affiche des messages de débogage pendant l'exécution des playbooks.
- **cron** : Gère les tâches cron sur les hôtes distants.
- **apt_key/yum_key** : Gère les clés GPG pour les dépôts APT ou YUM.
- **group** : Gère les groupes d'utilisateurs sur les hôtes.
- **get_url** : Télécharge des fichiers depuis Internet vers les hôtes distants.
- **wait_for** : Attend que certaines conditions soient remplies (par exemple, un port réseau est accessible).
- **docker_container** : Gère les conteneurs Docker sur les hôtes.

LAB - TP

- Fichiers de configurations
- Inventory
- Playbook, Play, Module, Task, Role...
- Syntaxe Yaml.
- Ansible Ad-hoc
- Commandes de base d'Ansible
- Configuration des noeuds : clés ssh

Ansible - Playbooks

- Module
- Gestion des Variables
- Notifications et Handlers
- Exécution step by step
- Saut de tasks
- Gestion des erreurs, dry-run
- Conditions, boucles et blocks

Ansible : Modules

- **Module** : les bouts de code copiés dans le système cible pour être exécutés afin de satisfaire à la déclaration de tâche
- ◆ Le code n'a pas besoin d'exister sur l'hôte distant -- Ansible le copie.
 - ◆ De nombreux modules sont fournis avec Ansible -- "piles incluses"
 - ◆ Des modules personnalisés peuvent être développés facilement
 - ◆ Il existe des modules de commande/shell pour les commandes simples
 - ◆ Le module **script** existe pour utiliser le code existant

```
- name: Le paquet httpd est présent
yum:
  name: httpd
  state: latest
```

Hôtes et groupe : variables Ansible

- Utilisation d'un port SSH différent sur un Hôte dans le fichier Inventaire

```
[Web]  
Server1.wizetraining.local:2222
```

- Utilisation des Alias

```
[Web]  
Server1.wizetraining.local ansible_port=22 ansible_host=192.168.35.102
```

- Utilisation du globbing pour englober plusieurs hôtes

```
[Web]  
Server[1:3].wizetraining.local
```

Hôtes et groupe : variables Ansible

- Attribuer des variables aux hôtes qui seront utilisées plus tard dans les playbooks[webserver]

```
[Web]
Server1.wizetraining.local http_port=80 https_port=443
Server2.wizetraining.local http_port=8080 https_port=8443
```

- Les variables peuvent également être appliquées à l'ensemble d'un groupe en une seule fois

```
[Web]
Server[1:3].wizetraining.local

[Web:vars]
ntp_server=tr.pool.ntp.org
proxy=proxy.wizetraining.local
```


Hôtes et groupe : Groupe de groupes

→ Pour créer des groupes de groupes, on utilise le suffixe : **children**.

```
[Web]
server1.wizetraining.local
server2.wizetraining.local
server3.wizetraining.local

[DB]
db1.wizetraining.local
db2.wizetraining.local
db3.wizetraining.local

[app1:children]
Web
DB
```

Playbook : variables Ansible

→ Comment déclarer une variable localement dans un Playbook

```
---  
- name: Installer et démarrer Apache  
  hosts: web  
  become: yes
```

```
  vars:  
    assets_dir: /var/www/html/static
```

```
  tasks:  
    - name: Création du répertoire  
      file:  
        path: {{ assets_dir }}  
        state: directory
```

Déclaration de variable dans le Play

Utilisation de la variable

Playbook : variables Ansible

- Il y a en fait 21 endroits où l'on peut placer des variables dans Ansible
 - ◆ https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html#understanding-variable-precedence
- Si une variable est placée à plusieurs endroits simultanément; voici l'ordre de priorité, de la plus petite à la plus grande (les dernières variables énumérées ont la priorité sur toutes les autres) :

```
$ ansible-playbook playbook.yml --extra-vars assets_dir=/web/files
```

- Valeurs sur la CLI (Ex: **-u my_user**, n'est pas une variable)
- role defaults (defined in role/defaults/main.yml) 1
- inventory file or script group vars 2
- inventory group_vars/all 3
- playbook group_vars/all 3
- inventory group_vars/* 3
- playbook group_vars/* 3
- inventory file or script host vars 2
- inventory host_vars/* 3
- playbook host_vars/* 3
- host facts / cached set_facts 4
- play vars
- play vars_prompt
- play vars_files
- role vars (defined in role/vars/main.yml)
- block vars (only for tasks in block)
- task vars (only for the task)
- include_vars
- set_facts / registered vars
- role (and include_role) params
- include params
- **extra vars en CLI** (for example, **-e "user=my_user"**) (toujours prioritaire)

Playbook : Notifications et Handlers

- Parfois, vous souhaitez qu'une tâche ne s'exécute que lorsqu'une modification est apportée à une machine.
- Ex : Vouloir redémarrer un service si une tâche met à jour la configuration de ce service, mais pas si la configuration reste inchangée.

```
tasks:
  - name: Mettre à jour la conf nginx
    copy:
      src=default.conf
      dest=/etc/nginx/nginx.conf
    notify:
      - Config a jour

handlers:
  - name: Config a jour
    service:
      name=nginx
      state=restarted
```

Playbook : Notifications et Handlers

- Les **Handlers** doivent être nommés pour que les tâches puissent les notifier à l'aide du mot-clé **notify**.
- Les **Handlers** peuvent également utiliser le mot-clé **listen**.
- En utilisant ce mot-clé, les handlers peuvent écouter des sujets qui peuvent regrouper plusieurs gestionnaires comme suit

```
tasks:
  - name: Tout redemarrer
    command: echo "cette tâche redémarre les services web"
    notify: "restart web services"

handlers:
  - name: Redemarrer memcached
    service:
      name: memcached
      state: restarted
      listen: "restart web services"

  - name: Redemarrer apache
    service:
      name: apache
      state: restarted
      listen: "restart web services"
```

Playbook : Exécution étape par étape

```
$ ansible-playbook ./install-nginx.yml -l web -step
PLAY [webservers] *****
TASK [setup] *****
ok: [web]
TASK [Install nginx] *****
ok: [web]
TASK [Start nginx] *****
ok: [web]
PLAY RECAP *****
web : ok=3 changed=0 unreachable=0 failed=0
```

