

Ansible

8 décembre 2018

Plan

1 Présentation d'Ansible

- Pourquoi avons-nous besoin de Ansible ?
- Définition ansible
- Terminologie Ansible
- les Avantages de l'utilisation de Ansible
- Les Fonctionnalité d'Ansible
- Architecture d'Ansible

2 Déploiement d'Ansible

Pourquoi avons-nous besoin de Ansible ?

- Considérez la routine fastidieuse d'administration d'un parc de serveurs.
 - Nous devons toujours garder à jour
 - Mettre à jour les modifications,
 - Copier les fichiers dessus,...
-
- Ces tâches rendent les choses très compliquées
 - Prennent beaucoup de temps.

Pourquoi avons-nous besoin de Ansible ?

- Auparavant, gestion manuelle des serveurs
- Installation manuelle des logiciels
- Mise à jours et modification des configurations manuellement

Pourquoi avons-nous besoin de Ansible ?

- Développement massive des data centers
- Complexité des applications hébergées
- Une forte croissance de publication de logiciel au sein de l'équipe de développement
- ⇒ Manque du rapidité du travail des développeurs
- Les administrateur sont influencé par le facteur temps
- Perte de temps dans les opération de configuration système
- ⇒ plein essor des outils de provisioning

Pourquoi avons-nous besoin de Ansible ?

- La solution ⇒ Ansible



c'est quoi ansible

- Ansible est un moteur d'automatisation simple qui automatise le cloud
- Gérer une infrastructure informatique
- Gérer le provisionnement
- Gérer la gestion de la configuration
- Gérer le déploiement d'applications
- Gérer les déploiements automatisés multi-environnements



c'est quoi Ansible

- l'orchestration de services les plus avancées
- Configuration des machines
- Installation des logiciels
- Mise à jours sans arrêt de services
- Décrire comment ces machines doivent être configurées ou quelles actions doivent être prise.

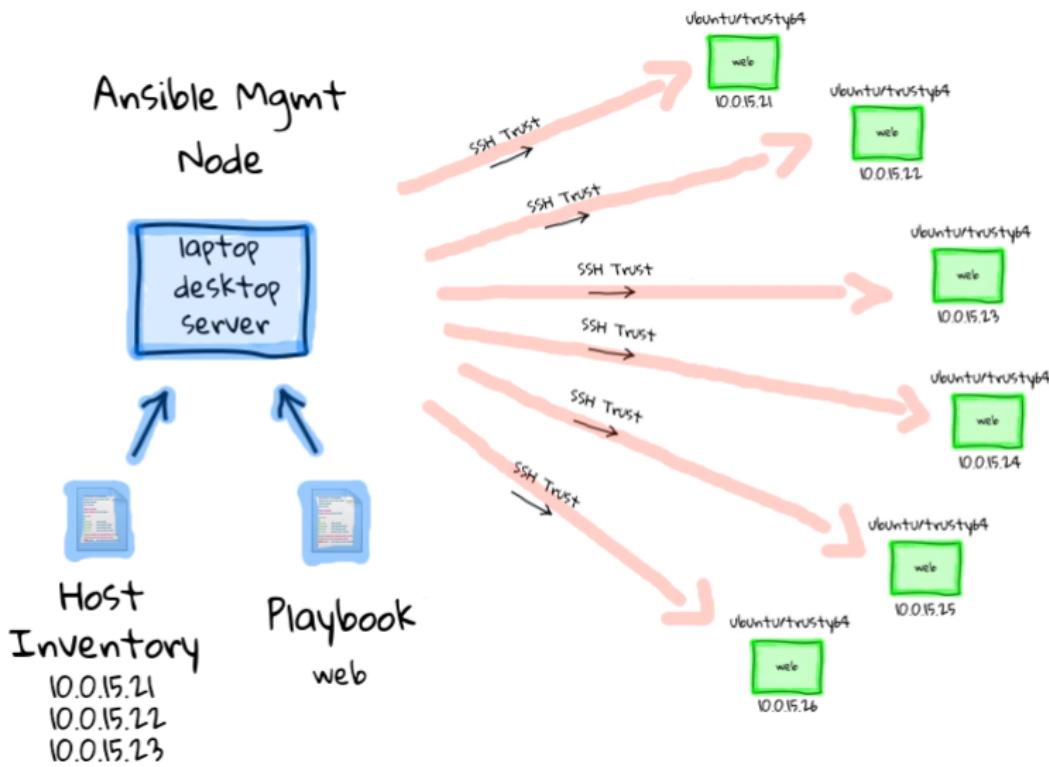


C'est Quoi Ansible

- Ansible doit uniquement être installé sur la machine de contrôle (la machine à partir de laquelle vous exécuterez les commandes)
- Aucun logiciel client n'est installé sur les ordinateurs du noeud.
- Il utilise SSH pour se connecter aux nœuds.
 - pouvoir être votre ordinateur portable.

C'est une solution simple à un problème compliqué.

C'est Quoi Ansible



c'est quoi Ansible

- Développé par Michael Dehaan en 2012.
- Il fonctionne sous un environnement Linux
- Racheté par Red Hat en 2015, premier fournisseur mondial de solutions logicielles Open Source.



c'est quoi Ansible

- Un logiciel open source
- Développé en langage Python
- S'appuie sur SSH pour la communication
- Un outils quasi-universelle
 - SSH et python livré en standard sur le plus part des serveurs



Terminologie Ansible

- **Contrôleur de machines** : ordinateur sur lequel est installé Ansible, chargé de l'exécution du provisioning sur les serveurs gérer.
- **Inventaire** : fichier d'initialisation contenant des informations sur les serveurs .
- **Playbook** : Le point d'accès aux l'approvisionnement d'Ansible, où l'automatisation est définie via des tâches utilisant le format YML.
- **Facts** : Variables globales contenant des informations sur le système, telles que les interfaces réseau ou le système d'exploitation.

Terminologie Ansible

- **Tâche** : bloc définissant une procédure unique à exécuter, par exemple, installer un package.
- **Module** : Un module résume une tâche système, exemple traiter des packages ou créer et modifier des fichiers.
- **Rôle** : Une manière prédéfinie d'organiser les playbooks et autres fichiers afin de faciliter le partage et la réutilisation d'un provisionnement.
- **Gestionnaires ou Handlers** : utilisés pour déclencher des changements d'état du service, tels que le redémarrage ou l'arrêt d'un service.

Simple :

- Utilisation d'une simple syntaxe écrite en YAML appelée playbooks .
- **YAML** est un langage de sérialisation des données lisible par l'homme.
- Aucune compétence particulière en matière de codage n'est requise
- Exécution les tâches dans l'ordre.
- Facilité d'installer .
- Démarrage rapide et simple.

Libre :

- Complètement libre.
- Sans requise de pare-feu supplémentaires sur les systèmes clients ou les hôtes à automatiser .
- Pas besoin de configurer séparément une infrastructure de gestion qui inclut
 - la gestion de l'ensemble de systèmes
 - le réseau
 - le stockage.



Puissant et flexible :

- Dispose de puissantes fonctionnalités permettant de modéliser les flux les plus complexes.
- Gérer l'infrastructure, les réseaux, les systèmes d'exploitation et les services
- Autonomie
- Fournir de modules prêt à l'emploi.
- Les capacités d'orchestrer l'environnement d'application, quel que soit l'endroit où il est déployé.



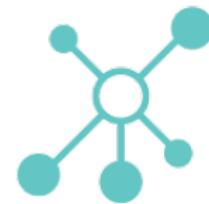
Efficace :

- Les modules Ansible fonctionnent via JSON
- Extensibilité avec des modules écrits dans un langage de programmation quelconque.
- Pouvoir personnaliser les module selon vos besoins.
- Par exemple, si un module d'envoi de messages en texte brut et que vous souhaitez également envoyer des images,
⇒ ajout les fonctionnalités d'envoi d'images



Approvisionnement :

- Les applications doivent avoir un cycle de vie
- rationaliser le processus d'amorçage et de démarrage des serveurs bare-metal ou des machines virtuelles, ou création des instances virtuelles du parti de modèles.
- Provisionnement des dernières plates-formes de cloud, hôtes et hyperviseurs virtualisés, périphériques réseau et serveurs sans système d'exploitation.



Approvisionnement :

- devient l'outil de connexion de tous vos pipelines de processus, en intégrant automatiquement bare-l'infrastructure à la gestion quotidienne.



- Permet de passer de manière transparente
 - à la gestion de la configuration
 - à l'orchestration
 - au déploiement d'applications
 - en utilisant le même langage d'automatisation simple et lisible par l'homme



Gestion de la configuration :

- Établir et maintenir la cohérence des performances de configuration .
- En enregistrant et en mettant à jour des informations détaillées décrivant le matériel et les logiciels d'une entreprise.
- Ces informations incluent les versions et les mises à jour appliquées aux packages de logiciels installés.
- les emplacements et adresses réseau des périphériques matériels.



Gestion de la configuration :

Par exemple

- Installer la nouvelle version de Tomcat sur toutes les machines d'une entreprise.
- difficulté d'actualiser manuellement chaque machine.
- Solution ⇒ instalation Tomcat une fois sur toutes vos machines grâce aux playbooks et à l'inventaire Ansible .
- ⇒ Exécutez le playbooks à partir de votre machine de contrôle et il sera installé sur tous vos nœuds.



Déploiement d'applications :

- Définir l'application avec Ansible
- Génération du déploiement avec Ansible Tower
- Gérer efficacement le cycle de vie complet de l'application
⇒ du développement à la production.



Déploiement d'applications :

Par exemple

- Pour déployer le moteur de servlet par défaut.
- Un certain nombre d'étapes doivent être franchies pour déployer le moteur.
- Déplacer une application .war d'un répertoire **dropins** vers un répertoire d'applications
- Ajouter un fichier **server.xml**
- Accédez à la page Web pour voir votre application.



Sécurité et conformité :

- Automatisés le processus d'analyse et de correction de la stratégie de sécurité
- Configuration d'informations de sécurité une fois dans le host de contrôle
- ⇒ Déploiement et intégration automatiquement dans tous les autres nœuds.
- Cryptage des informations d'identification récupérer (identifiant de l'utilisateur admin et mots de passe)



Orchestration :

- Définir l'interaction de plusieurs configurations
- ⇒ Assure que les éléments disparates peuvent être gérés dans leur ensemble.
- Création d'infrastructure alignée sur les applications qui peut être augmentée ou réduite en fonction des besoins de chaque application.
- Permet l'orchestration la demande métier sur les applications, les données et l'infrastructure.
- Définir les stratégies et les niveaux de service via des workflows automatisés, le provisioning et la gestion des modifications.

Orchestration :

Par exemple

- Pour déployer un nouveau site Web à la place de l'ancien mon site .
 - ① supprimer le site Web existant,
 - ② Déployer le nouveau site Web redémarrer l'équilibrEUR de charge (the load balancer) ou le cluster Web(si nécessaire).
- Remarquer des temps d'arrêt ⇒ a cause de suppression de trafic allant à ces machines via l'équilibrEUR de charge.
- Besoin d'une sorte de tâche préparatoire permettant à l'équilibrEUR de charge de mettre ce serveur Web en mode de maintenance

Orchestration :

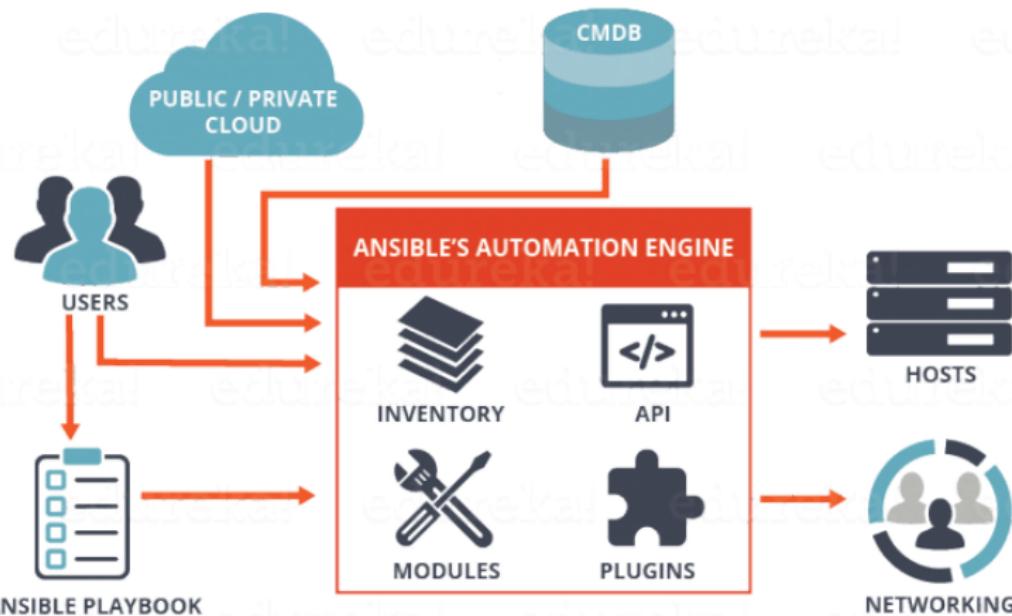
Par exemple

- Désactiver temporairement le trafic
- signifie l'ajout d'un bloc , indique qu'une tâche préalable consiste à désactiver le nœud Web dans l'équilibrEUR de charge.
- désactivons le trafic ⇒ pré-tâche
- Sortant du mode maintenance et activation de trafic vers ce nœud Web ⇒ post-tâche



