

ANSIBLE, AUTOMATISER LA GESTION DES SERVEURS

- **type:** formation
jours: 2
objectif: Maitriser Ansible!

PRÉSENTATION

- Du formateur
 - De **DocDoku**
-

À PROPOS DE VOUS

- Expérience avec Ansible
- Ce que vous attendez de la formation
- Vos projets à venir utilisant Ansible

MODALITÉS

- Horaires : 9h -> 17h30
- Pauses : 15 minutes matin et après midi
- Déjeuner : 1h30

OBJECTIFS PÉDAGOGIQUE

- Installer **Ansible** / Red Hat Ansible Engine.
- Créer et mettre à jour des **inventaires** d'hôtes gérés.
- Créer des **playbooks** pour automatiser les tâches d'administration.
- Créer des playbooks complexes pour gérer de gros **projets**.
- Simplifier le développement de playbooks à l'aide des **rôles** Ansible.
- **Protéger** des données sensibles utilisées par Ansible à l'aide d'Ansible Vault.
- Comprendre un premier niveau d'utilisation d'**Ansible Tower** et de Vagrant.

AGENDA

- [1] Positionnement d'Ansible
- [2] Installation et configuration d'Ansible
- [3] Les commandes Ad Hoc
- [4] Les PlayBooks
- [5] Ecrire du code modulaire
- [6] Commandes Avancées

[1] POSITIONNEMENT D'ANSIBLE

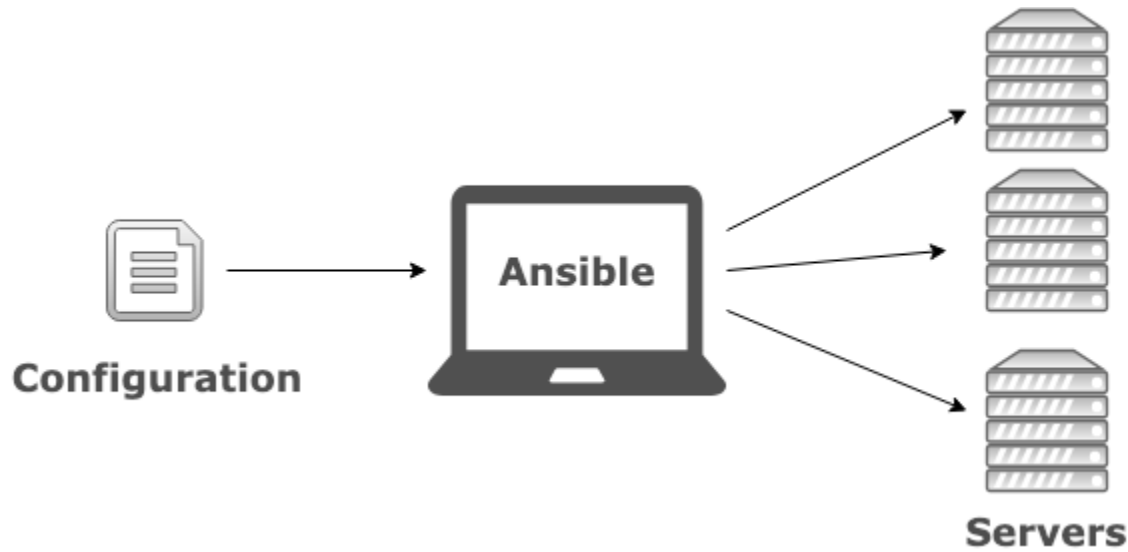
ANSIBLE ET DEVOPS

VUE GÉNÉRALE ANSIBLE

- Ansible :
 - Outil pour le **deploiement** ou changements **automatisés** des serveurs
 - Ecrit en Python
 - Système **agentless** capable de piloter les OS Windows, Linux, Unix, Mac

INTRODUCTION À ANSIBLE

Ansible est une plate-forme logicielle libre pour la **configuration** et la **gestion** des ordinateurs



ANSIBLE, UN OUTIL LIBRE

- Open-source
- Projet débuté en 2012
- **RedHat** aux commandes (rachat d'Ansible Inc. en 2015)
- Ansible est distribué sous la licence GPL 3.0
- 48K stars github
- 20k forks
- Dernière version en date 2.11.1 (mai 2021)

INFRASTRUCTURE AS CODE

Concepts

- **Gérer** et **provisionner** des machines
- Définition via des fichiers
- Pas de configuration manuelle

Avantages

- Réduction du cout
- Rapidité d'exécution
- Diminution du risque
- Traçabilité & reproductibilité