Playbook : dry-run le mode de verification

- Lorsque ansible-playbook est exécuté avec `--check`, il n'effectue aucune modification sur les systèmes distants.
- Tout module instrumenté pour prendre en charge le "mode de vérification check" (Vrai pour la plupart des modules principaux)
- Le mode check signalera les changements qu'il aurait apportés sans les effectuer

```
$ ansible-playbook install-nginx.yml --check
```

Playbook : dry-run le mode de verification

- Il peut arriver que vous souhaitiez modifier le comportement du mode de contrôle de certaines tâches.
- Cela se fait par l'intermédiaire de l'option `check_mode`, qui peut être ajoutée aux tâches.
 - ◆ `check_mode : yes` ⇒ Force une tâche à s'exécuter en mode check, même lorsque le playbook est appelé sans `--check`. C'est ce qu'on appelle.
 - ◆ `check_mode : no` ⇒ Force une tâche à s'exécuter en mode normal et à apporter des modifications au système, même lorsque le playbook est appelé avec `--check`.

```
tasks:
  - name: cette tâche apportera des
    modifications au système même en mode de
    contrôle
    command: /something/to/run
  --even-in-check-mode
  check_mode: no

  - name: cette tâche sera toujours exécutée en
    mode de contrôle et ne modifiera pas le système
    lineinfile:
      line: "important config"
      dest: /path/to/myconfig.conf
      state: present
  check_mode: yes
```


Playbook : Gestion des erreurs

- Lorsqu'Ansible reçoit un code de retour **non nul** d'une commande ou un **échec d'un module**, il arrête par défaut l'exécution sur cet hôte et continue sur les autres hôtes.
- Dans certaines circonstances, vous pouvez souhaiter un comportement différent.
 - ◆ Vous pouvez utiliser `ignore_errors` pour continuer malgré l'échec.
 - ◆ Vous pouvez ignorer un échec de tâche dû au fait que l'instance hôte est 'INJOIGNABLE' avec le mot-clé `ignore_unreachable`.

```
- name: Ne considérez pas cela comme un échec
  ansible.builtin.command: /bin/false
  ignore_errors: true

- name: Cette opération s'exécute, échoue et l'échec est ignoré.
  ansible.builtin.command: /bin/true
  ignore_unreachable: true
```

Playbook : Gestion des erreurs

- Lorsqu'Ansible reçoit un code de retour **non nul** d'une commande ou un **échec d'un module**, il arrête par défaut l'exécution sur cet hôte et continue sur les autres hôtes.
- Dans certaines circonstances, vous pouvez souhaiter un comportement différent.
 - ◆ Vous pouvez utiliser `ignore_errors` pour continuer malgré l'échec.
 - ◆ Vous pouvez ignorer un échec de tâche dû au fait que l'instance hôte est 'INJOIGNABLE' avec le mot-clé `ignore_unreachable`.

```
- name: Ne considérez pas cela comme un échec
  ansible.builtin.command: /bin/false
  ignore_errors: true

- name: Cette opération s'exécute, échoue et l'échec est ignoré.
  ansible.builtin.command: /bin/true
  ignore_unreachable: true
```

Playbook : Gestion des erreurs

- Ansible permet de définir ce que signifie **"échec"** pour chaque tâche à l'aide de la conditionnelle `failed_when`.
- ◆ Vous pouvez informer Ansible de l'échec d'une tâche en recherchant un mot ou une phrase dans la sortie d'une commande
 - ◆ Ou sur la base du code de retour

```
- name: Échec de la tâche lorsque la sortie d'erreur de la commande affiche FAILED
  ansible.builtin.command: /usr/bin/example-command -x -y -z
  register: command_result
  failed_when: "'FAILED' in command_result.stderr"

- name: Échec de la tâche lorsque les deux fichiers sont identiques
  ansible.builtin.raw: diff foo/file1 bar/file2
  register: diff_cmd
  failed_when: diff_cmd.rc == 0 or diff_cmd.rc >= 2
```

Playbook : Conditions

- Dans un Playbook, vous pouvez vouloir exécuter différentes tâches ou avoir différents objectifs, en fonction de :
- ◆ La valeur d'un **fact** (données sur le système distant. Ex: type de l'OS)
 - ◆ Une variable
 - ◆ Ou le résultat d'une tâche précédente.

Playbook : Conditions simple - When

- Lorsque vous exécutez la tâche ou le playbook, Ansible évalue la condition **when** pour tous les hôtes.
- Conditions basées sur **ansible_facts**

```
tasks:
  - name: Configurer SELinux pour démarrer mysql
    ansible.posix.seboolean:
      name: mysql_connect_any
      state: true
      persistent: true
      when: ansible_selinux.status == "enabled"
```

Playbook : Conditions simple - When

- Conditions basées sur **ansible_facts**
- List des facts ansible qui sont utilisés plus fréquemment :
 - ◆ `Ansible_facts['distribution']`
 - ◆ `Ansible_facts['os_family']`
 - ◆ `ansible_facts['distribution_major_version']`

```
tasks:
  - name: s assurer que Apache est installé sur CentOS
    yum:
      name: httpd
      state: present
    when: Ansible_facts['distribution'] == "CentOS" and ansible_facts['distribution_major_version'] == "7"

  - name: s assurer que Apache est installé sur Ubuntu
    apt:
      name: apache2
      state: present
    when: Ansible_facts['distribution'] == "Ubuntu"
```

Playbook : Boucles

→ Ansible propose les mots-clés `loop`, `with_<lookup>` et `until` pour exécuter une tâche plusieurs fois.