Architecture d'Ansible

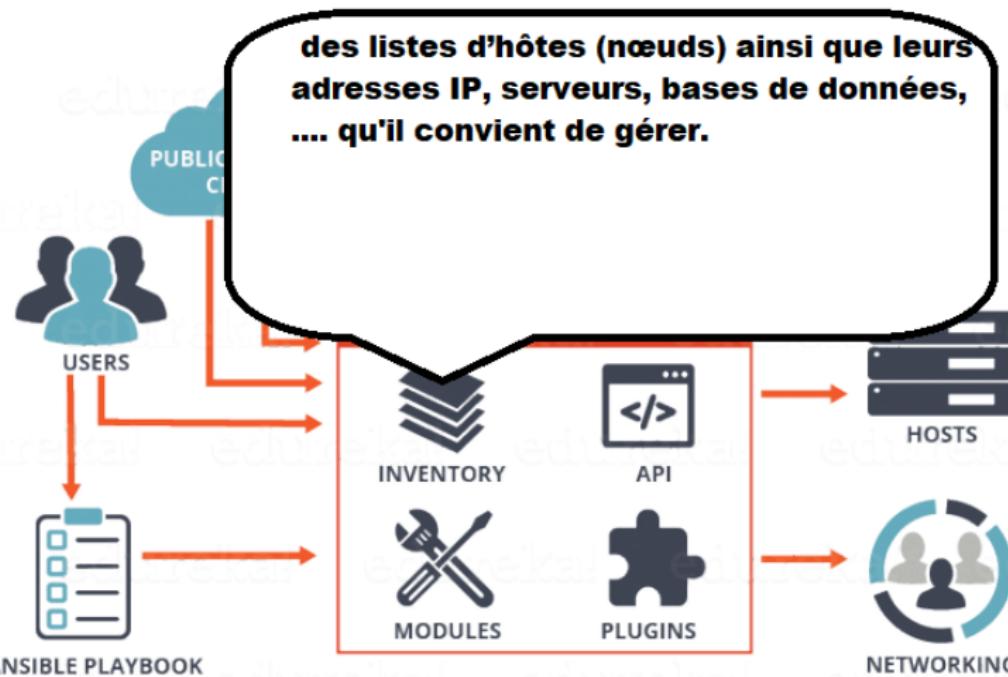
ANSIBLE ARCHITECTURE



Architecture d'Ansible

- Le moteur d'automatisation Ansible interagit directement avec les utilisateurs
- ⇒ écrire des playbooks pour exécuter le moteur d'automatisation Ansible .
- Il interagit également avec les services Cloud et la base de données de gestion de la configuration (CMDB).
- Le moteur d'automation Ansible comprend :

Les Inventaires

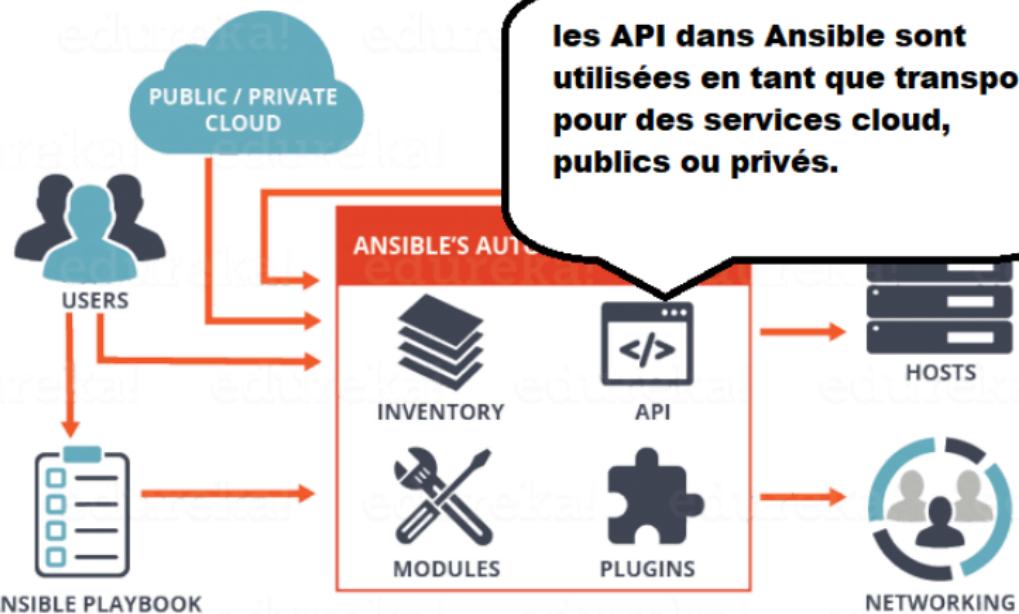


Les Inventaires

- gère plusieurs hôtes en parallèle.
- ⇒ le fichier d'inventaire d'Ansible
- ⇒ Enregistrées dans l'emplacement
`/etc/ansible/hosts`.
- Spécifier un fichier d'inventaire à l'aide de l' option :**-i <path>**
- Inventaire dynamique : Utilisation de plusieurs fichiers d'inventaire simultanément et extraction des inventaire à partir de sources dynamiques ou de cloud.

Les API

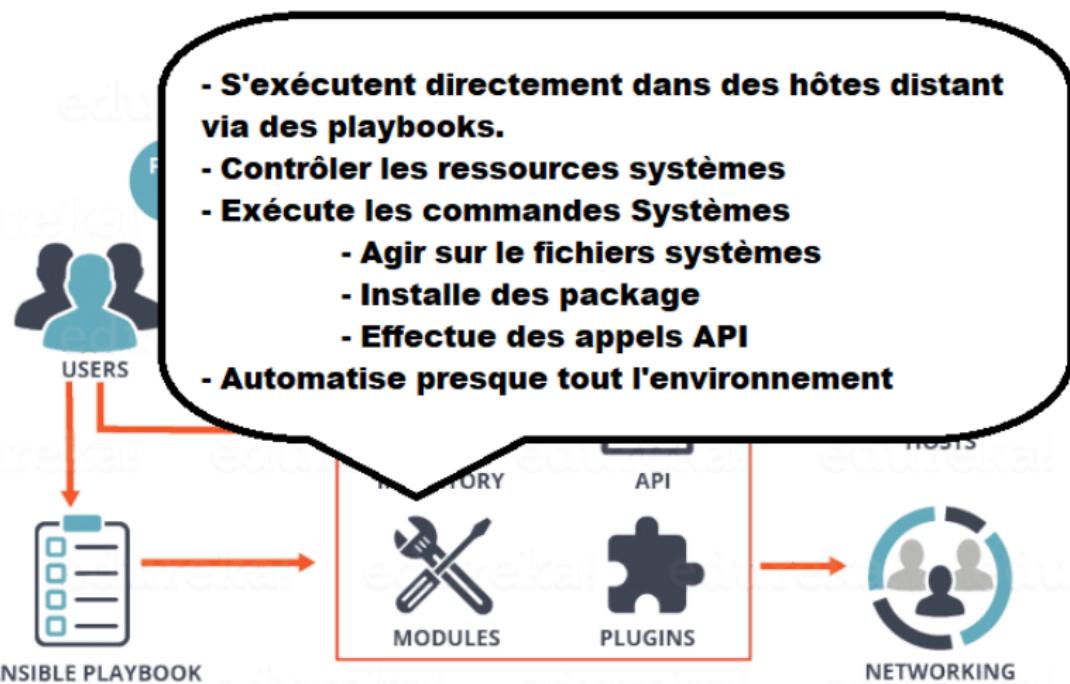
ANSIBLE ARCHITECTURE



Les API

- **l'API Ansible** : Python :Contrôle les nœuds
- **Les API REST** : fournir un accès aux ressources via des chemins URI
- **Ansible Tower** : Rendre Ansible utilisable via un service Web.

Les modules



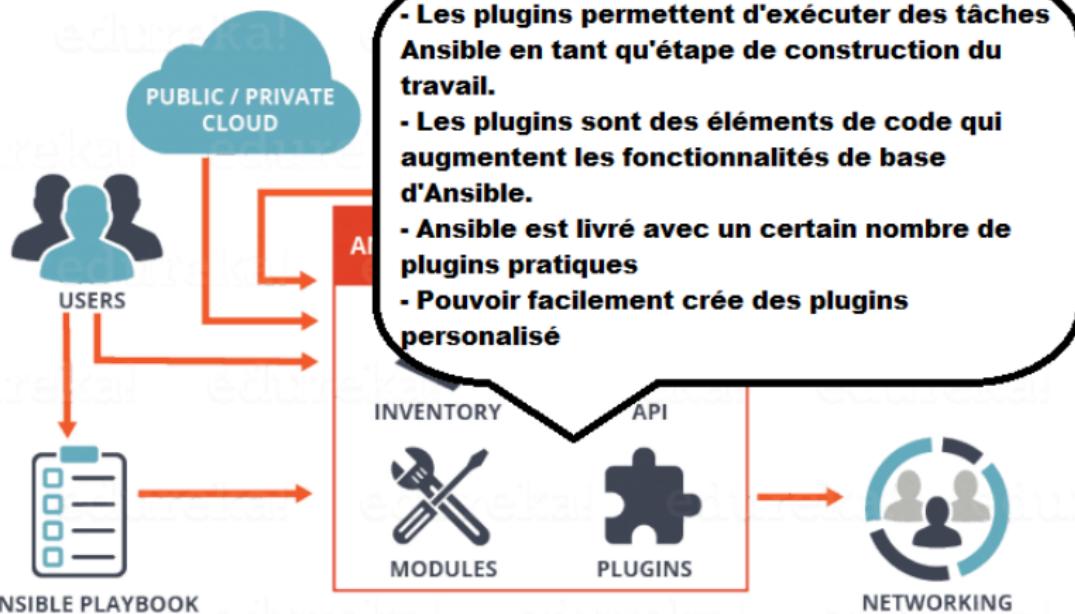
Les modules

Par exemple

- Les Modules de cloud : ex : cloudformation, crée ou supprime une pile de formation de cloud AWS
- Les modules de base de données :ex : mssql_db, gère les bases de données MYSQL des hôtes distants.
- Modules de surveillance :ex : Nagios , Effectuer des tâches courantes dans Nagios relatives aux temps d'arrêt et aux notifications.
- Modules de stockage : ex ,gluster_peer ,Attache, Détache des pairs au cluster
- Modules de notification : ex : rocketchat , Envoie des notifications à Rocket Chat

Les plugins

ANSIBLE ARCHITECTURE



les plugins

- **plugins d'Action** : agissent conjointement avec les modules pour exécuter les actions requises par les tâches du playbook
- Exécution automatiquement en arrière-plan
- **plugins cache** : implémentation du mécanisme de cache d'arrière-plan
- Permettant à Ansible de stocker des facts collectés
- stocker des données source d'inventaire sans avoir récupérer les données à partir de la source.

les plugins

- **Callback plugins** :Permettent de connecter à des événements Ansible afin d'afficher le journal.
- **Plugins de rappel** :permettant d'ajouter de nouveaux comportements à Ansible lors de la réponse à des événements
- **Plugins de connexion** : permettant de se connecter aux hôtes cibles pour lui permettre d'exécuter des tâches.
- **Les plugins de recherche** permettant d'accéder à des données provenant de sources extérieures.

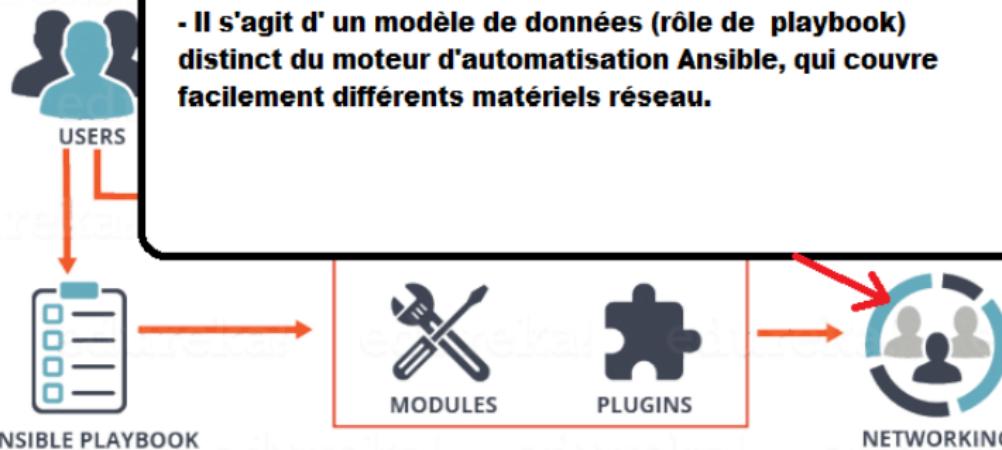
architecture Ansible

L'architecture d'Ansible contient quelques composants supplémentaires

Networking

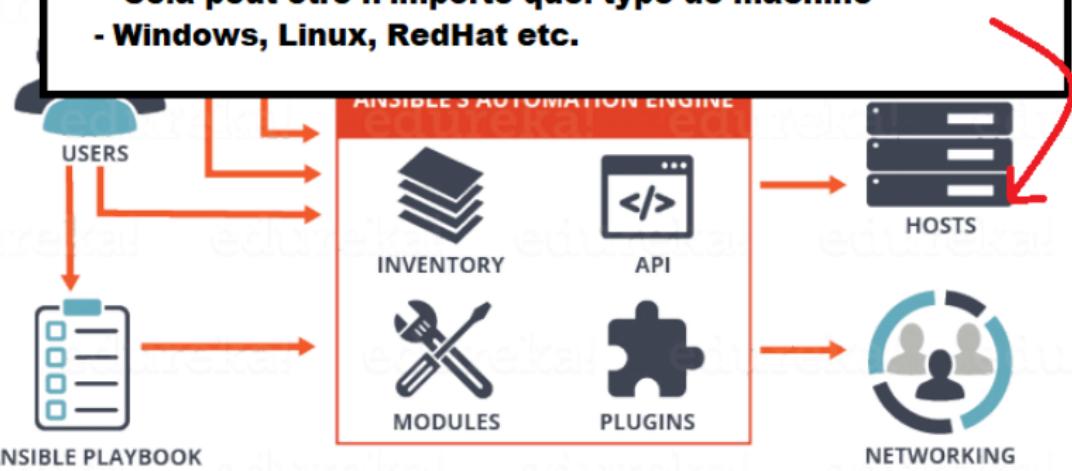
ANSIBLE ARCHITECTURE

- Il s'agit d'un modèle de données (rôle de playbook) distinct du moteur d'automatisation Ansible, qui couvre facilement différents matériels réseau.



Hosts

- les hôtes de l'architecture Ansible ne sont que des systèmes de nœuds qui sont automatisés par Ansible.
- Cela peut être n'importe quel type de machine
- Windows, Linux, RedHat etc.



Playbooks

ANSIBLE ARCHITECTURE

- Les playbooks sont de simples fichiers écrits au format YAML
- Décrivant les tâches à exécuter par Ansible.
- Les Playbooks peuvent déclarer des configurations, mais ils peuvent également orchestrer les étapes de tout processus de commande manuelle, même s'il contient des instructions de saut.
- Ils peuvent lancer des tâches de manière synchrone ou asynchrone.