GESTION DE CONFIGURATION

Un outil principalement conçu pour

- **Configurer** et **gérer** des ordinateurs (serveurs ou clients)
- Réaliser des **déploiements automatiques**
- **Installer** des logiciels
- **Exécuter** des tâches/scripts

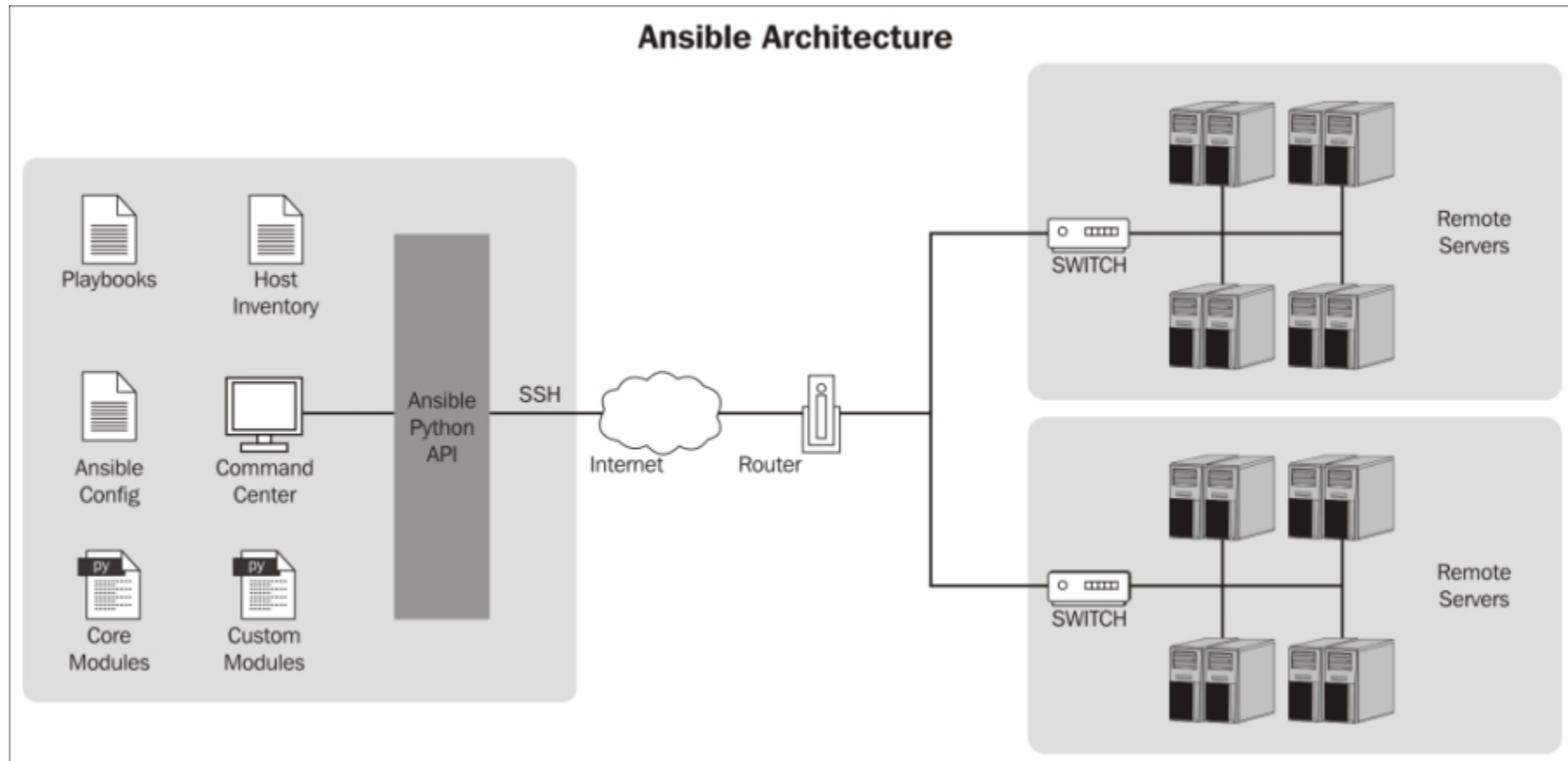
OUTILS ALTERNATIFS À ANSIBLE

- Rudder
- Chef
- Puppet
- SaltStack
- CFEngine
- GitlabCI
- Docker ?