- ◆ Les mots-clés `with_<lookup>` s'appuient sur les plugins Lookup. Lister les lookup

```
$ ansible-doc -t lookup -l
```

- ◆ Le mot-clé `loop` est équivalent à `with_list` (préférable d'utiliser `loop` pour les boucle simple)

```
- name: Ajouter certains utilisateurs
  user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  loop:
    - testuser1
    - testuser2
```

Playbook : Boucles avec with_<lookup>

- Utilisation d'une boucle avec "with_<lookup>"
- Ces mots clés sont utilisés pour transformer des structures de données en liste simple comme le fait "loop"

→ **Exemple :**

- ◆ with_list
- ◆ with_items
- ◆ with_indexed_items
- ◆ with_dict
- ◆ with_sequence
- ◆ with_random_choice

```
- name: Ajouter des Utilisateurs
  user:
    name : {{ item.name }}
    state : present
    groups : {{ item.groups }}
  with_items:
    - { name: 'testuser1', groups: 'wheel' }
    - { name: 'testuser2', groups: 'root' }
```


Playbook : Blocks

- Les blocks créent des groupes logiques de tâches.
- Permettent également de :
 - ◆ Grouper des tasks et leur appliquer une condition commune, boucle, handler...
 - ◆ Gérer des errors en bloc

```
tasks:
  - name: Installer, configurer, et démarrer Apache
    block:
      - name: Installer httpd
        yum:
          name:
            - httpd
          state: present

      - name: appliquer le template
        template:
          src: templates/src.j2
          dest: /etc/foo.conf

      - name: démarrer le service et l'activer
        service:
          name: httpd
          state: started
          enabled: True

    when: ansible_facts['distribution'] == 'CentOS'
    become: true
    become_user: root
    ignore_errors: true
```

Ansible - Tags

- Il peut être utile de n'exécuter que des parties spécifiques d'un Playbook au lieu l'ensemble (Ex: Playbook volumineux)
 - ◆ Ajouter des **Tags** aux **tasks**, soit individuellement, soit en héritant des **Tags** d'un block, play, role, ou import
 - ◆ Sélectionnez ou ignorez les **Tags** lorsque vous exécutez votre Playbook.
- **Tags spéciaux** : **always** and **never**
 - ◆ Si vous attribuez le tag **always** à une tâche ou à une play, Ansible exécutera toujours cette tâche ou ce play, à moins que vous ne l'ignoriez spécifiquement en CLI (`--skip-tags always`).
 - ◆ Si vous assignez le tag **never** à une tâche ou à une pièce, Ansible sautera cette tâche ou ce play à moins que vous ne le demandiez spécifiquement (`--tags never`).

Ansible - Tags

- Si vous exécutez ces 2 tâches dans un playbook avec `--tags configuration`, Ansible exécutera la tâches étiquetées `configuration` et ignorera la tâche qui n'a pas cette étiquette.

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

```
tasks:
- name: Install the servers
  ansible.builtin.yum:
    name:
      - httpd
      - memcached
    state: present
    tags:
      - packages
      - webserverns

- name: Configure the service
  ansible.builtin.template:
    src: templates/src.j2
    dest: /etc/foo.conf
    tags:
      - configuration
```

Playbook : Saut de tasks

→ Utilisation des tags pour exclure ou sélectionner des tâches

- ◆ `--tags all` - exécute toutes les tâches, ignore les balises (comportement par défaut)
- ◆ `--tags [tag1, tag2]` - exécute uniquement les tâches avec le tag1 ou le tag2
- ◆ `--skip-tags [tag3, tag4]` - exécute toutes les tâches sauf celles avec le tag3 ou le tag4
- ◆ `--tags tagged` - exécute uniquement les tâches comportant au moins un tag
- ◆ `--tags untagged` - exécute uniquement les tâches sans tag

```
$ ansible-playbook playbook.yml --tags all
```

LAB - TP

- Modules
- Gestion des Variables
- Notifications et Handlers
- Exécution step by step
- Saut de tasks
- Gestion des erreurs, dry-run
- Conditions, boucles et blocks

Ansible - Notions avancées

- Rôles Ansible
- Includes & tags
- Ansible-galaxy
- Templating et variables
- Les prompts et les facts
- Lookups, Plugins
- Ansible Vault

Rôles Ansible

- Un rôle est une collection de tâches, de fichiers, de variables et de templates regroupés dans une structure organisée.
- Il représente une unité logique de configuration et d'automatisation dans Ansible.
- Les rôles sont conçus pour être réutilisables et modulaires, facilitant ainsi la gestion de configurations complexes.
- Ils facilitent la réutilisation et la modularité du code.
 - ◆ **Exemple** : Création d'un rôle Apache pour l'installation et la configuration d'un serveur Web sur Red Hat.

Rôles Ansible - Pourquoi ?

- **Réutilisation** : Les rôles permettent de réutiliser du code, ce qui simplifie la maintenance et la gestion des configurations.
- **Modularité** : Vous pouvez découper vos playbooks en tâches logiques, ce qui facilite la compréhension et la collaboration.
- **Partage** : Les rôles peuvent être partagés avec d'autres utilisateurs d'Ansible via Ansible Galaxy ou d'autres moyens.
- **Simplicité** : Les rôles simplifient la structure de vos playbooks en les rendant plus lisibles et mieux organisés.

Rôles Ansible - Structure

- **my_role/**: Le répertoire principal du rôle, portant généralement le nom du rôle lui-même.
- **defaults/**: Ce répertoire contient des fichiers YAML, comme `main.yml`, où vous pouvez définir les valeurs par défaut pour les variables utilisées dans le rôle.
- **files/**: Dans ce répertoire, vous pouvez placer des fichiers statiques à copier sur les hôtes cibles lors de l'exécution du rôle.
- **handlers/**: Ce répertoire contient des fichiers YAML, comme `main.yml`, où vous définissez des gestionnaires d'événements (handlers) qui sont invoqués en réponse à des événements déclenchés par des tâches dans le rôle.
- **meta/**: Ce répertoire contient des informations méta sur le rôle, telles que des dépendances de rôle ou d'autres méta-informations. Le fichier `main.yml` peut contenir des informations sur les auteurs, les licences, et les dépendances.

```
my_role/
├── defaults/
│   └── main.yml
├── files/
│   └── file.txt
├── handlers/
│   └── main.yml
├── meta/
│   └── main.yml
├── tasks/
│   └── main.yml
├── templates/
│   └── template.j2
├── tests/
│   ├── inventory
│   └── test.yml
├── vars/
│   └── main.yml
└── README.md
```