ANSIBLE PLAYBOOK



MODULES



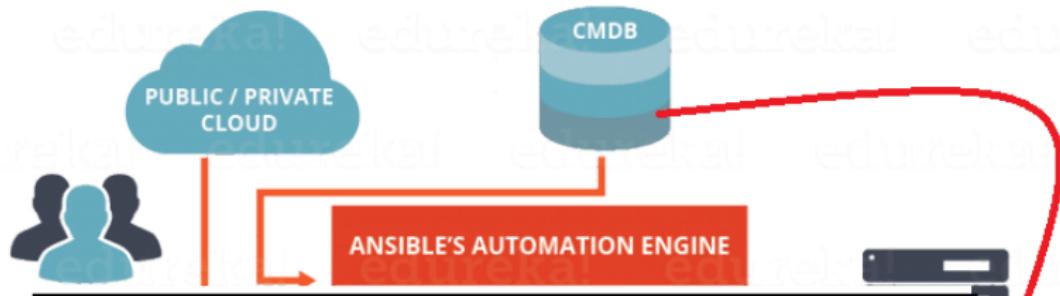
PLUGINS



NETWORKING

CMDB

ANSIBLE ARCHITECTURE



- Il s'agit d'un référentiel faisant office de dépôt de données pour les installations informatiques.
- Il contient des données relatives à une collection d'actifs informatiques (communément appelés éléments de configuration), ainsi que pour décrire les relations entre ces actifs.

Cloud

ANSIBLE ARCHITECTURE



- C'est un réseau de serveurs distants hébergés sur Internet pour stocker, gérer et traiter les données, plutôt qu'un serveur local.
- Pouvoir lancer des ressources et instances sur le cloud et vous connecter à vos serveurs.



ANSIBLE PLAYBOOK



MODULES



PLUGINS



HOSTS



NETWORKING

Plan

1 Présentation d'Ansible

2 Déploiement d'Ansible

- Installation Ansible
- Confirmez l'installation Ansible
- Configurer Ansible pour WinRM
- Configuration de l'accès SSH aux hôtes Ansible
- Configuration des hôtes Ansible
- Utilisation des commandes simples Ansible
- Les commandes AD-HOC

- Ansible est un outil disponible en ligne de commande.
- Automatisation de configuration de système et le déploiement de logiciels est son objectif principal .

GitHub

- Installation direct à partir de la source sur GitHub :
- Cloner le référentiel : **\$ git clone https://github.com/ceph/ceph-ansible.git**
- Décider quelle branche de ceph-ansible vous souhaitez utiliser.
- Choisir parmi les branches stables ou Utiliser la branche principale :**\$ git checkout \$branch**
- utilisation de pip et les exigences fournies.txt pour installer Ansible et les autres bibliothèques Python nécessaires : **\$ pip install -r requirements.txt**

Ansible sur RHEL et CentOS

- Acquérir Ansible sur RHEL et CentOS en installant à partir du canal Ansible
- Sur RHEL :
 - `$ subscription-manager repos
--enable=rhel-7-server-ansible-2-rpms`
- CentOS :
 - `$ sudo yum install ansible`

Ansible sur Ubuntu

- Acquérir Ansible sur Ubuntu en utilisant le PPA Ansible

```
$ sudo add-apt-repository ppa:ansible/ansible  
$ sudo apt update  
$ sudo apt install ansible
```

Installez les paquets Python

```
sudo apt-get install python-pip pip
```

Installez WinRM (Windows Remote Management)

```
sudo apt-get install pywinrm
```

Confirmez l'installation Ansible

- Testez l'installation Ansible en récupérant sa version à l'aide de la commande suivante.

```
ansible --version
ansible 2.4.1.0
config file = None
configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
ansible python module location = /usr/local/lib/python2.7/dist-packages/ansible-2.4.1.0-py2.7.egg/ansible
executable location = /usr/local/bin/ansible
python version = 2.7.13 (default, Jan 19 2017, 14:48:08) [GCC 6.3.0 20170118]
```

Configurer Ansible pour WinRM

- Allez dans le répertoire Ansible.

```
cd /etc/ansible
```

- Editer le **hosts** fichier avec votre éditeur de texte préféré.

```
nano hosts
```

Configurer Ansible pour WinRM

- Ajoutez les lignes suivantes au bas du windowsgroupe

```
[windows]
server1.domain.local
server1.domain.local
```

Configurer Ansible pour WinRM

- Nécessité de créer un emplacement de stockage chiffré contenant les identificateurs d'accès au serveur Windows.
- création d'un nouveau fichier **yml** crypté .
- Le nom du fichier doit correspondre au groupe d'hôtes auquel il doit s'appliquer.
- le groupe créé est appelé windows, le fichier le sera windows.yml.

```
mkdir group_vars
```

```
cd group_vars
```

```
ansible-vault create windows.yml
```

Configurer Ansible pour WinRM

- Dans le fichier yml, ajoutez les paramètres suivants.

```
ansible_ssh_user: _your_ssh_user_
ansible_ssh_pass: _your_ssh_pass_
ansible_ssh_port: 5986
ansible_connection: winrm
ansible_winrm_server_cert_validation: ignore
```

Configuration de l'accès SSH aux hôtes Ansible

- Communication avec les ordinateurs clients via SSH.
- Pouvoir générer l'authentification SSH basée sur un mot de passe, l'utilisation de clés SSH peut aider à simplifier les choses.
- Sur le serveur Ansible, utilisez la commande pour imprimer le contenu du fichier de clé publique SSH de votre utilisateur non root sur la sortie du terminal :

```
$ cat ~/.ssh/id_rsa.pub
```

Configuration de l'accès SSH aux hôtes Ansible

- Copier la sortie résultante dans votre presse-papiers, puis ouvrir un nouveau terminal et connecter à l'un de vos hôtes Ansible à l'aide de SSH :

```
$ ssh sammy@ansible_host_ip
```

- Basculez vers l' utilisateur root de la machine cliente :

```
$ su -
```

Configuration de l'accès SSH aux hôtes Ansible

- En tant qu'utilisateur **root** , ouvrez le **authorized_keys** dans le **~/.ssh** répertoire :

```
# nano ~/.ssh/authorized_keys
```

- Dans le fichier, collez votre clé SSH de l'utilisateur du serveur Ansible, puis enregistrez le fichier et fermez l'éditeur.
- Exécutez ensuite la commande **exit** pour retourner à l'utilisateur non root de l'hôte :

```
# exit
```

configurer un serveur

- Générer la clé publique qui permettra la communication entre le serveur et les hôtes :`ssh-keygen`
- Provisionner le fichier `/etc/ansible/hosts` en spécifiant uniquement les adresses IP ou le nom du serveur

```
[nomdugroupe]
@IPduserveur
```

configurer un serveur

- Certaines variables permettent de préciser le type de connexion :
 - le nom d'utilisateur
 - le port
 - le mot de passe à utiliser
- Connexion à l'hôte :
 - ansible_connection : type de connexion à l'hôte (local, smart, ssh ou paramiko)

- Connexion SSH :

- **ansible_ssh_host** : Nom de l'hôte.
- **ansible_ssh_port** : Numéro du port (si différent de 22).
- **ansible_ssh_user** : Nom de l'utilisateur.
- **ansible_ssh_pass** : Password à utiliser.
- **ansible_ssh_private_key_file** : fichier de clé privée à utiliser.

- Escalade de privilèges :

- **ansible_become** : Équivalent de ansible_sudo.
- **ansible_become_method** : Permet de définir la méthode d'escalade de privilège.
- **ansible_become_user** : Permet de définir l'utilisateur à impersonnifier.
- **ansible_become_pass** : Permet de définir le mot de passe élévant les privilèges utilisateur.

- Paramètres d'environnement d'hôte distant :
 - **ansible_shell_type** : Type de shell distant.
 - **ansible_python_interpreter** : L'interpréteur python de l'hôte distant.
 - **ansible__interpreter** : fonctionne pour n'importe quel exemple. interpréteur, Ruby par

Configuration des hôtes Ansible

- Ansible conserve la trace de tous les serveurs connus via un fichier hosts.
- Devoir configurer ce fichier avant de pouvoir commencer à communiquer avec les autres ordinateurs.
- Ouvrez le fichier avec les sudo privilèges, comme ceci :

```
$ sudo nano /etc/ansible/hosts
```

Configuration des hôtes Ansible

- Dans le fichier, un certain nombre d'exemples de configurations qui ont été commentées .
- Ces exemples ne fonctionneront pas pour nous car les hôtes énumérés dans chacun d'eux sont composés.
- Nous conserverons toutefois ces exemples dans le fichier pour nous aider à configurer si nous voulons implémenter des scénarios plus complexes.

Configuration des hôtes Ansible

- Le fichier hosts est assez flexible et peut être configuré de différentes manières.
- La syntaxe que nous allons utiliser, ressemble à ceci :

```
[group_name]
alias ansible_ssh_host=your_server_ip
```

- **group_name** une balise organisationnelle qui vous permet de faire référence à tous les serveurs énumérés sous un mot **alias**
- **alias** s'agit simplement d'un nom pour désigner un serveur spécifique.

Configuration des hôtes Ansible

- Imaginons avoir trois serveurs que nous allons contrôler avec Ansible.
- ces serveurs sont accessibles à partir du serveur Ansible en tapant :

```
$ ssh root@ansible_host_ip
```

Configuration des hôtes Ansible

- configuré correctement aucun mot de passe est demander.
- Supposant que les adresses IP de nos hôtes sont **203.0.113.1**, **203.0.113.2** et **203.0.113.3**.
- pouvoir référencer à ces individuellement **host1**, **host2** et **host3**, ou en groupe avec le nom serveur.

Configuration des hôtes Ansible

- Dans le bloc , ajouter au fichier hosts pour accomplir ceci :

/ etc / ansible / hosts

```
[servers]
host1 ansible_ssh_host=203.0.113.1
host2 ansible_ssh_host=203.0.113.2
host3 ansible_ssh_host=203.0.113.3
```

Configuration des hôtes Ansible

- Les hôtes peuvent être dans plusieurs groupes et les groupes peuvent configurer des paramètres pour tous leurs membres
- Avec les paramètres actuels, essayant à connecter à l'un de ces hôtes avec Ansible
- la commande échouerait

Configuration des hôtes Ansible

- clé SSH est intégrée pour l' utilisateur root sur les systèmes distants et Ansible essaiera par défaut de se connecter en tant qu'utilisateur actuel.
- Une tentative de connexion obtiendra cette erreur :

Output

```
host1 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh.",
    "unreachable": true
}
```

Configuration des hôtes Ansible

- Sur le serveur Ansible, utiliser un utilisateur appelé sammy .
- Ansible va essayer de se connecter à chaque hôte avec ssh sammy@server.
- Cela ne fonctionnera pas si l' utilisateur sammy n'est pas également sur le système distant.

Configuration des hôtes Ansible

- Créer un fichier qui indique à tous les serveurs du groupe "serveurs" de se connecter en tant qu'utilisateur **root** .
- Créer un répertoire dans la structure de configuration Ansible appelée **group_vars**.
- Dans ce dossier, créer des fichiers au format **YAML** pour chaque groupe que nous voulons configurer :

```
$ sudo mkdir /etc/ansible/group_vars  
$ sudo nano /etc/ansible/group_vars/servers
```

Configuration des hôtes Ansible

- Nous pouvons mettre notre configuration ici. Les fichiers YAML commencent par "—", assurez-vous de ne pas oublier cette partie

```
/ etc / ansible / group_vars / servers
```

```
---  
ansible_ssh_user: root
```

- Enregistrez et fermez ce fichier lorsque vous avez terminé.
- Pour les détails de configuration pour chaque serveur
- Ces détails se trouve dans un fichier
`/etc/ansible/group_vars/all`.
- pouvoir configurer les hôtes individuels en créant des fichiers dans un répertoire `/etc/ansible/host_vars`

Utilisation des commandes simples Ansible

- Envoyez une commande ping à tous les serveurs que vous avez configurés en tapant :

```
$ ansible -m ping all
```

Sortie Ping

```
host1 | SUCCESS => {  
    "changed": false,  
    "ping": "pong"  
}
```

```
host3 | SUCCESS => {  
    "changed": false,  
    "ping": "pong"  
}
```

```
host2 | SUCCESS => {  
    "changed": false,  
    "ping": "pong"  
}
```

Utilisation des commandes simples Ansible

- Il s'agit d'un test de base permettant de s'assurer que Ansible dispose d'une connexion à tous ses hôtes

Utilisation des commandes simples Ansible

- le **all** signifie tous les hôtes.
- Spécification de un groupe :

```
$ ansible -m ping servers
```

- Spécification d'un hôte individuel :

```
$ ansible -m ping host1
```

Utilisation des commandes simples Ansible

- spécification de plusieurs hôtes en les séparant par deux points :

```
$ ansible -m ping host1:host2
```

Utilisation des commandes simples Ansible

- Le module "shell" nous permet d'envoyer une commande de terminal à l'hôte distant et de récupérer les résultats.
- Par exemple, pour connaître l'utilisation de la mémoire sur notre machine **hôte1**, nous pourrions utiliser :

```
$ ansible -m shell -a 'free -m' host1
```

Sortie shell

	total	used	free	shared	buffers	cached
Mem:	3954	227	3726	0	14	93
-/+ buffers/cache:		119	3834			
Swap:	0	0	0			

Les commandes AD-HOC

- Les commandes Ad-Hoc sont utilisées pour vérifier rapidement le fonctionnement et pas besoin d'être enregistrés (comme les Playbooks) pour une utilisation ultérieure.
- si un redémarrage soudain est nécessaire sur des centaines de serveurs
- s'il est nécessaire de vérifier la disponibilité des serveurs sur le réseau,
- **les commandes ad hoc peuvent être utilisés.**

Quelques commandes Ad-Hoc

#Syntaxe

```
$ansible <host> -m <module> -a <arguments>
```

#Afficher le nom de machine (hostname) du nœud en utilisant le module Shell

```
$ansible host1 -m shell -a 'hostname'
```

#Transférer un fichier à tous les serveurs web

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

#Supprimer les répertoires et les fichiers qui contiennent récursivement

```
$ansible webservers -m file -a "dest=/var/log/httpd" state=absent"
```

#Vérifier si le paquet httpd est bien installé

```
$ansible webservers -m yum -a "name=httpd state=present"
```

#Redémarrer le service httpd

```
$ansible webservers -m service -a "name=httpd state=restarted"
```

Plan

- 3 Mise en œuvre de playbooks
- 4 Configuration de playbooks complexes
- 5 Gestion des variables et des inclusions
- 6 Mise en œuvre du contrôle des tâches
- 7 Mise en œuvre de modèles Jinja2
- 8 Mise en œuvre de rôles
- 9 Mise en œuvre d'Ansible Vault

Les Playbooks

- Écrit en YAML qui est un langage simple
- Une manière différente d'utiliser ansible que dans le mode d'exécution de tâches ad hoc
- Les modules ansible sont utilisés dans le Playbook pour exécuter l'opération.
- la base d'un système de gestion de configuration
- La base de déploiement multi-machines

Les Playbooks

- Adapté au déploiement d'applications complexes.
- Déclarer des configurations
- Orchestrer les étapes de tout processus de commande manuelle
- Même si différentes les étapes rebondir entre des ensembles de machines dans des ordres particuliers.
- Ils peuvent lancer des tâches de manière synchrone ou asynchrone.
- Pouvoir exécuter /usr/bin/ansibleprogramme pour des tâches ad-hoc

Les Playbooks

- Plus susceptibles d'être conservés dans le système de contrôle de version
- Utilisés pour repousser la configuration
- pour assurer que les configurations de systèmes distants sont conformes aux spécifications.
- Appliquer une politique sur les nœuds distants.