SSH

- Ansible se base sur **SSH**
- Envoie des commandes via SSH aux ordinateurs distants
- **Pas d'agent** à installer sur la machine distante
- Très pratique mais pas le plus performant

ARCHITECTURE ANSIBLE



PRINCIPE DE FONCTIONNEMENT

- Une machine de contrôle Ansible - **Control Node**
 - Emet des commandes Ad Hoc vers des machines distantes via l'outil ansible - **Managed Nodes**
 - Exécute un jeu d'instruction via des playbooks
- Le fichier d'**inventaire** de l'hôte détermine les machines cibles

LES PRINCIPAUX COMPOSANTS D'ANSIBLE

- Modules
- Inventaires
- PlayBooks
- Plugins

[2] INSTALLATION ET CONFIGURATION

INSTALLATION ET MISE EN OEUVRE

INSTALLATION

- Par les dépôts du système (Linux)
- Par le gestionnaire de modules Python

https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

PRÉREQUIS D'INSTALLATION SUR LA MACHINE CLIENTE

Ansible peut être lancé depuis n'importe quelle machine ayant Python 2 (2.7) ou Python 3 (versions 3.5 et supérieures)

- Red Hat
- Debian
- CentOS
- macOS,
- BSDs
- ...

PRÉREQUIS D'INSTALLATION SUR LA MACHINE DISTANTE

- SSH
- Python 2 (2.7) ou Python 3 (versions 3.5 et supérieures)
 - Nécessaire pour les modules
 - Peut s'installer via une commande Ansible !

```
$ ansible myhost --become -m raw -a "yum install -y python2"
```

CONFIGURATION D'ANSIBLE

- Via un fichier de configuration
 - `/etc/ansible/ansible.cfg`
 - <https://github.com/ansible/ansible/blob/devel/examples/ansible.cfg>
- Via les options de la ligne de commande pour surcharger les valeurs de configuration

REMARQUES

- La **machine de contrôle** doit être basée sur **Linux**
(Windows n'est pas autorisé)
- La machine cible (hôte / noeud) peut être Linux / macOS / Windows
- Seul Python est nécessaire
- Aucun logiciel d'agent requis

GUIDE UTILISATEUR INSTALLATION ANSIBLE

- Ansible s'installe uniquement sur la machine de contrôle
- Via :
 - les paquets des distributions (Ubuntu/Debian)
 - les dépôts tiers EPEL (Redhat/Centos)
 - pip

GUIDE UTILISATEUR INSTALLATION ANSIBLE

- Installation d'Ansible sur la machine de contrôle
 - via les paquets de distribution [1/2]
 - Ubuntu

```
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo apt-add-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible
```


GUIDE UTILISATEUR INSTALLATION ANSIBLE

- Installation d'Ansible sur la machine de contrôle
 - via les paquets de distribution [2/2]
 - Debian

```
$ sudo apt-key adv --keyserver keyserver.ubuntu.com  
--recv-keys 93C4A3FD7BB9C367  
$ sudo apt update  
$ sudo apt install ansible
```

GUIDE UTILISATEUR INSTALLATION ANSIBLE

- Installation d'Ansible sur la machine de contrôle
 - via les dépôts tiers EPEL (Redhat/Centos)
 - Centos

```
$ sudo yum install ansible
```

GUIDE UTILISATEUR INSTALLATION ANSIBLE

- Installation d'Ansible sur la machine de contrôle
 - via PIP
 - Si PIP n'est pas encore installé

```
$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py  
$ python get-pip.py --user
```

- Puis installer Ansible

```
pip install --user ansible
```

GUIDE UTILISATEUR INSTALLATION ANSIBLE

- Configurer l'accès SSH de la machine cible (noeud)
 - Générer une clé sur la machine de contrôle :

```
ssh-keygen
```

- Afficher le contenu de la clé publique :

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

- Copier le contenu de la clé
 - Copier la clé publique SSH de la machine de contrôle sur le noeud

PILOTER LES MACHINES WINDOWS

PRÉREQUIS WINDOWS

- Windows est un système d'exploitation non POSIX
 - Différences majeures => modules spécifiques
 - L'hôte Windows doit
 - être version 7, 8.1, 10, Server 2008, 2008 R2
 - PowerShell 3.0
 - .NET 4.0
 - WinRM activé

WINRM

- WinRM => Windows Remote Management
- Service Windows permettant de gérer l'ordinateur depuis le réseau
- Disposer d'un compte utilisateur est indispensable
- Protocole SOAP (transport HTTP)
- Deux composants principaux du service WinRM à paramétrer
 - *listener*
 - *service*

ECOUTEURS WINRM (LISTENERS)

Les services WinRM écoutent les requêtes sur un ou plusieurs ports.