Rôles Ansible - Structure

- **tasks/**: C'est ici que vous définissez les tâches principales du rôle dans le fichier main.yml.
- **templates/**: Dans ce répertoire, vous placez des modèles Jinja2 qui peuvent être utilisés pour générer des fichiers de configuration dynamiquement.
- **tests/**: Ce répertoire est utilisé pour les tests du rôle, et il peut contenir un fichier d'inventaire et un playbook de test.
- **vars/**: Vous pouvez définir des variables spécifiques au rôle dans ce répertoire, généralement dans le fichier main.yml.
- **README.md**: Un fichier README qui décrit le rôle, son objectif, son utilisation, et d'autres informations utiles pour les utilisateurs du rôle.

```
my_role/
├── defaults/
│   └── main.yml
├── files/
│   └── file.txt
├── handlers/
│   └── main.yml
├── meta/
│   └── main.yml
├── tasks/
│   └── main.yml
├── templates/
│   └── template.j2
├── tests/
│   ├── inventory
│   └── test.yml
├── vars/
│   └── main.yml
└── README.md
```

Rôles Ansible - Utilisation

- Pour utiliser un rôle dans un playbook, utilisez la directive `roles` avec le nom du rôle.
- Les rôles peuvent être spécifiés avec des variables personnalisées pour une configuration flexible.
- Lors de l'exécution, Ansible applique le rôle aux hôtes cibles conformément aux tâches et aux variables définies dans le rôle.

```
---  
- hosts: webserver  
  roles:  
    - role: '/chemin/ver/le/role/web'
```

```
---  
- hosts: prod  
  roles:  
    - common  
    - web  
    - hardening
```

Ansible - Includes

- L'utilisation de **include** est une manière de réutiliser du code Ansible en l'important depuis un autre fichier.
- Cela permet de décomposer des playbooks complexes en morceaux plus petits et plus faciles à gérer.
- Avantages :
 - ◆ **Réutilisation** : Les inclusions permettent de réutiliser des tâches, des rôles ou des fichiers de variables dans différents playbooks.
 - ◆ **Organisation** : Vous pouvez organiser votre code en le divisant en fichiers logiques, ce qui rend la maintenance plus facile.
 - ◆ **Clarté** : Les inclusions rendent les playbooks plus lisibles en évitant la duplication de code.
 - ◆ **Partage** : Vous pouvez partager des inclusions avec d'autres utilisateurs d'Ansible pour favoriser la collaboration.

Ansible - Includes - types d'inclusion

→ Ansible prend en charge différents types d'inclusions :

- ◆ **include_tasks** : Importez des tâches à partir d'un fichier séparé.
 - Ex: include_tasks: tasks/my_tasks.yml
- ◆ **roles (Inclusion de Rôles)** : Réutilisez des rôles Ansible complets.
 - Ex : roles: my_role
- ◆ **include_vars** : Importez des variables depuis un fichier.
 - Ex : include_vars: vars/my_vars.yml

```
- hosts: all
  tasks:
    - debug:
        msg: task1

    - name: Include task list in play
      include_tasks:
        file: tasks-apache.yaml
```

Ansible-galaxy

- Ansible Galaxy est une plateforme communautaire et un outil pour la gestion de contenu Ansible.
- Il offre un moyen pratique de partager, découvrir et réutiliser des rôles Ansible créés par la communauté.
- hub de contenu Ansible où les utilisateurs peuvent télécharger, partager et collaborer sur des rôles et des collections Ansible.

Ansible-galaxy

→ Pourquoi Utiliser Ansible Galaxy ?

- ◆ **Réutilisation** : Vous pouvez accéder à des milliers de rôles Ansible créés par la communauté pour gagner du temps.
- ◆ **Collaboration** : Partagez vos rôles avec d'autres utilisateurs et collaborez sur des projets communs.
- ◆ **Qualité** : Les rôles Ansible Galaxy sont souvent bien documentés, testés et maintenus.

→ Utilisez la commande `ansible-galaxy install` pour télécharger un rôle à partir d'Ansible Galaxy

```
$ ansible-galaxy install username.rolename
```

→ Utilisez la commande `ansible-galaxy init` pour créer un nouveau rôle

```
$ ansible-galaxy init myrole
```

Templating et variables

- Ansible utilise le moteur de modèle Jinja2 pour générer des fichiers de configuration dynamiquement en fonction des variables.
- Les modèles sont des fichiers avec des balises Jinja2, comme `{{ variable }}`, qui sont remplacées par les valeurs des variables lors de l'exécution.
-

```
# Exemple de modèle Jinja2 pour un fichier de
configuration Apache
<VirtualHost *:{{ web_server_port }}>
    DocumentRoot /var/www/myapp-{{ app_version }}/public
    ServerName myapp.local
</VirtualHost>
```

```
---
- name: Générer un fichier de configuration Apache
  hosts: localhost
  vars:
    web_server_port: 80
    app_version: "1.0"
    server_name: "myapp.local"
  tasks:
    - name: Utiliser le module template pour générer le
      fichier de configuration
      ansible.builtin.template:
        src: apache_config.j2
        dest: /etc/httpd/conf.d/myapp.conf
        notify: Reload Apache
```


Les prompts et les facts

- Ansible offre des mécanismes pour collecter des informations (facts) sur les cibles et pour interagir avec l'utilisateur via des prompts.
- Ansible peut collecter automatiquement des facts sur les cibles (hôtes) lors de l'exécution des playbooks.
- Les facts comprennent des informations sur le système, le réseau, les disques, etc.
- Vous pouvez accéder aux facts collectés via des variables prédéfinies, telles que `ansible_facts`.

```
# Collecte automatique des facts lors de
l'exécution d'un playbook
---
- name: Collecte des Facts
  hosts: localhost
  tasks:
    - name: Afficher les Facts
      debug:
        var: ansible_facts
```

Les prompts et les facts

- utilisation des facts dans les playbooks pour prendre des décisions ou personnaliser les tâches en fonction des informations collectées.

```
# Utilisation des facts pour décider quelle tâche exécuter
---
- name: Exemple d'utilisation de Facts
  hosts: localhost
  tasks:
    - name: Vérifier le système d'exploitation
      debug:
        msg: "Le système d'exploitation est {{ ansible_facts['ansible_os_family'] }}."
      when: ansible_facts['ansible_os_family'] == "Debian"
```

Ansible : Lookups, Plugins

- **Play** : Spécification de haut niveau pour un groupe de tâches, qui indique à la pièce les hôtes sur lesquels elle s'exécutera et contrôle le comportement, comme la collecte de données ou le niveau de privilège.
- Il peut y avoir plusieurs **Plays** dans un **Playbook** qui s'exécutent sur différents hôtes.

```
---  
- name: Installer et démarrer Apache  
  hosts: web  
  become: yes
```