Les Playbooks

- Assure le suivi des serveurs
- la répartition de charge
- Les playbooks contiennent un groupe d'hôtes et un ensemble de tâches.
- Un playbook est un scénario décrivant les actions qui seront réalisées par les serveurs
 - les paquets à installer
 - les fichiers à créer
 - les services à démarrer ou arrêter, ...

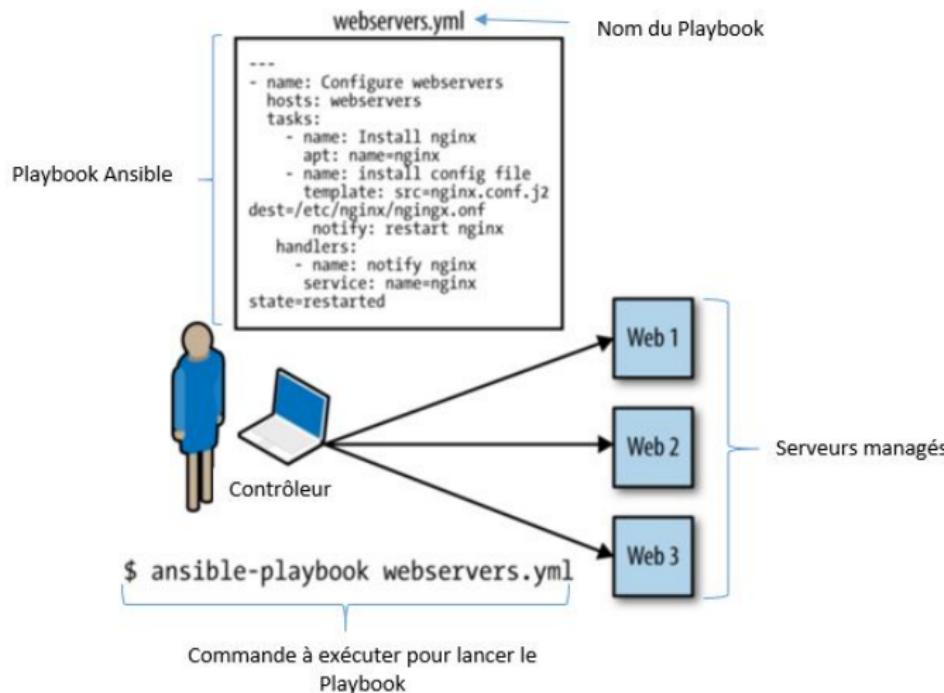
Les Playbooks



Explication

- Les Playbooks sont utilisés du la simple configuration
- à la gestion des nœuds distants
- jusqu'au déploiement avancé
- impliquant des mises à jour

Exécution d'un Playbook Ansible sur les serveurs managés Web1, Web2 et Web3



un Playbook de configuration d'un serveur Web via nginx

```
- name: Configure webserver with nginx
hosts: webservers
sudo: True

tasks:
- name: install nginx
apt: name=nginx update_cache=yes
- name: copy nginx config file
copy: src=files/nginx.conf dest=/etc/nginx/sites-available/default
- name: enable configuration
file: >
dest=/etc/nginx/sites-enabled/default
src=/etc/nginx/sites-available/default
state=link|
- name: copy index.html
template: src=templates/index.html.j2 dest=/usr/share/nginx/html/index.html
mode=0644
- name: restart nginx
service: name=nginx state=restarted
```

Sortie du Playbook de configuration d'un serveur Web via nginx

```
PLAY [Configure webserver with nginx] ****
GATHERING FACTS ****
ok: [testserver]

TASK: [install nginx] ****
changed: [testserver]

TASK: [copy nginx config file] ****
changed: [testserver]

TASK: [enable configuration] ****
ok: [testserver]

TASK: [copy index.html] ****
changed: [testserver]

TASK: [restart nginx] ****
changed: [testserver]

PLAY RECAP ****
testserver : ok=6    changed=4    unreachable=0    failed=0
```

Voici quelques Commandes de base

- **ansible-playbook –help** : Afficher l'aide
- **ansible-playbook script.yml** : Lancer le playbook script.yml, attention il faut que le playbook soit dans le même dossier
- **ansible-playbook /etc/ansible/script.yml** : Lancer le playbook script.yml peu importe où vous vous trouvez dans l'arborescence de votre poste

Voici quelques Commandes de base

- **ansible-playbook -i mesjolishosts script.yml** :
Lorsque l'option i n'est pas renseignée, Ansible prend par défaut le fichier /etc/ansible/hosts mais dans ce cas-ci on prend le fichier mesjolishosts dans le dossier courant
- **ansible-playbook -i mesjolishosts script.yml -e MAJOLIEVARIABLE=PC-PUPUCE** : -e extra-vars pour ajouter des variables
- **ansible-playbook script.yml -vvv** : -v verbose mode, -vvv encore plus de verbosité, -vvvv pour débuguer

créer un playbooks

- Un playbook est composé des éléments suivants :
 1. hosts
- fait référence au fichier d'inventaire des hôtes
- exemple :

```
hosts: all
```

créer un playbooks

2. tasks

- fait référence aux tâches qui seront exécutées sur les hôtes.
- exemple :

```
tasks:  
  - name: ensure apache is at the latest version  
    yum:  
      pkg: httpd  
      state: latest
```

créer un playbooks

3. vars (optionnel)

- fait référence à certaines variables tels que les ports d'écoute ou le nombre de connexion concurrentes maximum.
- exemple :

```
vars:  
    http_port: 80
```

créer un playbooks

4. remote_user (optionnel)

- fait référence à l'utilisateur distant disposant des priviléges les plus élevés.
- exemple :

```
handlers:  
  - name: restart apache  
    service:  
      name: httpd  
      state: restarted
```

créer un playbooks

4. handlers

- fait référence aux actions annexes (redémarrage de service, par exemple).
- exemple :

```
handlers:  
  - name: restart apache  
    service:  
      name: httpd  
      state: restarted
```

Lancer un script sur un host

- Playbook script.yml

```
---
```

```
- name: Transfer and execute a script
  hosts: PC-PUPUCE
  tasks:
    - name: Execute the script
      script: files/script.sh
```

Explication

- Un fichier YAML (extension yml) débute toujours par trois tirets,
- c'est une convention.
- Nommer la tâche
- renseigner le host sur lequel on souhaite que s'exécute le playbook.
- Définir la tâche à accomplir.

Ajouter la clé d'un dépôt

- Utilisation du playbook : ansible-playbook apt_key.yml
- Ajouter la clé d'un dépôt sur le(s) host(s) : par le module apt-key
- Playbook apt_key.yml

```
---
- name: Add an APT signing key
  hosts: server
  tasks:
    - name: Add the APT signing key
      apt_key: url=http://www.dotdeb.org/dotdeb.gpg state=present
```

Explication

- le playbook concerne non pas un seul host mais un groupe (servers),
- on peut mettre un host
- un groupe (ou sous-groupe)
- all pour tous les hosts.

Lancer un scan sur le(s) host(s)

- Utilisation du playbook : ansible-playbook clamscan.yml
-e host=SRV-WEB
- Playbook clamav.yml

```
---
- name: Scan du serveur
  hosts: "{{ host }}"
  tasks:
    - name: Mise à jour des bases de ClamAV
      command: freshclam
    - name: Scan du serveur
      command: /usr/bin/clamscan -r / --exclude-dir=/sys/ --exclude-dir=/proc/ --exclude-dir=/dev/ --infected
```

Explication

- Utilisation du module **command** pour lancer la commande **clamscan**.
- préciser le chemin complet de l'exécutable.
- La nouveauté c'est bien évidemment host qui est une variable, elle sera remplacée par **SRV-WEB** car on a utilisé -e **host=SRV-WEB**.

Rebooter le(s) host(s)

- Utilisation du playbook : ansible-playbook reboot.yml -e host=all
- Playbook reboot.yml

```
---
- name: Reboot all hosts
  hosts: "{{ host }}"
  tasks:
    - name: Reboot hosts
      command: /sbin/reboot
```

Copier un fichier (de configuration par exemple) sur le(s) host(s)

```
ansible-playbook copy.yml -e 'host=server file=ntp.conf dest=/etc/ntp.conf' # vous remarquerez les guillemets pour protéger les variables #
```

● Playbook copy.yml

```
---
```

```
- name: Copy a file
  hosts: "{{ host }}"
  tasks:
    - name: copy the file
      copy: src=files/{{ file }} dest={{ dest }} owner=root group=root mode=0644 backup=yes
```

Copier les fichiers de configuration bash pour root

- Utilisation du playbook : ansible-playbook copy_bash.yml -e host=home
- copy_bash.yml

```
---
- name: Copy .bashrc, .bash_aliases, .bash_functions for root
hosts: "{{ host }}"
tasks:
  - name: copy the files for root
    copy: src=files/{{ item }} dest=/root/{{ item }} owner=root group=root mode=0644 backup=yes
    with_items:
      - .bashrc
      - .bash_aliases
      - .bash_functions
```

Installer des packages

- Installer des packages (on fait un update suivi d'un upgrade auparavant)
- Utilisation du playbook : ansible-playbook install_packages.yml -e host=all
- install_packages.yml

install_packages.yml

```
---
```

```
- name: Update APT package cache, upgrade then install packages
  hosts: "{{ host }}"
  tasks:
    - name: Update APT package cache and upgrade
      apt: upgrade=yes update_cache=yes cache_valid_time=7200

    - name: Install packages
      apt: name={{ item }} state=present
      with_items:
        - chkconfig
        - cifs-utils
        - clamav
        - clamav-daemon
        - clamav-freshclam
        - cron-apt
        - debian-goodies
        - fail2ban
        - htop
        - logrotate
        - logwatch
        - manpages-fr
        - manpages-fr-dev
        - manpages-fr-extra
        - ssmtp
        - ncdu
        - ntp
        - rkhunter
        - rsync
        - rsyslog
        - sysv-rc-conf
```

Explication

- Ansible va installer tous ces packages
- L'option cache_valid_time=7200 permet d'éviter de mettre à jour le cache si il est plus récent que 2h (7200 secondes).

Mettre à jour le fichier sshd_config

- Mettre à jour le fichier sshd_config, en faire une sauvegarde si modification il y a puis redémarrer le service SSH
- Utilisation du playbook : ansible-playbook ssh.yml -e host=PC-PUPUCE
- Playbook ssh.yml

```
---
- name: Copy sshd_config
  hosts: "{{ host }}"
  tasks:
    - name: copy sshd_config
      copy: src=files/sshd_config dest=/etc/ssh/sshd_config owner=root group=root mode=0644 backup=yes
      notify: restart ssh

  handlers:
    - name: restart ssh
      service: name=ssh state=restarted
```

Explication

- les handlers, une action qui ne sera exécutée que sur notification d'une autre action.
- le fichier sshd_config ne nécessite aucune modification
- il n'avait aucun sauvegarde et le service SSH n'est pas redémarré.
- Si le sshd_config nécessite une modification
- Ansible fait une sauvegarde de l'ancien sshd_config puis redémarre le service SSH.
- notify : restart ssh et name : restart ssh, il faut évidemment que ce soit identique.

Installer Webmin

- Utilisation du playbook : ansible-playbook webmin.yml
- Utilisation du playbook : ansible-playbook ssh.yml -e host=PC-PUPUCE
- webmin.yml

```
---
```

```
- name: Install Webmin
hosts: server
tasks:
  - name: download webmin
    get_url: url=http://www.webmin.com/download/deb/webmin-current.deb dest=/tmp/webmin.deb

  - name: install dependencies
    apt: name={{ item }} state=present update_cache=yes
    with_items:
      - apt-show-versions
      - libauthen-pam-perl

  - name: install webmin
    apt: deb=/tmp/webmin.deb
    notify: delete webmin.deb

handlers:
  - name: delete webmin.deb
    shell: rm /tmp/webmin.deb
```

Plan

- 3 Mise en œuvre de playbooks
- 4 Configuration de playbooks complexes
- 5 Gestion des variables et des inclusions
- 6 Mise en œuvre du contrôle des tâches
- 7 Mise en œuvre de modèles Jinja2
- 8 Mise en œuvre de rôles
- 9 Mise en œuvre d'Ansible Vault

- Le Globbing : c'est matcher des résultats avec des wildcards.

Par exemple

si je me met dans /etc/ansible/ et que je fais un rm *.yml,
c'est du globbing, je matche tous les fichiers se terminant par
.yml et je les supprime

Installer les VMware Tools sur un ESXi

- ansible-playbook vmwaretools.yml -e host=SRV-WEB
- Playbook vmwaretools.yml

vmwaretools.yml

```
---
```

- name: Install VMware Tools
 - hosts: "{{ host }}"
 - tasks:
 - name: Update APT package cache and upgrade`apt: upgrade=yes update_cache=yes cache_valid_time=7200`
 - name: Install packages`apt: name={{ item }} state=present`
 - with_items:
 - autoconf
 - automake
 - binutils
 - make
 - psmisc
 - cpp
 - name: Install linux-headers`shell: aptitude -y install linux-headers-$(uname -r)`
 - name: Untar the archive`unarchive: src={{ item }} dest=/tmp`
 - with_fileglob:
 - files/VMwareTools*.tar.gz
 - name: Launch VMwareTools installation`shell: "{{ item }}"`
 - with_items:
 - /tmp/vmware-tools-distrib/vmware-install.pl --default
 - rm -r /tmp/*

Explication

- module unarchive qui permet de gérer la décompression des archives .tar.gz et zip.
- le Globbing ⇒ L'expression files/VMwareTools*.tar.gz permet de matcher le fichier qui s'appelle VMwareTools-9.0.10-1481436.tar.gz dans mon dossier files.

Explication

Quel est l'intérêt ?

- Ce fichier change souvent de nom car on supprime l'ancienne version de ce fichier puis on met une nouvelle version régulièrement.
- Avec le Globbing, on ne touche pas au playbook
- inutile de l'ouvrir ou de le modifier.