Pour afficher les écouteurs en cours d'exécution sur le service WinRM :

```
winrm enumerate winrm/config/Listener
```

Exemple de retour possible

```
Address = *  
Transport = HTTP  
Port = 5985  
Hostname  
Enabled = true  
URLPrefix = wsman  
CertificateThumbprint  
ListeningOn = 10.0.2.15, 127.0.0.1, 192.168.56.155, ::1,  
fe80::5efe:10.0.2.15%6, fe80::5efe:192.168.56.155%8,  
fe80:: ffff:ffff:fffe%2, fe80::203d:7d97:c2ed:ec78%3,  
fe80::e8ea:d765:2c69:7756%7
```


- **Transport:** HTTP ou HTTPS
- **Port:** Le port d'écoute
 - par défaut 5985 pour HTTP et 5986 pour HTTPS
 - correspond à la variable hôte *ansible_port*
- **URLPrefix:** Le préfixe de l'URL du WS
 - par défaut wsman
 - Si modifié, la variable hôte *ansible_winrm_path* doit être définie
- **CertificateThumbprint:** Si HTTPS, il s'agit de l'empreinte numérique du certificat

CONFIGURATION DU SERVICE

Si le service n'est pas actif, il faut exécuter en tant qu'Administrateur :

```
winrm quickconfig
```

OU

```
winrm quickconfig -transport:https
```

Côté Ansible, pour chaque machine cible Windows :

```
ansible_host: 192.168.1.40
ansible_user: Edouard Durand
ansible_password: secret
ansible_connection: winrm
ansible_winrm_transport: ntlm
ansible_port: 5985
ansible_winrm_scheme: http
```

LES INVENTAIRES

- Les fichiers d'inventaire sont des fichiers qui contiennent la **liste des serveurs**
 - DNS
 - IPs
 - Possibilité de faire des **groupes**

Un simple inventaire

```
ma_machine1 ansible_host=10.0.0.1 ansible_port=22  
ma_machine2 ansible_host=10.0.0.2 ansible_port=22
```

GESTION DES GROUPEES

Pour automatiser les commandes sur de multiples machines, on crée des groupes

```
[groupe_1]
```

```
10.0.0.2
```

```
10.0.0.3
```

```
[groupe_2]
```

```
some-domain.net
```

```
other-domain.com
```

```
[groupe_3]
```

```
192.168.1.23
```

```
...
```

GESTION DES GROUPEES

Il est possible d'utiliser des variables et énumérations

```
[groupe_1]  
10.0.0.[2:9]
```

```
[groupe_2]  
machine-[a:z].my-domain.net
```

```
[groupe_3]  
localhost someVar=SOME_VALUE
```

CAS D'USAGE DES GROUPES

Gestion des différentes plateformes et localisations

[paris-webservers]

www-paris-1.example.com

www-paris-2.example.com

[toulouse-webservers]

www-tls-1.example.com

www-tls-2.example.com

[webservers:children]

paris-webservers

toulouse-webservers

Il est ensuite possible de jouer des commandes sur tous les webservers, ou bien de limiter selon une zone donnée

INVENTAIRE DYNAMIQUE

Les inventaires statiques marchent bien mais

- obligent un maintien des inventaires lorsque de nouvelles machines sont ajoutées, détruites ou reconstruites.

Conséquence : changement de code pour chaque modification de l'infrastructure

INVENTAIRE DYNAMIQUE

- Mise en place depuis Ansible 2.0
- Nouvelles fonctionnalités
 - Chargement des inventaires depuis un script
 - Des scripts communautaires pour les principaux cloud providers (AWS, Azure, OpenStack, DigitalOcean, GCP, ...)

LES OUTILS EN LIGNE DE COMMANDES

- **ansible** : Exécution d'une commande unique
- **ansible-playbook** : Exécution de playbook
- **ansible-doc** : Accès au listing + documentation des serveurs
- **ansible-vault** : Gestion de fichiers chiffrés
- **ansible-galaxy** : Accès au dépôt des rôles d'Ansible

CONFIGURATION ET INVENTAIRE DE PLUSIEURS NOEUDS CLIENT

- Ouvrir le fichier hosts qui se trouve dans /etc/ansible.

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

- Groupes de noeuds et association des adresses IP

```
[group_name]  
ALIAS NODE_IP
```

```
[webservers]  
WEB1 192.168.1.100  
WEB2 192.168.1.101  
WEB2 192.168.1.103
```

CONFIGURATION ET INVENTAIRE DE PLUSIEURS NOEUDS CLIENT

On peut aussi y créer des variables arbitraires (mise en valeur de machine dans l'inventaire, condition d'accès, etc.)

```
[webservers]  
192.168.1.100 ansible_user=root  
192.168.1.101 type=master  
192.168.1.103 type=slave
```

[3] LES COMMANDES AD HOC

COMMANDES AD-HOC

Commandes que l'on ne souhaite pas sauvegarder dans un fichier ansible => Permet d'**exécuter** une tâche **directement** depuis une ligne de commande

- Cas d'usage :
 - Reboot / éteindre une machine
 - Redémarrer un service
 - Modifier une configuration
 - ...