Ansible : Les Filtres

- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_filters.html
- Filtre Ansible:
 - ◆ Un outil de transformation de données utilisé dans les templates Jinja2.
 - ◆ Permet de modifier, formater ou manipuler des variables.
 - ◆ Utilisé avec le symbole **pipe** “ | ”
- Exemples de Filtres Courants:
 - ◆ **default**: Définit une valeur par défaut pour une variable si elle n'est pas définie ou est vide.
 - ◆ **regex_replace** : Remplace ou supprime des chaînes de caractères en utilisant des expressions régulières. (Exo 12)
 - ◆ **map**: Transforme une liste en appliquant un filtre à chaque élément. Ex: **map('capitalize')**
 - ◆ **password_hash**: pour hasher le mot de passe (Exo 13)
 - Ex: `{{ 'secretpassword' | password_hash('sha512', 'mysecretsalt') }}`

Ansible : Les Filtres

- Cette expression prend la liste `users`, applique le filtre `map('capitalize')` et ensuite le filtre avec `list`.
- Après que `map` ait appliqué `capitalize` à chaque élément, le filtre `list` est utilisé pour convertir le résultat en une liste : `["Alice", "Bob", "Charlie"]`

```
vars:
  users: ['alice', 'bob', 'charlie']
tasks:
  - name: Capitalize user names
    debug:
      msg: "{{ users | map('capitalize') | list }}"
```

```
vars:
  username: "{{ lookup('env', 'USER') | default('guest', true) }}"
tasks:
  - name: Greet user
    debug:
      msg: "Hello, {{ username }}"
```

Ansible Vault

- Ansible Vault est un outil intégré à Ansible pour la gestion sécurisée des données sensibles, telles que les mots de passe et les clés secrètes.
- Il permet de chiffrer et de déchiffrer des fichiers contenant des données sensibles.
- Pourquoi Utiliser Ansible Vault ?
 - ◆ **Sécurité** : Pour protéger les informations sensibles stockées dans vos playbooks et rôles.
 - ◆ **Conformité** : Pour répondre aux exigences de sécurité et de conformité en matière de gestion des données sensibles.
 - ◆ **Gestion Centralisée** : Pour stocker toutes les informations sensibles au même endroit.
-

Ansible Vault

- Utilisez la commande `ansible-vault encrypt mysecrets.yml` pour chiffrer des fichiers contenant des données sensibles.
- Vous devrez fournir un mot de passe maître pour chiffrer et déchiffrer ces fichiers.

```
$ ansible-vault encrypt mysecrets.yml
```

- Dans vos playbooks, utilisez la commande `ansible-vault edit mysecrets.yml` pour modifier des fichiers chiffrés en toute sécurité.

```
$ ansible-vault edit mysecrets.yml
```

- Vous pouvez stocker des variables chiffrées dans vos playbooks et y accéder en toute sécurité.

```
# Exemple de variable chiffrée dans un playbook
api_key: !vault |
    $ANSIBLE_VAULT;1.1;AES256
    66313964306361353865666231323539636330323135313966636630326637613339323437636133
```

LAB - TP

- Rôles Ansible
- Includes & tags
- Ansible-galaxy
- Templating et variables
- Les prompts et les facts
- Lookups, Plugins
- Ansible Vault

Ansible - Pour aller plus loin

- Développer ses propres modules
- Ansible Tower
- Inventaire dynamique
- Automatisation CI avec Ansible
- Combinaison Ansible + Terraform

Développer ses propres modules

- Les modules Ansible sont des éléments essentiels pour l'automatisation des tâches.
- Parfois, vous devez créer vos propres modules personnalisés pour répondre à des besoins spécifiques.
- Pourquoi Créer Vos Propres Modules ?
 - ◆ **Adaptation** : Les modules personnalisés permettent de répondre à des exigences uniques de votre infrastructure.
 - ◆ **Réutilisation** : Vous pouvez réutiliser vos modules personnalisés dans plusieurs playbooks.
 - ◆ **Complexité** : Pour gérer des opérations complexes ou des systèmes non pris en charge par les modules existants.

Création d'un Module Ansible

→ Exemple de code “Hello Word”

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

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()
```

Création d'un Module Ansible

→ Pour utiliser notre nouveau module on le définit le playbook

```
- hosts: localhost
  tasks:
    - name: Test de notre module
      test:
        register: result

    - debug: var=result
```

```
$ ansible-playbook test-playbook.yml
TASK [Test de notre module] *****
ok: [localhost]

TASK [debug] *****
ok: [localhost] => {
  "result": {
    "changed": false,
    "failed": false,
    "meta": {
      "result": "hello world"
    }
  }
}
```

Création d'un Module Ansible

- **from ansible.module_utils.basic import *:**
importation de la librairie permettant de créer des modules Ansible.
- **main():** le point d'entrée de notre module.
- **AnsibleModule():** c'est la classe qui nous permet de créer et manipuler notre module Ansible, comme par exemple la gestion des paramètres de notre module.
- **response = {"result" : "hello world"}:** ce sont les métadonnées de notre module sous forme d'un dictionnaire.
- **module.exit_json():** cette partie désigne la fin d'exécution de notre module, elle permet l'affichage des métadonnées et l'état de votre module.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

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()
```