Remarque

l'usage que je fais du globbing est limité

Imaginez que vous souhaitez supprimer tous les fichiers *.log ou *.jpeg d'un dossier

Sauvegarder puis afficher un fichier

- Utilisation du playbook : ansible-playbook backup_cat.yml -e 'host=SRV-TEST file=/etc/hostname'
- Playbook backup_cat.yml

```
---
- name: Backup the file and open
  hosts: "{{ host }}"
  tasks:
    - name: backup the file
      shell: cp -p {{ file }} {{ file }}_{{ ansible_date_time.date }}.bak

    - name: open the file
      shell: /bin/cat {{ file }}
      register: cat
    - debug: var=cat.stdout_lines
```

Explication

Quel est l'intérêt ?

- assons à register, cela sert ici à enregistrer la sortie de /bin/cat /etc/hostname.
- Debug nous sert ici à afficher la sortie de cat.
- On a var=cat.stdout_lines constitué de la variable cat point stdout_lines (la sortie standard)
- les flux standards sont composés de l'entrée standard stdin
- la sortie standard stdout
- l'erreur standard stderr.

Remarque

- Utilité de ce playbook lors de la première configuration du poste.
- Lancer certains fichiers de configuration qui ne nécessitent normalement aucune modification (`/etc/hostname` et `/etc/resolv.conf` sont des bons exemples).
- sauvegarder les fichiers forts importants pour le système au cas où il y aurait une modification au même temps je contrôle qu'ils sont corrects.

Copier le fichier interfaces avec des variables

- Utilisation du playbook : ansible-playbook copy_interfaces.yml -e 'host=SRV-NEW
ipv4=192.168.1.16'
- Playbook copy_interfaces.yml

```
---
- name: Copy /etc/network/interfaces
  hosts: "{{ host }}"
  tasks:
    - name: copy the file
      template: src=templates/interfaces.j2 dest=/etc/network/interfaces owner=root group=root mode=0644
```

Copier le fichier interfaces avec des variables

- Templates interfaces.j2

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet static
    address {{ ipv4 }}
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.255.255
    gateway 192.168.1.1
```

Explication

- Utilisation du module template
- pas de variable host
- la seconde variable nommée ipv4 se trouve dans le fichier interfaces.j2.
- Le fichier interfaces.j2 est le template

Explication

- L'extension .j2 signifie par convention que c'est un fichier Jinja2.
- le playbook va copier le template interfaces.j2 sur le poste Client en remplaçant la variable ipv4 par l'adresse IP 192.168.1.16
- affectation des droits aux fichier
- faire un backup avant de modifier du fichier.

Copier le fichier hosts avec des variables

- Utilisation du playbook : ansible-playbook
copy_hosts.yml -e 'host=SRV-NEW ipv4=192.168.1.16'
- Playbook copy_hosts.yml

```
---
- name: Copy /etc/hosts
  hosts: "{{ host }}"
  tasks:
    - name: copy the file
      template: src=templates/hosts.j2 dest=/etc/hosts owner=root group=root mode=0644 backup=yes
```

Copier le fichier hosts avec des variables

- Template hosts.j2

```
127.0.0.1      localhost
127.0.1.1      {{ host }}
{{ ipv4 }}      {{ host }}.domain.tld      {{ host }}

# The following lines are desirable for IPv6 capable hosts
::1      localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Copier le fichier admin avec des variables

- Copier le fichier admin (/etc/cron.d/admin) avec des variables dedans
- Utilisation du playbook : ansible-playbook
copy_cron.d_admin.yml -e 'host=SRV-NEW minute=00 hour=23'
- Playbook copy_cron.d_admin.yml

```
---
- name: Copy /etc/cron.d/admin
hosts: "{{ host }}"
tasks:
  - name: copy the file
    template: src=templates/admin.j2 dest=/etc/cron.d/admin owner=root group=root mode=0600 backup=yes
```

Copier le fichier admin avec des variables

- Template admin.j2

```
{{ minute }} {{ hour }} * * * caroot /bin/bash /backup.sh
```

Plan

- 3 Mise en œuvre de playbooks
- 4 Configuration de playbooks complexes
- 5 Gestion des variables et des inclusions
- 6 Mise en œuvre du contrôle des tâches
- 7 Mise en œuvre de modèles Jinja2
- 8 Mise en œuvre de rôles
- 9 Mise en œuvre d'Ansible Vault

Les Variables

- Ansible fournit de nombreuses variables sur l'environnement serveur.
- Le module **setup** nous renseigne sur ces variables :

Les variables

```
ansible buzut -m setup
buzut | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "37.59.21.45"
        ],
        "ansible_all_ipv6_addresses": [
            "2001:41d0:8:852d::",
            "fe80::225:90ff:fe7c:fa36"
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "01/03/2014",
        "ansible_bios_version": "3.0a",
        [...]
    },
    "ansible_lsb": {
        "codename": "trusty",
        "description": "Ubuntu 14.04.4 LTS",
        "id": "Ubuntu",
        "major_release": "14",
        "release": "14.04"
    },
    [...]
}
```

Les Variables

- Les variables sont utilisées dans les playbooks.
- Pouvoir accéder à une variable de la manière suivante :

```
# variable simple  
{{ ansible_architecture }}  
  
# pour accéder à une propriété  
{{ ansible_lsb.major_release }}  
  
# pour accéder à un tableau (première propriété)  
{{ ansible_all_ipv4_addresses[0] }}
```

Les variables

- vhosts contient maintenant la liste des éléments présents dans `/etc/apache2/sites-enabled/` tel que l'affiche la commande `ls`.
- Il y a un type de variable un peu particulier
- ⇒ pouvoir s'avérer très utile : ce sont les variables prompt.
- Au moment de l'exécution ⇒ la valeur à attribuer est demander à l'utilisateur

Les variables

Par exemple

- Dans le cas d'un playbook permettant d'installer un nouveau serveur,
- Il suffit donc de procéder comme ceci :

```
---
- hosts: "{{ servernames }}"
  vars_prompt:
    - name: "servernames"
      prompt: "Which hosts would you like to setup?"
      private: no
  tasks:
    [...]
```

- ansible considère le prompt comme password-sensitive
- ansible n'affiche pas les caractères.

Les variables

- Possible de passer des variables au playbook directement depuis la CLI
- ⇒ au moment de l'invocation .
- Reprendre le même exemple que ci-dessus en enlevant le vars_prompt :

```
---
```

```
- hosts: "{{ servernames }}"
  tasks:
    [...]
```

Les variables

- Il suffit simplement d'appeler le playbook :
ansible-playbook baseServer.yml –extra-vars
"servernames=db6".
- les variables prennent des guillemets lorsqu'elles débutent une ligne.
- illustration :

```
# ici la variable doit être entourée de guillemets
- hosts: "{{ servernames }}"
  tasks:
    [...]

# mais pas dans ce cas là
- name: Enable VirtualHost
  file:
    src: /etc/nginx/sites-available/{{ domain }}
    dest: /etc/nginx/sites-enabled/{{ domain }}
    state: link
```

Les variables

- Debug, le var_dump d'Ansible
- Récupérer des variables → l'objet de debug.
- Reprenons l'exemple précédent :

```
---
- hosts: buzut
  tasks:
    - name: determine vhosts
      command: /bin/ls /etc/apache2/sites-enabled/
      register: vhosts
    - debug: msg="{{ vhosts }}"
```

L'exécution

```
---
# on lance le playbook
ansible-playbook test.yml

PLAY *****
TASK [setup] *****
ok: [buzut]

TASK [determine vhosts] *****
changed: [buzut]

TASK [debug] *****
```

Suite d'exécution

```
ok: [buzut] => {
    "msg": {
        "changed": true,
        "cmd": [
            "/bin/ls",
            "/etc/apache2/sites-enabled"
        ],
        "delta": "0:00:00.014652",
        "end": "2016-03-24 23:21:32.612755",
        "rc": 0,
        "start": "2016-03-24 23:21:32.598103",
        "stderr": "",
        "stdout": "buzut.conf\\default.conf",
        "stdout_lines": [
            "buzut.conf",
            "default.conf"
        ],
        "warnings": []
    }
}

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

Les inclusion

- L'include dans Ansible ⇒ exactement comme le include de PHP.
- permet de scinder des tâches en différents fichiers et de les importer au besoin dans vos playbooks.
- Imaginez une tâche qui se charge d'installer un dæmon de monitoring
- voudrez qu'elle s'exécute sur le serveurs de base de données
- **Sans l'include** ⇒ devoir répéter dans tous le playbooks
- une simple référence suffit.

Les inclusion

- Définir les tâches dans un fichier dédié ⇒ `setupMonitoring.yml`.

```
---
```

```
- name: Install monitoring agent
  apt:
    name: blabla
```

```
# on ajoute toutes les tâches que l'on veut
```

Les inclusion

- ce fichier sera intégré directement dans un playbook
- Pour référencer les tasks : plaçant directement la liste de tâches.
- Une fois ce fichier créé et enregistré
- admettons pour l'exemple que nous enregistriions toutes nos tâches dans tasks et que les playbooks soient à la racine
- voilà à quoi ressemblerait le playbook.

Les inclusion

```
---
```

```
- hosts: dbservers
  vars:
    email: mon@email.fr

  vars_prompt:
    - name: "dbrootpasswd"
      prompt: "Database root password"

  tasks:
    - include: tasks/commonSetup.yml
    - include: tasks/setupMonitoring.yml
    - include: tasks/installMySQL.yml

    # on peut bien entendu mélanger des includes et des tâches classiques
    - name: Install htop
      apt:
        name: htop
```

Les inclusion

- L'endroit où l'include est utilisé
 - Pour signifie une inclusion de tache
 - Pour signifie une inclusion de playbook
- Dans l'exemple précédent → l'insertion est après tasks :
- inclusion que des tâches.
- Si l'insertion est au premier niveau du playbook
- inclusion que de playbook

Plan

- 3 Mise en œuvre de playbooks
- 4 Configuration de playbooks complexes
- 5 Gestion des variables et des inclusions
- 6 Mise en œuvre du contrôle des tâches
- 7 Mise en œuvre de modèles Jinja2
- 8 Mise en œuvre de rôles
- 9 Mise en œuvre d'Ansible Vault

Liste de tâches

- Les tâches sont exécutées dans l'ordre
- une à la fois
- sur toutes les machines associées au d'hôte pattern.
- avant de passer à la tâche suivante.
- dans un Play, tous les hôtes vont recevoir les mêmes directives de tâches.
- Le but d'une play est de mapper une sélection d'hôtes sur des tâches.

Liste de tâches

- Lors de l'exécution Playbook \implies s'exécute de haut en bas
- les hôtes dont les tâches ont échoué sont supprimés de la rotation de l'ensemble du Playbooks.
- En cas d'échec, corrigez simplement le fichier de playbooks et réexécuter
- Le but de chaque tâche est d'exécuter un module avec des arguments très spécifiques.
- Pouvoir utiliser des variables dans les arguments de modules.

Liste de tâches

- une module peut être idempotent , c-a-d exécuter un module plusieurs fois dans une séquence → avoir le même résultat que de l'exécuter une seule fois.
- Pour réaliser l'idempotency → demander à un module de vérifier
 - si l'état final souhaité a déjà été atteint
 - si cet état est atteint
 - quitter sans effectuer aucune action.
- Si tous les modules utilisés par un Playbook sont idempotents → le playbooks est idempotent.

Listes des tâches

- Chaque tâche a un name,
- le résultat de tâche est inclue dans de l'exécution du playbook.
- le résultat lisible par l'homme.
- fournir une bonne description de chaque étape de la tâche.
- Si le nom n'est pas fourni
- la chaîne alimentée en 'action' sera utilisée pour la sortie.

Liste des tâches

- Voici à quoi ressemble une tâche élémentaire.
- le module de service prend des key=value en arguments :

```
tasks:  
  - name: make sure apache is running  
    service:  
      name: httpd  
      state: started
```

Liste des tâches

- les modules de commande et de shell sont les seuls à prendre une liste d'arguments sans utiliser le key=value .

```
tasks:  
  - name: enable selinux  
    command: /sbin/setenforce 1
```

Liste des tâches

- Les modules de commande et shell se soucient des codes de retour.
- Une commande dont le code de sortie réussi n'est pas nul
- → faire de cette façon

```
tasks:  
  - name: run this command and ignore the result  
    shell: /usr/bin/somecommand || /bin/true
```

- ou ça

```
tasks:  
  - name: run this command and ignore the result  
    shell: /usr/bin/somecommand  
    ignore_errors: True
```

Liste des tâches

- Si la ligne d'action prend trop de temps
- faire un break et mettre en retrait les lignes de continuation :

```
tasks:  
  - name: Copy ansible inventory file to client  
    copy: src=/etc/ansible/hosts dest=/etc/ansible/hosts  
          owner=root group=root mode=0644
```

```
tasks:  
  - name: create a virtual host file for {{ vhost }}  
    template:  
      src: somefile.j2  
      dest: /etc/httpd/conf.d/{{ vhost }}
```

Inclure une liste de lecture ou de tâches

- Pouvoir inclure un fichier avec une liste de jeux ou de tâches à exécuter dans le playbook actuel.
- Les fichiers avec une liste de lectures ne peuvent être inclus qu'au niveau supérieur.
- Les listes de tâches ne peuvent être incluses que là où les tâches sont normalement exécutées (en cours).

Inclure une liste de lecture ou de tâches

- Avant Ansible version 2.0, tous les inclus étaient «statiques» et étaient exécutés lors de la compilation du chaîne d'action.
- Les inclus statiques ne sont pas soumis à la plupart des directives.
- exemple, des boucles ou des conditions sont appliquées à chaque tâche héritée.

Inclure une liste de lecture ou de tâches

- Depuis Ansible 2.0, les tâches incluses sont dynamiques et se comportent davantage comme des tâches réelles.
- signifie qu'ils peuvent être mis en boucle
- ignorés et utiliser des variables de n'importe quelle source.
- Ansible essaie de le détecter automatiquement,
- utiliser la directive static (Ansible 2.1) pour contourner la détection automatique.

Inclure une liste de lecture ou de tâches

- c'est une fonctionnalité essentielle d'Ansible(plutôt qu'un module)
- Include a un des comportements clairs
- Include a un comportements non intuitifs
- qu'il fonctionne dans un contexte
 - statique
 - dynamique
 - liste d'action
 - un playbooks
- afin de clarifier les comportements → Utilisant les modules

- include_tasks
- include_role
- import_playbook
- import_tasks

Inclure une liste de lecture ou de tâches