COMMANDES AD-HOC

Redémarrer un serveur :

```
$ ansible myhost -a "/sbin/reboot"
```

Installer un paquet

```
$ ansible myhost --become -m raw -a "yum install some-package"
```

Avec escalade de privilèges

```
$ ansible myhost -a "/some/command" -u username --become
```

Parallélisme (par défaut 5)

```
$ ansible myhosts -a "/sbin/reboot" -f 10
```

AUTHENTICATION

Gestion de l'authentification par mot de passe ou clé SSH

Prompt du mot de passe

```
$ ansible myhost -a "/some/command" -u user -k
```

Utilisation d'une clé SSH

```
$ ansible myhost "/some/command" -u user --private-key=/path/to/key
```


UTILISATEURS ET GROUPE

Manipulation des comptes utilisateurs existants :

- Modification
- Suppression

```
$ ansible all -m user -a "name=foo password=<crypted password here>"
```

```
$ ansible all -m user -a "name=foo state=absent"
```

DEPLOIEMENT À PARTIR DU "SOURCE CONTROL"

Déploiement à partir de git

```
$ ansible webservers -m git -a  
"repo=https://foo.example.org/repo.git dest=/srv/myapp version=HEAD"
```

GESTION DES SERVICES

- Démarrer un service

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

- Stopper un service

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

TRANSFERT DE FICHIERS

Envoi de fichiers avec une commande ad-hoc avec le module copy

- src : chemin sur la machine cliente
- dest : chemin sur le serveur distant

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

Avec le module file (plus d'options, gestion des droits, création automatique de répertoires, ...)

```
$ ansible myhost -m file -a "dest=/etc/hosts mode=600"
```

GESTION DES PAQUETS [1/2]

Gestion de **yum** et **apt**

S'assurer qu'un paquet est installé, sans le mettre à jour

```
$ ansible myhost -m yum -a "name=nginx state=present"
```

S'assurer qu'un paquet est installé, avec les dernières mises à jour

```
$ ansible myhost -m yum -a "name=nginx state=latest"
```

GESTION DES PAQUETS [2/2]

S'assurer qu'un paquet n'est pas installé

```
$ ansible myhost -m yum -a "name=acme state=absent"
```

OPÉRATIONS EN TÂCHE DE FOND

[1/3]

- Les opérations longues
 - peuvent être exécutées en arrière plan
 - potentiellement limitées dans le temps
 - Exemple : opération exécutée en arrière plan avec un délai de 3600 secondes (-B) et sans interrogation (-P):

```
$ ansible all -B 3600 -P 0 -a  
    "/usr/bin/long_running_operation --do-stuff"
```

OPÉRATIONS EN TÂCHE DE FOND

[2/3]

- Vérification ultérieure de l'état de travail

```
$ ansible web1.example.com -m async_status -a "jid=488359678239.2844"
```


OPÉRATIONS EN TÂCHE DE FOND

[3/3]

- Faire une interrogation du statut
- Exemple :
 - Exécuter pendant 30 minutes maximum ($-B 30 * 60 = 1800$)
 - Interroger le statut ($-P$) toutes les 60 secondes

```
$ ansible all -B 1800 -P 60  
-a "/usr/bin/long_running_operation --do-stuff"
```

[4] LES PLAYBOOKS

VUE D'ENSEMBLE DES PLAYBOOKS

Les playbooks sont des fichiers YAML qui décrivent les commandes à réaliser sur les serveurs

EXEMPLE BASIQUE

```
- hosts: webservers
  become: true
  gather_facts: false
  tasks:
    - name: Hello world task
      raw: echo "Hello server"
    - name: Ping internet
      raw: ping -c 1 8.8.8.8
    - name: Install python
      raw: yum install python
```

SYNTAXE YAML

GÉNÉRALITÉS

- Acronyme : Yet Another Markup Language
- Proposé par Clark Evans en 2001
- Utilise les concepts d'autres langages comme XML et Json
- Dédié à la description de données
- S'intègre avec beaucoup de langages de programmation
- Interdit l'utilisation des tabulations