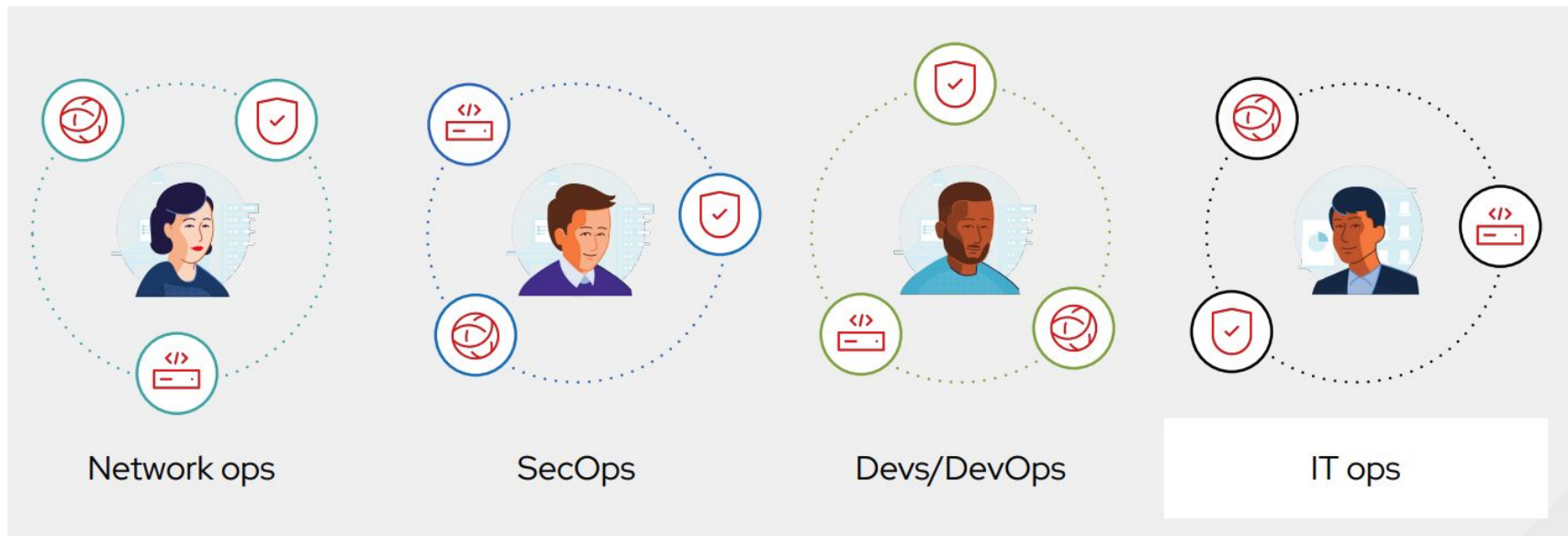
Inventaire dynamique

- L'inventaire peut également être constitué dynamiquement dans un environnement où les IP et nom des serveurs sont changeant (dynamique)
- Ces sources comprennent :
 - ◆ Cobbler (<http://cobbler.github.io/>)
 - ◆ Des API dans le Cloud : AWS, GCP, Azure, OpenStack...

Ansible Automation Platform - CI/CD

- La Red Hat Ansible Automation Platform est une solution d'automatisation et d'orchestration de niveau entreprise proposée par Red Hat.
- Conçue pour aider les organisations à automatiser, gérer et optimiser leur infrastructure informatique et leurs processus métier.
- Quelques caractéristiques et composants de Ansible Automation Platform :
 - ◆ Moteur d'Automatisation Ansible-core: exécution des tâches et des playbooks
 - ◆ Contrôle d'Accès Basé sur les Rôles (RBAC) : organiser et contrôler qui peut accéder et exécuter des tâches
 - ◆ Hub d'Automatisation : référentiel de contenu d'automatisation certifié, comprenant des rôles et des modules Ansible pré-construits.
 - ◆ APIs et Extensibilité et le Support