```
- hosts: localhost
  tasks:
    - debug:
        msg: play1

- name: Include a play after another play
  include: otherplays.yaml

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

    - name: Include task list in play
      include: stuff.yaml

    - debug:
        msg: task10

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

    - name: Include task list in play only if the condition is true
      include: "{{ hostvar }}.yaml"
      static: no
      when: hostvar == "truehost"
```

Plan

- 3 Mise en œuvre de playbooks
- 4 Configuration de playbooks complexes
- 5 Gestion des variables et des inclusions
- 6 Mise en œuvre du contrôle des tâches
- 7 Mise en œuvre de modèles Jinja2
 - Introduction
 - Quelles sont les templates
 - Installation du Jinja2
 - Structure de Contrôle de templates

JINJA 2

- un moteur de gabarit Python
- utilisé pour créer des formats de balisage HTML, XML ...
- renvoyés à l'utilisateur via une réponse HTTP.



Introduction JINJA 2

- elle est conçue pour être flexible, rapide et sécurisée.
- Être familiarisé avec d'autres langues basées sur le texte, telles que Smarty ou Django ⇒ sentir bien avec Jinja2
- Il est à la fois convivial pour les concepteurs et les développeurs de s'en tenir aux principes de Python
- Ajouter des fonctionnalités utiles pour les environnements de modèles



Pourquoi Jinja2 est utile ?

- Sa syntaxe de balise de modèle est cohérente
- Open source
- Peut être utilisé par d'autres bibliothèques de code.



Les avantages de jinja2

- **Sandboxed Execution** : fournit un cadre protégé pour l'automatisation des programmes de test dont le comportement est inconnu et doit être analysé

Les avantages de jinja2

- **HTML Escaping** : Jinja 2 est doté d'un puissant échapement HTML automatique
- permet d'empêcher les scripts inter-site (XSS Attack).
- Il existe des caractères spéciaux ⇒ ont une signification particulière dans les templates.

Quelles sont les templates

- À l'époque, les serveurs avaient une collection de fichiers
- tels que des fichiers HTML → envoyés à la demande des clients.
- C'étaient des données statiques envoyées.

Quelles sont les templates

- Dans le monde Web moderne ⇒ moins de données statiques
- Des données dynamique survenue des client aux serveurs
- Le Web dépend des demandes de clients
- la création de templates Jinja2 est utilisée.

Quelles sont les templates

- Un modèle contient des variables
- remplacées par les valeurs
- transmises aux templates .
- Les variables sont utiles avec les données dynamiques.

Installation du jinja2

```
easy_install Jinja2  
pip install Jinja2
```

Utilisation de base de l'API

- Le moyen le plus simple de créer un template
- le système de fichiers est chargé par des template :

```
>>> from jinja2 import Template
>>> template = Template('Hello {{ name }}!')
>>> template.render(name='John Doe')
u'Hello John Doe!'
```

Que font les moteur templates

- Utilisation de moteurs templates dans les applications Web
- exemple :

```
 {{name}} avait un petit {{animal}}.
```

- Si nom = 'marie' et animal = 'agneau'
- résultat :

```
Mary avait un petit agneau.
```

- Utilisé pour la génération de texte

Que font les moteur templates

- Un exemple HTML

```
<html>
<head>
<title> {{title}} </ title>
</ head
<body>
<h1> {{header}} </ h1>
<p> {{body}} </ p>
</ corps>
</ html>
```

Changement Du templates

- Il existe plusieurs façons de charger des modèles.
- utilisant le FileSystemLoader .
- Cette déclaration d'importation \Rightarrow est implicite dans tous les exemples de code.

```
depuis l'environnement d'importation jinja2, FileSystemLoader
```

Changement Du templates

- Pour passer à la répertoire contenant le templates au FileSystemLoader

```
file_loader = FileSystemLoader ('templates')
```

- chargement d'environnement

```
env = Environment (loader = file_loader)
```

Création du premier templates

- le premier modèle à construire est à l'intérieur du répertoire de templates.
- Appeler hello_world.txt
- écrivez dans hello_world.txt le texte suivant :

```
Bonjour le monde!
```

- chargement de templates

```
template = env.get_template ('hello_world.txt')
```

Création du premier templates

- pouvoir rendre et imprimer.

```
output = template.render ()  
print (sortie)
```

- Ce qui va être imprimer

```
Bonjour le monde!
```

Ajouter une variable

- créer un nouveau modèle appelé lamb.txt contenant

```
 {{name}} avait un petit agneau.
```

- Le { {nom} } est une variable de templates.
- Charger le nouveau templates .

```
template = env.get_template ('lamb.txt')
```

Ajouter un variable

- à l'appel passer la variable en argument

```
output = template.render (name = 'Mary')
print (sortie)
```

- Sortie :

```
Mary avait un petit agneau
```

Changer une le contenue du variable(nom)

- à l'appel passer la variable en argument

```
output = template.render (name = 'Jason')
print (sortie)
```

- Sortie :

```
Jason avait un petit agneau
```

Ajouter deux ou plusieurs variables

- ajoutons une autre variable

```
 {{name}} avait un petit {{animal}}.
```

- Passer plusieurs variables en arguments

```
output = template.render (name = 'Bob', animal = 'cat')  
print (sortie)
```

- Sortie :

```
Bod avait un petit chat.
```

Utiliser des Objets

- accéder à des Objets

```
 {{data.name}} avait un petit {{data.animal}}.
```

- Passer un dictionnaire

```
person = {}
person ['name'] = 'Frank'
person ['animal'] = 'dog'

sortie = template.render (data = person)
print (sortie)
```

Utiliser des Objets

- Sortie :

```
Frank avait un petit chien.
```

Les Conditions

- créer un modèle appelé vérité.txt

```
{% if truth%}  
Ceci est vrai  
{% else%}  
Ceci est faux  
{% endif%}
```

- passer un variable TRUE en argument

```
template = env.get_template ('truth.txt')  
output = template.render (vérité = True)  
print (sortie)
```

- Sortie :

C'est vrai

Les Conditions

- Remplacer la variable par FALSE

```
template = env.get_template ('truth.txt')
output = template.render (vérité = False)
print (sortie)
```

Les Conditions

- Sortie :

```
C'est faux
```

Boucle For

- créer n modèle appelé rainbow.txt

```
{% pour la couleur en couleurs%}
    {{color}}
{% endfor%}
```

- passer une liste de couleur

```
template = env.get_template ('rainbow.txt')

colors = ['rouge', 'green', 'blue']

output = template.render (colors = colors)
print (output)
```

Boucle For

- Sortie :

```
rouge
```

```
vert
```

```
bleu
```

Héritage dans un template

- Combiner les block
- nous utiliserons un HTML comme exemple.
- créer un template appelé, header.html

```
<HEAD>
  <TITLE> {{titre}} </TITLE>
</HEAD>
```

- créer un autre templates intitulé base.html Inclure le template header.html

```
<HTML>
  {% include 'header.html'%}
  <BODY>
  </BODY>
</HTML>
```

Héritage dans un template

- Maintenant rendre :

```
template = env.get_template ('base.html')

output = template.render (title = 'Titre de la page')
print (output)
```

- Sortie

```
<HTML>
  <HEAD>
    <TITLE> Titre de la page </ TITLE>
  </ HEAD>
<BODY>
</ BODY>
</ HTML>
```

Héritage dans un template

- permettre aux template enfants d'utiliser le fichier base.html .

```
<HTML>
    {% include 'header.html'%}
<BODY>
    {% contenu du bloc%} {% endblock%}
</ BODY>
</ HTML>
```

- créer un modèle appelé child.html qui étend le modèle base.html

```
{% extend "base.html"%}
```

```
{% le contenu du bloc%}
```

```
<p>
```

```
 {{body}}
```

```
</ p>
```

```
{% endblock%}
```

Héritage dans un template

- Rendre le nouveau templates creer

```
template = env.get_template ('child.html')

output = template.render (title = 'Titre de la page', body =
'Stuff'))
print (output)
```

- Sortie

```
<HTML>
<HEAD>
    <TITLE> Titre de la page </ TITLE>
</ HEAD>
<BODY>
    <p>
        Contenu
    </ p>
</ BODY>
```

Plan

- 3 Mise en œuvre de playbooks
- 4 Configuration de playbooks complexes
- 5 Gestion des variables et des inclusions
- 6 Mise en œuvre du contrôle des tâches
- 7 Mise en œuvre de modèles Jinja2
- 8 Mise en œuvre de rôles
- 9 Mise en œuvre d'Ansible Vault

Introduction à la gestion de rôle

- Les outils de gestion de configuration → premier réflexe est de construire des rôles complexes (fait tous).
- L'objectif → construire des rôles simples et efficaces
- pouvoir combiner et faire évoluer facilement.
- combinant plusieurs rôles par inclusion → réaliser des tâches complexes.

Des rôles simples

- créer des rôles → qui se limitent à quelques tâches
- En précisant un objectif unique.
- un rôle est une succession de `when` : ou de `with_items` :
 - contient 42 tâches, il est assurément trop complexe.
- Un exemple de rôle simple → création d'un compte utilisateur.

Des rôles simples

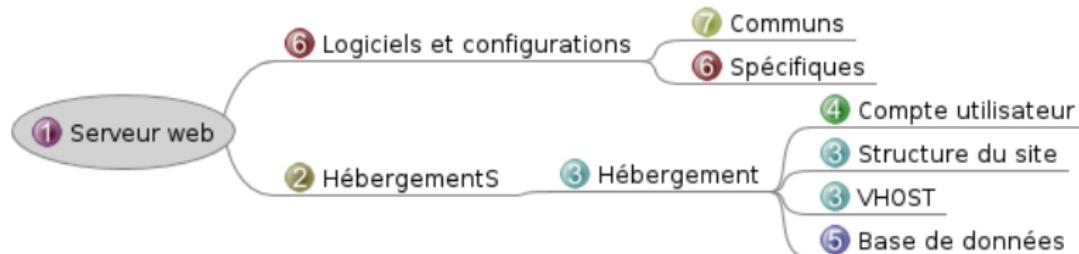
```
1 ---  
2 # tasks file for roles/x_user_add  
3  
4 ##  
5 # création du groupe principal si aucun spécifié  
6 - name: x - Creation du groupe principal pour "{{ x_user.uname }}"  
7   group:  
8     name="{{ x_user.uname }}"  
9     gid={{ x_user.uid | default(omit) }}  
10    when: x_user.gid is undefined  
11    become: yes  
12  
13 ##  
14 # création de l'utilisateur  
15 - name: x - Création de l'utilisateur "{{ x_user.uname }}"  
16   user:  
17     name="{{ x_user.uname }}"  
18     uid="{{ x_user.uid | default(omit) }}"  
19     password="{{ x_user.hash | default("*") }}"  
20     createhome=yes  
21     state=present  
22     shell="{{ x_user.shell | default('/bin/bash') }}"  
23     group="{{ x_user.gid | default(x_user.uname) }}"  
24     groups="{{ x_user.groups | default(omit) }}"  
25   become: yes
```

Des rôles simples

- Création un compte utilisateur → l'utilisation des paramètres obligatoires et d'autres optionnels,
- "`var | default(omit)`" : permet de se passer de condition et de simplifier énormément les rôles.
- la variable `x_user` n'est pas définie → le rôle inclus avec un autre

Des rôles combinables et imbriquables

- combinant et imbriquant des rôles → parviendrons à réaliser des ensembles complexes d'actions.
- grâce à la fonction « include » d'Ansible



Des rôles combinables et imbriquables

- vars : permet de transmettre des données au fichier inclus.

```
1 ---
2 # tasks file for roles/x_web_add_site
3
4 [...]
5 ##
6 # appel x_user_add et création de l'utilisateur
7 - include: "{{ roles_path }}x_user_add/tasks/main.yml"
8   vars:
9     x_user: "{{ x_web }}"
10 [...]
```

Des rôles qui prennent des arguments

```
1 ---  
2 - name: Playbook pour créer un hébergement  
3   hosts: web  
4  
5   roles:  
6     - { role: x_web_add_site, selected_web: "ancel1" }
```

- Le fichier de tâche du rôle x_web_add_site
- ⇒ équivalent au rôle n 3 « hébergement » de mon schéma plus haut, qui permet de créer et activer un hébergement

Des rôles qui prennent des arguments

```
1 ---
2 # tasks file for roles/x_web_add_site
3
4 ##
5 # set_fact pour propager la variable aux fichiers inclus
6 - set_fact:
7     cur_web: "{{ selected_web }}"
8
9 ##
10 # inclusion du fichier de variables, pour définir x_web en fonction de cur_web
11 - include_vars: "{{ roles_path }}x_web_add_site/vars/main.yml"
12
13 ##
14 # appel x_user_add et création de l'utilisateur
15 - include: "{{ roles_path }}x_user_add/main.yml"
16   vars:
17     x_user: "{{ x_web }}"
18
19 ##
20 # création de la structure du site
21 - name: x_web - ajout des fichiers de base pour "{{ x_web.uname }}"
22   template:
23     src="{{ item }}"
24     dest="/home/{{ x_web.uname }}/www/"
25     owner="{{ x_web.uname }}"
26     group="{{ x_web.gid }}"
27     force=no
28   with_fileglob:
29     - "{{ roles_path }}/x_web_add_site/templates/bundle/*"
```

Des rôles qui prennent des arguments

```
30 ##  
31 # création du vhost dans /etc/apache2/site-available  
32 - name: x_web - création du vhost "{{ x_web.uname }}"  
33   template:  
34     dest="/etc/apache2/sites-available/{{ x_web.uname }}"  
35     src=vhost.conf  
36     force=no  
37   become: yes  
38  
39 ##  
40 # Activation du vhost  
41 - name: x_web - activation du vhost "{{ x_web.uname }}"  
42   command: a2ensite {{ x_web.uname }}  
43   args:  
44     creates: /etc/apache2/sites-enabled/{{ x_web.uname }}  
45   become: yes  
46   notify:  
47     - restart apache
```

Mon fichier de variable :

```
1 ---  
2 # vars file for roles/x_web_add_site  
3  
4 x_web: "{{ web_enabled[cur_web] }}"
```

Des rôles qui prennent des arguments

- l'équivalent de l'inventaire JSON serait du genre :

```
1 "web_enabled":{  
2     "ancel1":{  
3         "uid":"xxx",  
4         "gid":"yyy",  
5         "shell":"/bin/false",  
6         "uname":"ancel1",  
7         "ServerName":"ancel1.fr",  
8         "ServerAliases": "",  
9         "tld":"fr",  
10        "cms":"Wordpress",  
11        "php":"54",  
12        "vhost":"1",  
13        "hash":"zzz"  
14    },  
15    "exemple2":{  
16        "uid":"xxx",  
17        "gid":"yyy",  
18        "shell":"/bin/false",  
19        "uname":"exemple2",  
20        "ServerName":"exemple.fr",  
21        "ServerAliases": "*.exemple.fr exemple.com",  
22        "tld":"fr",  
23        "cms":"Wordpress",  
24        "php":"54",  
25        "vhost":"1",  
26        "hash":"zzz"  
27    }  
28}  
29}
```

Des rôles qui prennent des arguments

- l'équivalent de l'inventaire JSON serait du genre :