SYNTAXE YAML

LES COMMENTAIRES SONT PRÉCÉDÉS D'UN
'#'

```
# Ceci est un commentaire YAML
```

LES SCALAIRES

- Les scalaires sont l'ensemble des types YAML qui ne sont pas des collections (listes ou tableaux associatifs)

```
# Les chaînes de caractères
# Les nombres
- Entier
- Octal
- Décimal
- Hexadécimal
...
# Booléen
# Dates
...
```

SYNTAXE YAML

LES SCALAIRES

Les scalaires peuvent être en général déclarés simplement

■ Exemple :

```
- Entier: 12
- Octal : 014
- Hexa : 0xC
- Booleen : true ou false
- Date : 2020-09-25
- Chaîne de caractère : Je suis un string.
...
```


SYNTAXE YAML

LES SCALAIRES

LES CHAÎNES DE CARACTÈRES [1/2]

- Une chaîne de caractères peut être entre simples quotes

```
'A singled-quoted string in YAML'
```

- Si contient des apostrophes : doubler les apostrophes intermédiaires pour les échapper

```
'A single quote '' in a single-quoted string'
```

- Si retour à la ligne, on utilise les doubles quotes

```
"A double-quoted string in YAML\n"
```

SYNTAXE YAML

LES SCALAIRES

LES CHAÎNES DE CARACTÈRES [2/2]

- Multi-lignes, avec conservation des retours à la ligne

```
adresse: |  
  DocDoku  
  76 allée Jean Jaurès  
  31000 Toulouse
```

- Et sans

```
indication: >  
  Suivez la flèche rouge  
  continuez tout droit puis  
  tournez à gauche après le gros bananier.
```

SYNTAXE YAML

LES SCALAIRES

- Autres types de scalaires reconnus par les parseurs YAML

```
nul: null
nul bis: ~
vrai: true
vrai bis: yes
vrai ter: on
faux: false
faux bis: no
faux ter: off
```

SYNTAXE YAML

LES COLLECTIONS [1/3]

- Il existe deux types de collections reconnues par YAML
 - les séquences (ou listes)
 - les tableaux associatifs

SYNTAXE YAML

LES COLLECTIONS

LES SÉQUENCES [1/2]

■ Syntaxes :

- PHP
- Perl
- Python # une séquence peut avoir une sous-séquence
 - Python 1
 - Python 2

SYNTAXE YAML

LES COLLECTIONS

LES SÉQUENCES [2/2]

- Les séquences supportent le format JSON

```
json_map: {"clé": "valeur"}  
json_seq: [1, 2, 3, "soleil"]
```

SYNTAXE YAML

LES COLLECTIONS

LES TABLEAUX [1/2]

Appelés Map ou Dictionnaires dans certains langages, ils associent une valeur à une clé

- Les tableaux sont créés par indentation

```
une_map_imbriquée:  
  clé: valeur  
  autre_clé: autre valeur  
  autre_map_imbriquée:  
    bonjour: bonjour
```

SYNTAXE YAML

LES COLLECTIONS

LES TABLEAUX [2/2]

- Les clés
 - pas forcément de type string
 - 0.25 : une clé de type flottant
 - peuvent être sur plusieurs lignes débutant par "?"

```
? |  
  ceci est une clé  
  sur de multiples lignes: et ceci est sa valeur
```


SYNTAXE YAML

RÉFÉRENCES

Les données répétées peuvent être signalées par **&** puis référencées par *****

```
adresse-facturation: &ad001
  ville:  Toulouse
  cp:    31000
  complement: |
              76 allée Jean-Jaurès
              Bat A, Bal 197

adresse-livraison:  *ad001
```

MACHINES ET UTILISATEURS

Dans chaque playbook, on spécifie la machine et l'utilisateur

```
- hosts: my-example-domain.com  
  remote_user: root
```

On peut aussi définir un utilisateur par tâche

```
- hosts: webservers  
  become: true  
  gather_facts: false  
  tasks:  
    - name: Hello world task  
      raw: echo "Hello server"  
      remote_user: foo
```

LISTE DES TÂCHES

Le rôle d'une tâche est d'exécuter un "module"

Les tâches sont exécutées :

- Dans l'ordre de définition
- Sur tous les serveurs
- L'une après l'autre

Si l'une des machines rencontre une erreur, elle sera exclue pour les commandes suivantes

DÉFINITION D'UNE TÂCHE (1/2)

Le but d'une tâche est d'exécuter un **module** avec des paramètres spécifiques

Une tâche doit être **idempotente** (l'opération a le même effet si on l'applique plusieurs fois)

Chaque tâche doit avoir un nom et une commande (ou module)

DÉFINITION D'UNE TÂCHE (2/2)