Ansible Automation Platform - CI/CD

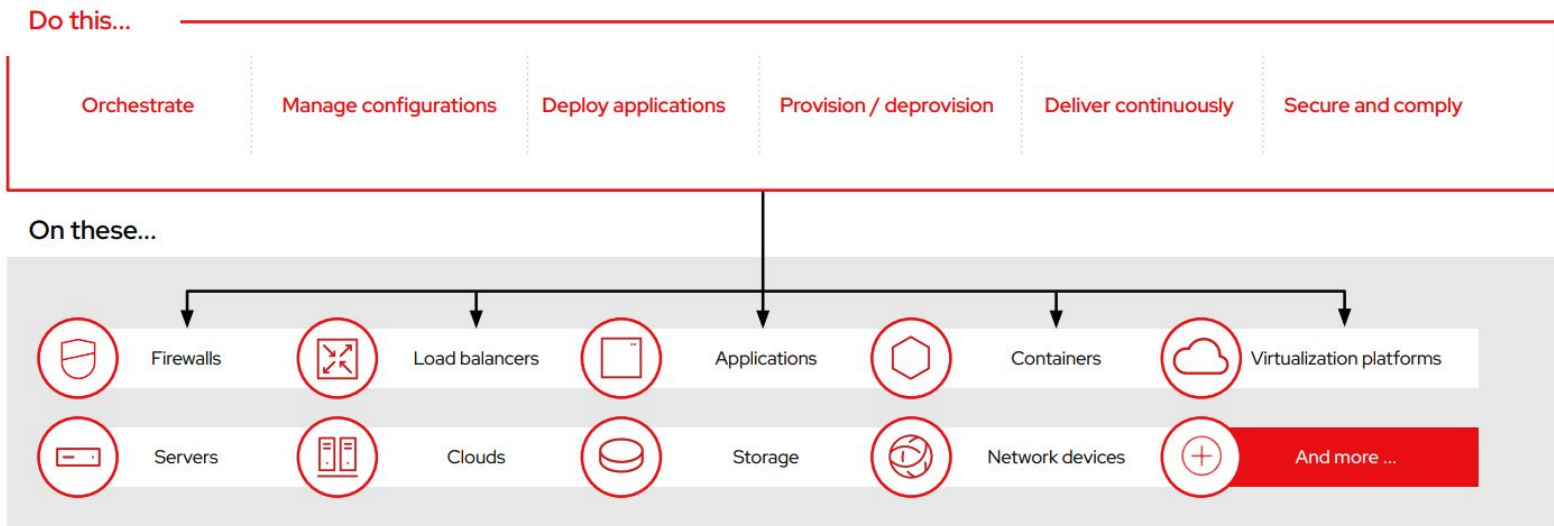


Source Red Hat

Ansible Automation Platform - CI/CD

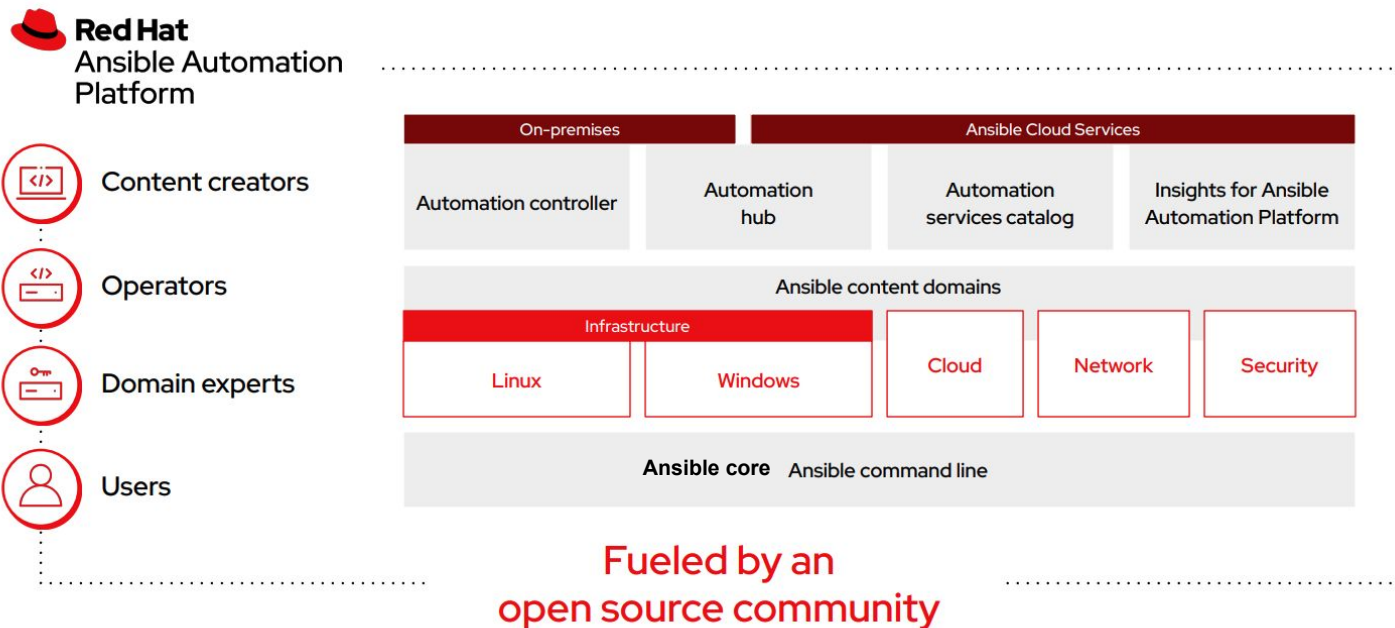
Automate the deployment and management of automation

Your entire IT footprint



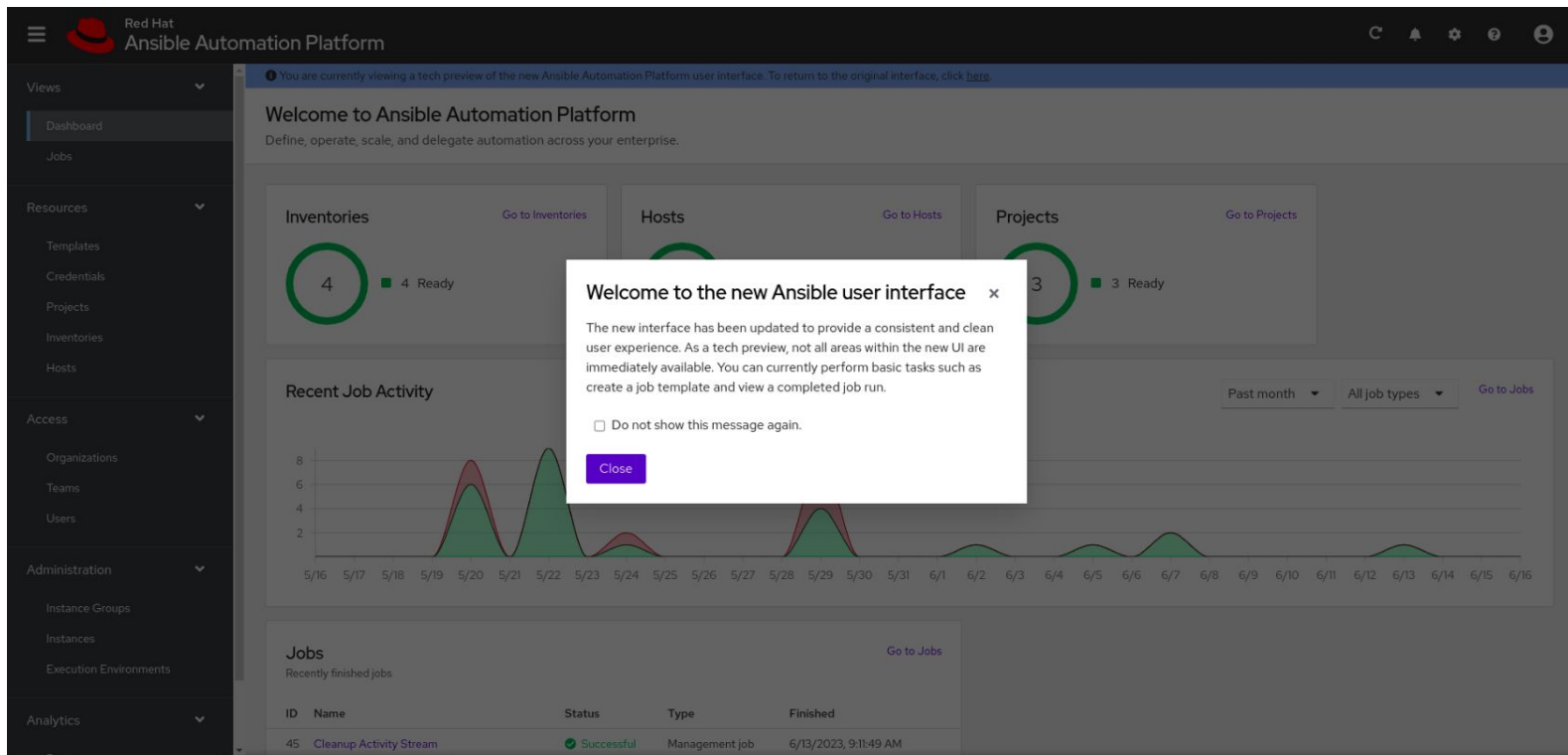
Source Red Hat

Ansible Automation Platform - CI/CD



Source Red Hat

Ansible Automation Platform - CI/CD



Ansible Automation Platform - CI/CD

Ansible Tower Interface Overview

Navigation Sidebar:

- VIEWS
 - Dashboard
 - Jobs
 - Schedules
 - My View
- RESOURCES
 - Templates
 - Credentials
 - Projects
 - Inventories
 - Inventory Scripts
- ACCESS
 - Organizations
 - Users
 - Teams
- ADMINISTRATION
 - Credential Types
 - Notifications

MY VIEW Dashboard

JOB TEMPLATES (8)

SEARCH	Q	KEY
Compact Expanded Name (Ascending) ▼		
Demo Job Template	Job Template	Progress: 3 green, 5 red
Example	Job Template	
Job template with slicing	Job Template	Progress: 2 red, 8 grey
May Job Template	Job Template	
New Template with Dependencies	Job Template	
New Workflow Job Template	Workflow Template	Progress: 2 green, 8 grey
WF in WF	Workflow Template	Progress: 1 red, 8 grey
WF using JT	Workflow Template	

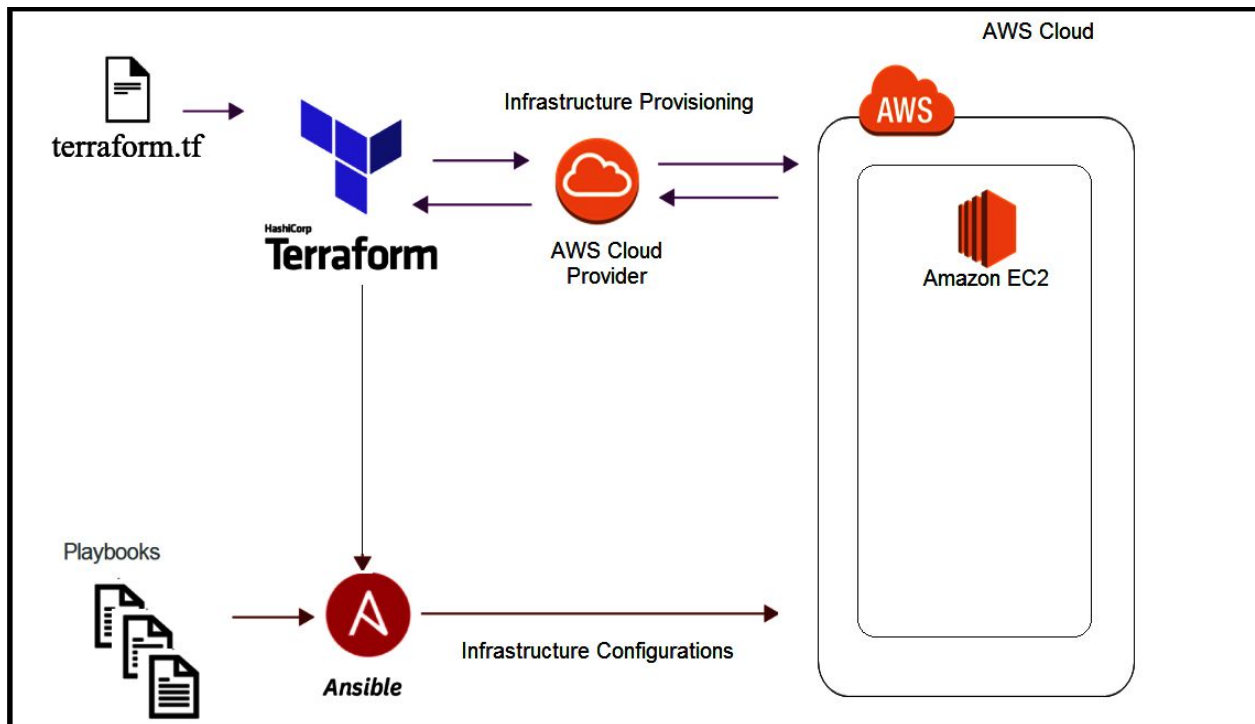
ITEMS 1 - 8

JOBS (16)

MY JOBS ALL JOBS

SEARCH	Q	KEY
Compact Expanded Finish Time (Descending) ▼		
111 - New Workflow Job Template	Workflow Job	
109 - Project from Git	SCM Update	
101 - WF in WF	Workflow Job	
102 - Job template with slicing	Playbook Run	
100 - New Workflow Job Template	Workflow Job	
87 - Demo Job Template	Playbook Run	
74 - Demo Job Template	Playbook Run	
23 - Demo Job Template	Playbook Run	
19 - Demo Job Template	Playbook Run	

Combinaison Ansible + Terraform



Combinaison Ansible + Terraform

- Ansible et Terraform sont deux outils d'automatisation complémentaires pour la gestion d'infrastructure.
- En les combinant, vous pouvez bénéficier de la puissance de Terraform pour la gestion de l'infrastructure et d'Ansible pour la configuration des serveurs.
- Cas d'Utilisation - Provisionnement de Machines Virtuelles
 - ◆ Utilisez Terraform pour provisionner des machines virtuelles (VMs) dans un cloud.
 - ◆ Utilisez Ansible pour configurer les VMs en installant des logiciels, des mises à jour, etc.
- Cas d'Utilisation - Gestion de Réseaux Virtuels
 - ◆ Utilisez Terraform pour créer des réseaux virtuels, des sous-réseaux et des règles de pare-feu.
 - ◆ Utilisez Ansible pour configurer les routeurs et les serveurs de sécurité dans le réseau virtuel.

Combinaison Ansible + Terraform

→ Cas d'Utilisation - Déploiement d'Applications

- ◆ Utilisez Terraform pour provisionner des ressources cloud (VMs, bases de données, etc.) pour votre application.
- ◆ Utilisez Ansible pour déployer et configurer l'application sur les ressources provisionnées.

→ Avantages :

- ◆ Bénéficie des points forts de ces deux outils pour la gestion de nos applications
- ◆ Isolation des Responsabilités : Terraform gère l'infrastructure, Ansible gère la configuration.
- ◆ Scalabilité : Vous pouvez étendre votre infrastructure et votre configuration de manière flexible.
- ◆ Réutilisation : Les modules Terraform et les rôles Ansible peuvent être réutilisés pour d'autres projets.
- ◆ IaC de toute notre Stack (Infra + applicatif) at la gérer de manière Idempotent