```
1 # tasks file for roles/x_web_provision
2
3 - include: "{{ roles_path }}x_web_add_site/tasks/main.yml"
4   vars:
5     selected_web: "{{ item }}"
6     with_items: "{{ web_enabled }}"
7 [...]
```

Des rôles qui prennent des arguments

- Le rôle est intitulé `x_web_add_site`
- l'appelle du role seras depuis le playbook
- En utilisant la variable `selected_web="ancel1"`
- `set_fact` : permet de définir la variable `cur_web` , en associant la même valeur que `selected_web` (« `ancel1` »).
- le fichier de variables définit une variable `x_web`
- → prend la valeur de tableau `web_enabled` (qui lui contient tous vhosts)
- les données qui correspondent à `selected_web` .
- `x_user` prendra la valeur de `x_web`
- inclurons le rôle `x_user_add`
- Ajoutant les données nécessaires pour créer l'utilisateur

Des rôles qui prennent des arguments

- Les variables ont un scope limité au rôle
- Ne sont pas transmises au niveau dessous.
- Il est nécessaire d'utiliser **vars** : lors de l'include pour pouvoir les propager.

Des rôles qui prennent des arguments

- choisir quel vhost déployer avec selected_web .
- surcharger selected_web avec les extra_vars :
ansible-playbook -i inventory.php playbooks/hebergement.yml –check –diff -e selected_web="exemple2"
- Étudier un fichier pour déployer un hébergement précis
- créer autant de lignes dans le playbook qu'il y a de vhosts à créer.
- Où inclure le rôle dans un autre rôle avec une boucle.

Des rôles « inception »

- le rôle x_web_add_site est un rôle unitaire et simple.
- quand le serveur est tout neuf, il faut pouvoir tous les créer d'un seul coup.
- Devrons créer un nouveau rôle x_web_provision , équivalent du rôle n 2 du schéma,
- qui boucle l'include du rôle x_web_add_site :

Des rôles « inception »

- modifier le playbook pour appeler le nouveau rôle :

```
1 ---  
2 - name: Playbook pour créer TOUS les hébergements  
3   hosts: web  
4  
5   roles:  
6     - { role: x_web_provision }
```

Plan

- 3 Mise en œuvre de playbooks
- 4 Configuration de playbooks complexes
- 5 Gestion des variables et des inclusions
- 6 Mise en œuvre du contrôle des tâches
- 7 Mise en œuvre de modèles Jinja2
- 8 Mise en œuvre de rôles
- 9 Mise en œuvre d'Ansible Vault

Introduction Ansible Vault

- Ansible Vault est une fonctionnalité
- Permettant aux utilisateurs de chiffrer des valeurs et des structures de données
- Cela permet de sécuriser toutes les données sensibles
- Il ne doit pas être visible publiquement⇒ comme les mots de passe ou les clés privées.
- Ansible déchiffre automatiquement le contenu crypté ⇒ par le coffre-fort ⇒ à l'exécution et la disponibilité de clé.
- l'aide de la commande ansible-vault

Créer des fichiers encryptés

- Editez le fichier et rajoutez par exemple la ligne suivante :

```
ansible-vault create example.yml
```

- Le terminal demandera de saisir un mot de passe
 - utilisé pour déchiffrer le fichier par la suite
 - permettre de lire son contenu.
- Une fois le mot de passe saisi → ouvrit le fichier avec un éditeur de texte tel que vi ou vim.

Créer des fichiers encryptés

- créer un fichier encrypté par la commande ansible-vault avec l'option create :

```
hello world!
```

- Le terminal demandera de saisir un mot de passe
- Ensuite, ECHAP + : + x et appuyer sur Entrée pour que vos changements pris en compte.

Créer des fichiers encryptés

- Afficher le contenu du fichier à l'aide de la commande cat

```
$ cat example.yml
$ANSIBLE_VAULT;1.1;AES256
6330326165343262353139646165376565356664313061383764393934333
6635356162643038633331633936336234613935626631300a64373135353
3639343662393335663038393039396136303038306635653364346235306
3835353133303838310a65303361363833313334363834386438343734386
6333
```

Voir le contenu de fichiers encryptés

- afficher le contenu \Rightarrow l'option view

```
ansible-vault view example.yml
```

Editer un fichier encrypté

- Éditer un fichier encrypté \Rightarrow l'option edit

```
ansible-vault edit example.yml
```

Editer un fichier encrypté

- Si aucun changement est effectuer sur le contenu
- considéré comme un fichier différent par Ansible.
- le task sera en état CHANGED .

Changer le mot de passe associé à un fichier encrypté

- modifier le mot de passe \implies l'option rekey

```
ansible-vault rekey example.yml
```

- demande d'abord le mot de passe existant
- deux reprises, le nouveau mot de passe que vous voulez mettre en place.

Encrypter un fichier existant

- Encrypter un fichier \Rightarrow l'option encrypt
- fichier toto.yml contient le textes suivants

```
the world is mine
```

- Encrypter avec la commande suivante :

```
ansible-vault encrypt toto.yml
```

Encrypter un fichier existant

- Un mot de passe est demander
- Affichage d'un messages indiquant que l'encryption a bien été effectué :

```
Encryption successful
```

Décrypter un fichier

- Décrypter ce fichier toto.yml \implies l'option decrypt

```
ansible-vault decrypt toto.yml
```

- Une fois le mot de passe saisi,
- Un message est afficher :

```
Decryption successful
```

Décrypter un fichier

- faire cat toto.yml → affichage du contenu du fichier.

```
ansible-vault decrypt toto.yml
```

- Une fois le mot de passe saisi,
- Un message est afficher :

```
Decryption successful
```

Encrypter une variable

- encrypter une variable en ligne de commande

```
ansible-vault encrypt_string password123
```

- suppose que le mot de passe est password123.
- Ansible demander de saisir un mot de passe pour encrypter
- Le mot de passe encrypté s'affichera sous la forme suivante :

```
!vault |
$ANSIBLE_VAULT;1.1;AES256
613065613931346232363232303831373363646431373136326236323335
3664376535633365316662613464663365303930316563380a3132323964
31316431356361376135346336383336166363962656165393936383262
3834373437333336390a623963353939360366462326234643464663062
3165
```

Encrypter une variable

- Utiliser une variable à l'intérieur d'un playbook.
- pouvoir afficher le mot de passe encrypté.
- le playbook est comme suite :

Encrypter une variable

```
---  
  
- name: how to use vault  
  hosts: all  
  vars:  
    my_password: !vault |  
$ANSIBLE_VAULT;1.1;AES256  
613065613931346232363232303831373363646431373136326236323335  
3664376535633365316662613464663365303930316563380a3132323964  
313164313563613761353463363833336166363962656165393936383262  
3834373437333336390a6239633539393630366462326234643464663062  
3165  
  
  tasks:  
  
    - name: show message  
      debug:  
        msg: "My password is {{ my_password }}"
```

Encrypter une variable

- Appliquer la commande suivantes :
- ajouter l'option **--ask-vault-pass**

```
ansible-playbook -i hosts deploy.yml --ask-vault-pass
```

Résultat de playbooks

```
PLAY [how to use vault] ****
TASK [Gathering Facts] ****
ok: [SERVER_A]

TASK [show message] ****
ok: [SERVER_A] => {
    "msg": "My password is password123"
}

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

Plan

10 Résolution des problèmes liés à Ansible

11 Mise en œuvre d'Ansible Tower

Plan

10 Résolution des problèmes liés à Ansible

11 Mise en œuvre d'Ansible Tower

- Exemple d'utilisation Ansible Tower

Ansible Tower

- Ansible Tower une interface Web utilisateur facile à utiliser pour Ansible.
- Il contient les fonctions ansible les plus importantes.
- Tower centralise l'infrastructure avec contrôle d'accès basé sur les rôles
- la planification des tâches et la gestion des stocks graphique.
- L'API et le CLI REST de Tower font qu'il est facile à intégrer dans les outils existants.
- Tower est un produit payant, qui nécessite une licence.
- Une version d'essai gratuite est mise à disposition pour tester le produit.

Ansible Tower

TOWER Projects Inventories Job Templates Jobs admin

87 Hosts	0 Failed Hosts	2 Inventories	0 Inventory Sync Failures	3 Projects	0 Projects Sync Failures
----------	----------------	---------------	---------------------------	------------	--------------------------

Job Status

Period: Past Month • job Type: All • Successful Failed

Date	Successful Jobs	Failed Jobs
05/29	2	0
05/30	1	0
05/31	16	0
06/01	10	2
06/02	12	2
06/03	0	0
06/04	15	0
06/05	0	0
06/06	0	0
06/07	0	0
06/08	0	0
06/09	0	0
06/10	0	0
06/11	0	0
06/12	0	0
06/13	0	0
06/14	0	0
06/15	0	0
06/16	0	0
06/17	0	0
06/18	1	0
06/19	2	0
06/20	0	0
06/21	0	0
06/22	0	0
06/23	0	0
06/24	0	0
06/25	0	0
06/26	0	0
06/27	0	0
06/28	1	0
06/29	7	0

Host Status

Period: Past Month • job Type: All • Successful Failed

Date	Successful Hosts	Failed Hosts
05/29	2	0
05/30	1	0
05/31	16	0
06/01	10	2
06/02	12	2
06/03	0	0
06/04	15	0
06/05	0	0
06/06	0	0
06/07	0	0
06/08	0	0
06/09	0	0
06/10	0	0
06/11	0	0
06/12	0	0
06/13	0	0
06/14	0	0
06/15	0	0
06/16	0	0
06/17	0	0
06/18	1	0
06/19	2	0
06/20	0	0
06/21	0	0
06/22	0	0
06/23	0	0
06/24	0	0
06/25	0	0
06/26	0	0
06/27	0	0
06/28	1	0
06/29	7	0

Recently Used Job Templates

- Update systems
- deploy cloud software
- Fix shellshock
- Boot the cloud
- basic scan

[See all job templates](#)

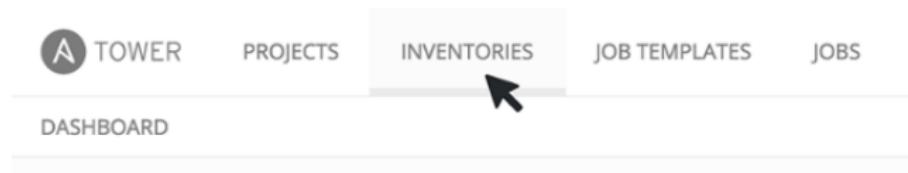
Recent Job Runs

Update systems	a minute ago
Update systems	5 minutes ago
deploy cloud software	16 minutes ago
Fix shellshock	21 minutes ago
Boot the cloud	23 minutes ago

[See all job runs](#)

CRÉER UN INVENTAIRE

- Cliquez sur INVENTAIRES dans le menu du haut.



CRÉER UN INVENTAIRE

- Une fois sur la page Inventories, cliquez sur le bouton vert + **AJOUTER** pour ajouter un inventaire.

CRÉER UN INVENTAIRE

- Nommez l'inventaire et cliquez sur le bouton vert SAVE .

NEW INVENTORY

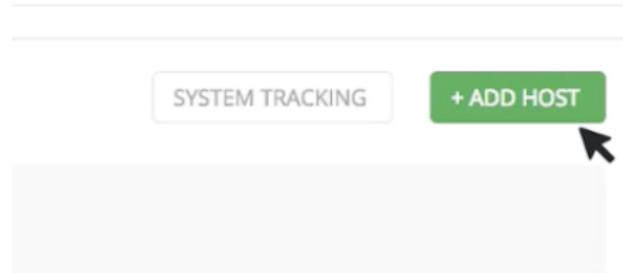
DETAILS PERMISSIONS

*NAME

The screenshot shows a 'New Inventory' creation interface. It has two tabs at the top: 'DETAILS' (which is selected and highlighted in dark grey) and 'PERMISSIONS' (which is white). Below the tabs is a field labeled '*NAME' with a red asterisk indicating it is required. A large, empty input field is provided for entering the name. At the bottom right of the interface are several small navigation icons.

CRÉER UN INVENTAIRE

- Sur la page suivante, cliquez sur le bouton vert + **AJOUTER UN HÔTE** pour ajouter un nouvel hôte.



CRÉER UN INVENTAIRE

- Nommez l'hôte, ajoutez VARIABLES et cliquez sur le bouton vert SAVE .

CREATE HOST ON

*HOST NAME ?

VARIABLES ? YAML JSON

```
1 ---
2 ansible_connection: local|
```

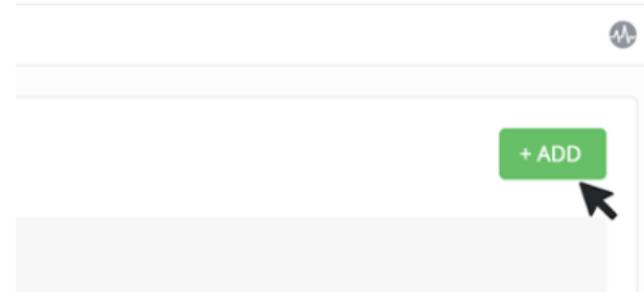
CRÉER UN PROJET

- le Playbook provient d'un dépôt sur GitHub.
- Cliquez sur PROJETS dans le menu du haut.



CRÉER UN PROJET

- Une fois sur la page Projets, cliquez sur le bouton vert + AJOUTER pour ajouter un projet.



CRÉER UN PROJET

- le Playbook provient d'un dépôt sur GitHub.
- Nommez le projet, sélectionnez **SCM TYPE**, ajoutez **SOURCE DETAILS** et cliquez sur le bouton vert **SAVE**.

*NAME

Demo Project

*SCM TYPE

Git



SOURCE DETAILS

*SCM URL

<https://github.com/ansible/ansible-tower-samples>

SCM UPDATE OPTIONS

- Clean
- Delete on Update
- Update on Launch

AJOUTER UN CREDENTIAL

- une information d'identification vierge de la machine est ajoutée. Un nom d'utilisateur, un mot de passe et une clé SSH peuvent tout aussi bien être ajoutés.
- Cliquez sur l'icône représentant une roue dentée dans le menu supérieur, puis sélectionnez CRÉDIDENTIELS .