Exemple d'une tâche qui démarre un service http

Nom du module : service

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

TÂCHES NE NÉCESSITANT PAS PYTHON

Chaque module a besoin de Python, sauf le module "raw"

Cela permet par exemple d'installer Python...

```
tasks:  
- name: Install python  
  raw: yum install python
```

GESTION DES ERREURS

L'exécution du playbook s'arrête à la moindre erreur

On peut toutefois ignorer les erreurs

```
tasks:  
  - name: Do something and ignore failures  
    shell: /some/command || /bin/true
```

Ou bien avec un module :

```
tasks:  
  - name: Do something and ignore failures  
    shell: /some/command  
    ignore_errors: true
```

GESTION DES ERREURS: BLOCK

Souvent si une erreur se produit => tout un bloc en échec

Equivalent du try/catch/finally

```
tasks:
  - block:
    - debug:
      msg: go tache 1
    - raw: /bin/false
    - debug:
      msg: go tache 2
  rescue:
    - debug:
      msg: dans le rescue
  always:
    - debug:
      msg: toujours exécuté
```


UTILISATION DE VARIABLES

Il est possible d'injecter des variables dans les playbooks

La façon la plus simple est de les définir directement dans le playbook

```
- hosts: dev
vars:
  var1: hello
  var2: world
tasks:
  - name: Say hello
    raw: echo "{{ var1 }}" "{{ var2 }}"
```

EXTERNALISER LES VARIABLES

On peut aussi définir ces variables dans un fichier externe

Playbook :

```
- hosts: dev
  vars_files:
    - vars.yml
```

vars.yml :

```
var1: hello
```

AFFICHER UNE VARIABLE

Permet de débbugger / s'assurer qu'une variable est bien définie

Utilisation du module "debug"

```
vars:  
  foo: bar  
tasks:  
  - name: Display var  
    debug:  
      msg: Hello {{ foo }}
```

PROMPT UTILISATEUR

Les variables peuvent être définies par une entrée utilisateur

Playbook :

```
- hosts: dev
  vars_prompt:
    - name: couleur
      prompt: Votre couleur préférée ?
      private: no
```


VARIABLES

Les variables peuvent être propres aux machines

Playbook :

```
host_vars/  
  host1.yml          # Variables pour le host host1  
  host2.yml          # Variables pour le host host2  
  
group_vars/  
  group1.yml         # Variables pour le groupe group1  
  group2.yml         # Variables pour le groupe group2
```

VARIABLES: RETOUR DE COMMANDE

- Les commandes peuvent produire un résultat
- Ansible offre la structuration du retour (parsing)
- Accrochés à la variable **register**
 - propriété de la tâche
- Lire la documentation du module pour connaître les *returned values*
-  variable propre au host

VISUALISER LES OPÉRATIONS EN DRY-RUN

Permet de s'assurer de l'ordre et des tâches à exécuter

Option **--check**

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

Ou bien directement dans le playbook au niveau de la tâche

```
tasks:  
  - name: "Won't be played"  
    command: /some/command  
    check_mode: yes
```

LES FACTS (1/2)

Récupération de données de la machine distantes sous forme de variables

Récupération d'un seul fact (ip de la machine)

```
$ ansible myhost -m setup -a 'filter=ipv4'
```

Liste complète des facts :

```
ansible myhost -m setup
```

Le retour de cette commande renvoie la totalité des facts

LES FACTS (2/2)

Utilisation de variables au sein du playbook

```
- hosts: myhost
gather_facts: true # permet de collecter les facts
tasks:
  - name: Display hostname
    debug:
      msg: "Welcome to {{ ansible_hostname }}"
  - name: Display IP
    debug:
      msg: "This is my IP on eth0 ! {{ ansible_eth0.ipv4.address }}
```

STRUCTURES DE CONTRÔLE

Le "if" d'Ansible : **when**

Permet par exemple de ne pas exécuter une tâche selon la valeur d'une variable

```
tasks:  
  - name: "shut down only debian systems"  
    command: /sbin/shutdown -t now  
    when: ansible_facts['os_family'] == "Debian"
```

WHEN

Exemple plus poussé, avec récupération de variable

Mots clés : **failed**, **succeeded**, **skipped**

```
tasks:
- command: /bin/false
  register: foo
  ignore_errors: True
- command: /some/command
  when: foo is failed
- command: /other/command
  when: foo is succeeded
- command: /yet/another/command
  when: foo is skipped
```

LOOP

Permet de boucler - mot clé **item**

```
tasks:  
  - command: echo {{ item }}  
    loop: [ 1, 2, 3, 4, 5 ]
```