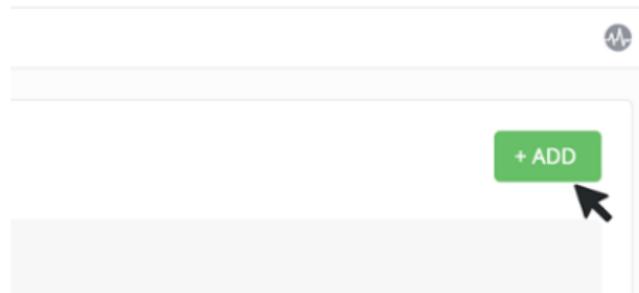
The screenshot shows the Ansible Tower web interface. At the top, there is a navigation bar with icons for user (admin), settings, inventories, jobs, and power. A cursor points to the settings icon. Below the navigation bar, the main content area has a title 'CREDENTIALS' and a descriptive text: 'Add passwords, SSH keys, etc. for Tower to use when launching jobs against machines, or when syncing inventories or projects.' At the bottom right of the content area, there are several small navigation icons.

CREDENTIALS

Add passwords, SSH keys, etc. for Tower to use when launching jobs against machines, or when syncing inventories or projects.

AJOUTER UN CREDENTIAL

- Une fois sur la page Informations d'identification, cliquez sur le bouton vert + AJOUTER pour ajouter des informations d'identification.



AJOUTER UN CREDENTIAL

- Nommez les informations d'identification, sélectionnez TYPE et cliquez sur le bouton vert SAVE .

* NAME

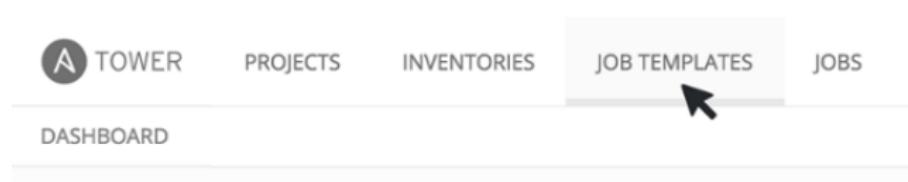
Demo Credential

* TYPE 

Machine

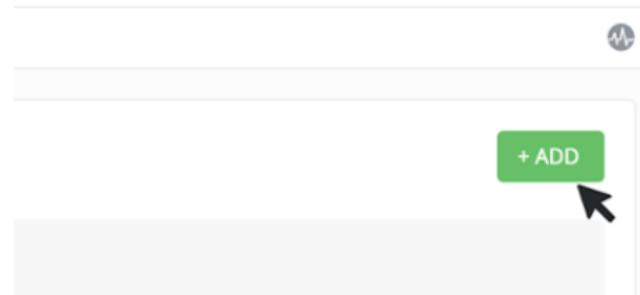
Créer un template job

- Click on JOB TEMPLATES in the top menu .



Créer un template job

- Click on JOB TEMPLATES dans le menu supérieur .



Créer un template job

- Une fois sur la page Modèles de travail, cliquez sur le bouton vert + AJOUTER pour ajouter un modèle de travail.

NEW JOB TEMPLATE

DETAILS COMPLETED JOBS PERMISSIONS NOTIFICATIONS

* NAME
Demo Job Template

* INVENTORY 
  Prompt on launch

Créer un template job

- INVENTORY et PROJECT seront ajoutés car un seul de ces logiciels a été créé dans Tower

***JOB TYPE** ⓘ

Run

Prompt on launch

***PLAYBOOK** ⓘ

hello_world.yml

Créer un template job

- Sélectionnez JOB TYPE , choisissez un PLAYBOOK et cliquez sur le bouton vert SAVE .

***JOB TYPE** ⓘ

Run

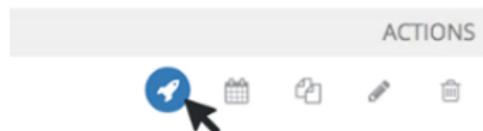
Prompt on launch

***PLAYBOOK** ⓘ

hello_world.yml

Créer un template job

- Cliquez sur TEMPLATES JOB dans le menu du haut, puis cliquez sur l'icône fusée pour lancer le Job



Une utilisation réelle de la NASA

- Considérons le défi commercial auquel la NASA a été confrontée.
- La NASA devait déplacer 65 applications d'un centre de données basé sur du matériel traditionnel
- vers un environnement basé sur le cloud
- pour une plus grande agilité et des économies de coûts.
- En raison de la rapidité d'exécution, de nombreuses applications ont été migrées vers un environnement cloud.



Une utilisation réelle de la NASA

- Cela a créé un environnement qui couvrait plusieurs clouds virtuels (VPC) et comptes AWS impossibles à gérer facilement.
- Même des choses simples, comme s'assurer que chaque administrateur système avait accès à chaque serveur, ou un simple correctif de sécurité, étaient extrêmement lourdes.



Une utilisation réelle de la NASA

- La solution consistait à utiliser Ansible Tower pour gérer et planifier l'environnement cloud.
- Pour résoudre les problèmes rencontrés par la NASA avec le manque de gestion centralisée et un environnement diversifié
- Ils ont évalué plusieurs solutions et décidé de mettre en œuvre Ansible Tower.
- La NASA exploite maintenant Ansible Tower pour gérer son environnement de manière très organisée et planifiée.



Comment la NASA utilise Ansible

- **Ansible Tower** était doté d'un tableau de bord contenant le résumé de l'état de tous les hôtes et travaux
- Permettant à la NASA de regrouper tout le contenu et de gérer les autorisations d'accès de différents départements.
- de scinder l'organisation en associant contenu et autorisation de contrôle pour les groupes.

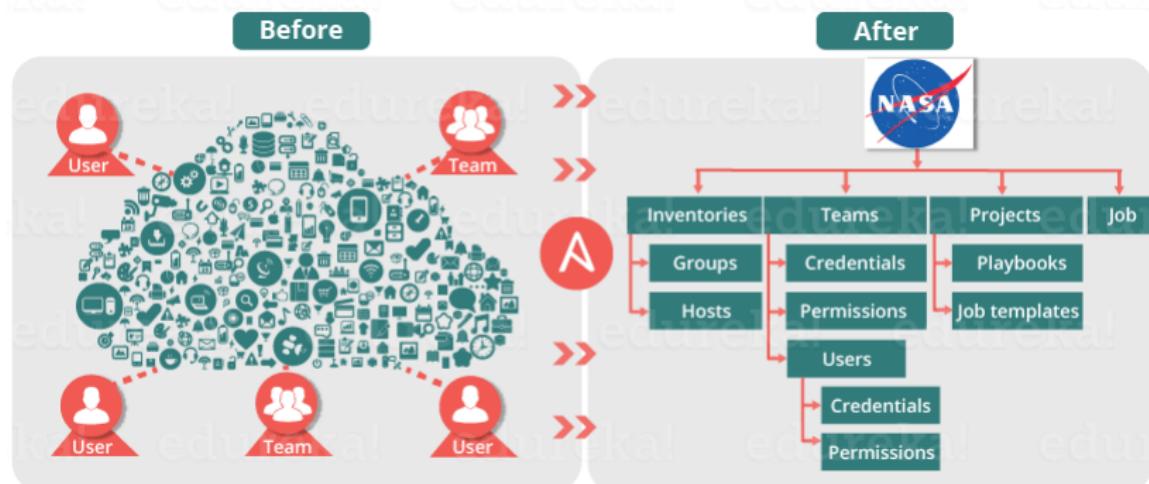
Comment la NASA utilise Ansible

- Ansible Tower est une interface Web permettant de gérer Ansible.
- L'un des principaux éléments de la liste de souhaits des utilisateurs d'Ansible
- une interface utilisateur conviviale pour la gestion de déploiements rapides et la surveillance de ses configurations.
- Ansible Management a répondu à cela avec Ansible Tower.

Comment la NASA utilise Ansible

- Ansible a réparti les tâches entre les équipes en attribuant divers rôles.
- Gère le nettoyage de l'historique des travaux,
- Des flux d'activité
- Des données marquées pour suppression et des informations de suivi du système.

Comment la NASA utilise Ansible



la NASA a réalisé les économies suivantes

- Les serveurs d'applications Web de la NASA font l'objet de correctifs routiniers et automatiques via Ansible Tower avec un très simple playbook de 10 lignes.
- Ansible est également utilisé pour résoudre les problèmes de sécurité
- il est utilisé pour résoudre les problèmes OpenSSL.
- Cela a non seulement permis de gagner du temps, mais a également permis de résoudre rapidement un problème de sécurité extrêmement préoccupant.

la NASA a réalisé les économies suivantes

- Grâce à la mise en œuvre d'Ansible, la NASA est mieux équipée pour gérer son environnement AWS.
- Ansible a permis à la NASA d'améliorer les opérations et la sécurité de ses clients.
- augmentation l'efficacité en équipe.
- Si on voit par les chiffres :
- La mise à jour de nasa.gov est passée de plus d'une heure à moins de cinq minutes.
- Les mises à jour des correctifs de sécurité sont passées d'un processus de plusieurs jours à 45 minutes.

la NASA a réalisé les économies suivantes

- Réalisation de la surveillance de la mémoire RAM et du disque en temps quasi réel (sans agents)
- Mise en service de comptes de système d'exploitation sur l'ensemble de l'environnement en moins de 10 minutes
- La définition des AMI standard (Amazon Machine Image) est passée de 1 heure de configuration manuelle à un processus d'arrière-plan invisible et transparent.
- Le temps d'installation des piles d'application a été réduit de 1 à 2 heures à moins de 10 minutes par pile.

Plan

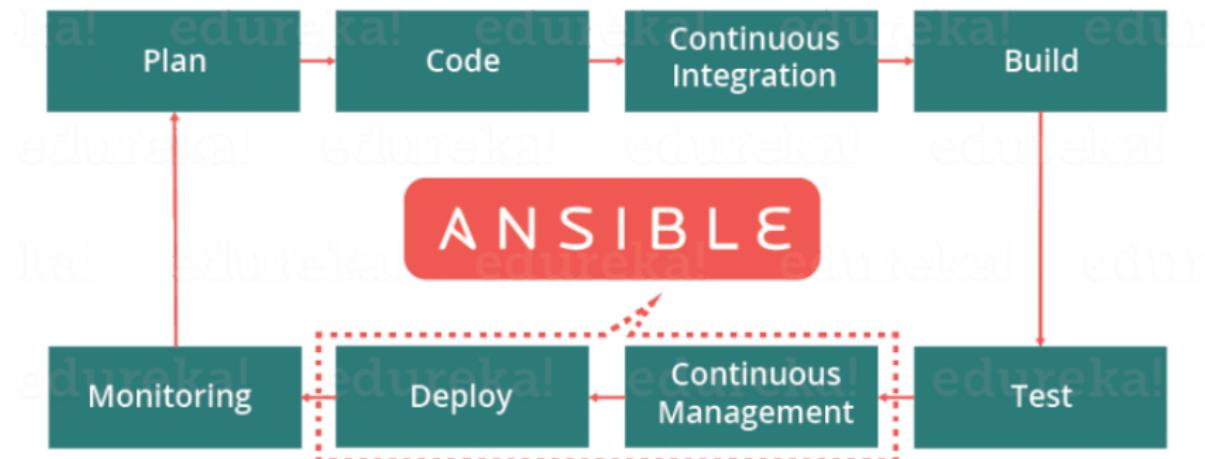
- 12 Mise en œuvre d'Ansible dans un environnement DevOps
 - Ansible dans DevOps

- 13 Révision approfondie

Ansible dans DevOps

- Dans DevOps, le développement et les opérations sont intégrés.
- Cette intégration est très importante pour la conception d'applications modernes pilotées par des tests.
- Ansible intègre cela en fournissant un environnement stable à la fois pour le développement et les opérations, ce qui permet une orchestration fluide.

DevOps



Ansible dans DevOps

- Ansible gère l'ensemble de l'infrastructure DevOps.
- Lorsque les développeurs commencent à penser à l'infrastructure dans le cadre de leur application, c-a-d en tant infrastructure en tant que code (IaC), la stabilité et les performances deviennent normatives.
- L'infrastructure en tant que code est le processus de gestion et de approvisionnement d'une infrastructure informatique (processus, serveurs sans système d'exploitation, serveurs virtuels, etc.)

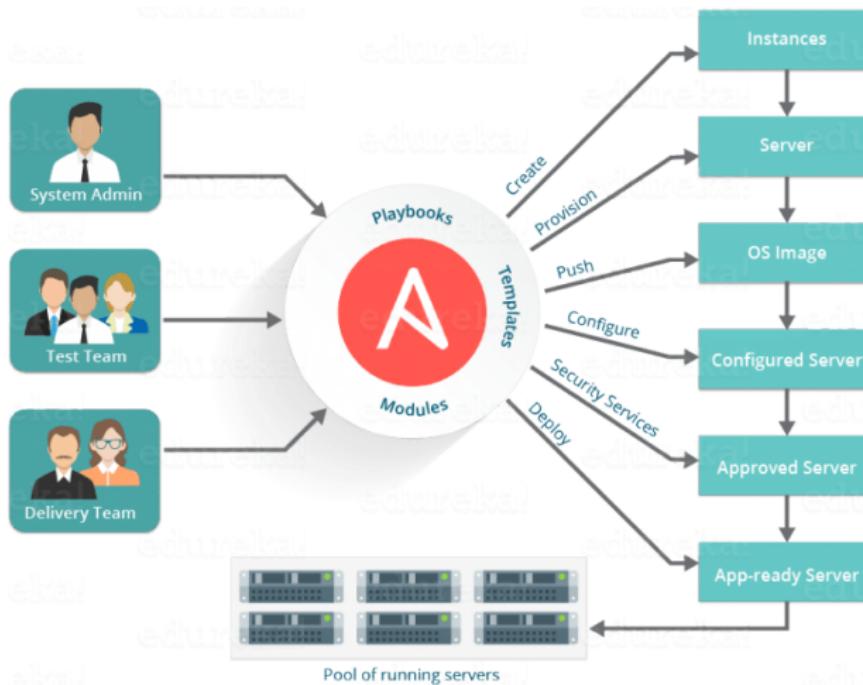
Ansible dans DevOps

- leur configuration via des fichiers de définition pouvant être traités par une machine, plutôt que de la configuration matérielle physique ou de l'utilisation d'outils de configuration interactifs.
- C'est ici que l'automatisation Ansible joue un rôle majeur et se distingue parmi ses pairs.

Ansible dans DevOps

- Dans DevOps, les administrateurs système travaillent en étroite collaboration avec les développeurs
- la vitesse de développement est améliorée et les activités consacrées
 - à l'optimisation des performances
 - à l'exécution des tâches
 - à la réalisation de tâches qui prennent plus de temps et moins de temps à la résolution des problèmes

Ansible simplifie les tâches des administrateurs système et des autres utilisateurs



Plan

- 12 Mise en œuvre d'Ansible dans un environnement DevOps
- 13 Révision approfondie