Valeur par défaut si variable non définie (**default**)

```
tasks:  
  - command: echo {{ item }}  
    loop: {{ someList|default([1,2,3]) }}
```

Boucle sur un dictionnaire (**item.key**, **item.value**)

```
tasks:  
  - command: "echo {{ item.key }} : {{ item.value }}"  
    loop: "{{ query('dict', mydict|default({})) }}"
```

TEMPLATES

- Généralités sur les templates
 - basé sur jinja2 :
<https://jinja.palletsprojects.com/en/2.10.x/templates/>
 - gère :
 - les boucles
 - les listes
 - les variables
 - les tests logiques

TEMPLATES

- syntaxe :
 - variable: **{{variable}}**
 - Les instructions: **{%instruction%**
 - Les commentaires: **{#commentaire#}**
- Filtre ou formatage

```
{{ some_variable | to_json }}  
{{ some_variable | to_yaml }}
```

TEMPLATES

- Variables disponibles pour les templates
 - **ansible_managed**: valeur définie dans ansible.cfg
 - **template_host**: contient le host name
 - **template_uid**: id de l'utilisateur
 - **template_path**: chemin du template
 - **template_fullpath**: chemin absolu du template
 - **template_destpath**: chemin du template sur la machine distante (depuis Ansible 2.8)
 - **template_run_date**: date à laquelle le fichier final a été créé

TEMPLATES

- Exemples d'utilisation de templates [1/3]

```
name: Template a file to /etc/files.conf
template:
  src: /mytemplates/foo.j2
  dest: /etc/file.conf
  owner: bin
  group: wheel
  mode: '0644'
```


TEMPLATES

- Exemples d'utilisation de templates [2/3]
 - Template qui crée un DOS-Style

```
template:  
  src: config.ini.j2  
  dest: /share/windows/config.ini  
  newline_sequence: '\r\n'
```

TEMPLATES

- Exemples d'utilisation de templates [3/3]
 - Template qui copie un nouveau fichier "sudoers" après avoir validé avec la commande "visudo"

```
template:  
  src: /mine/sudoers  
  dest: /etc/sudoers  
  validate: '/usr/sbin/visudo -cf %s'
```

[5] ECRIRE DU CODE MODULAIRE

NOTIFICATIONS ET HANDLERS

Un handler est une tâche qui peut être lancée automatiquement à la suite d'autres tâches

```
tasks:
  - name: Start Nginx
    service:
      name: nginx
      state: started
      notify: my handler name

handlers:
  - name: my handler name
    debug:
      msg: "Nginx is started !!!"
```

MODULARISER LE PLAYBOOK [1/2]

Ansible se base sur certains noms de dossiers/fichiers pour automatiser et faciliter l'écriture des playbooks. Voici un exemple d'organisation

```
production          # Inventaire pour la production
staging             # Inventaire pour la pre-prod

group_vars/
  group1.yml         # Variables du groupe "group1"
  group2.yml         # Variables du groupe "group2"
```

MODULARISER LE PLAYBOOK [2/2]

```
host_vars/  
  myhost1.yml          # Variables pour le host myhost1  
  myhost2.yml          # Variables pour le host myhost2  
  
library/               # Modules maison  
filter_plugins/        # Filtres maison  
site.yml               # Playbook principal  
group1.yml             # Playbook du groupe group1  
group2.yml             # Playbook du groupe group2
```

LES RÔLES (1/2)

Les rôles servent à la modularisation des playbooks

Ils peuvent être **ré-utilisés**

Un rôle peut contenir tout ou une partie des dossiers suivants :

```
tasks - Tâches à exécuter
handlers - Les handlers associés
defaults - Variables par défaut du rôle
vars - Surcharge des variables par défaut
files - Fichiers à transférer
templates - Templates à déployer
meta - Méta-données
```

LES RÔLES (2/2)

Dans les dossiers précédemment cités, on retrouve un fichier "main.yml"

Ce nom de fichier est utilisé par Ansible

On peut toutefois re-découper la liste des tâches si besoin

```
# roles/myrole/tasks/main.yml
- import_tasks: redhat.yml
  when: ansible_facts['os_family']|lower == 'redhat'
- import_tasks: debian.yml
  when: ansible_facts['os_family']|lower == 'debian'
- import_tasks: other-tasks.yml
```


UTILISATION DES RÔLES

Dans un playbook, on peut référencer les rôles à jouer

```
- hosts: myhosts
  roles:
    - role: '/path/to/myrole'
```

Surcharger des variables

```
- hosts: myhosts
  tasks:
    - include_role:
        name: myrole
  vars:
    foo: 'baz'
    var2: 8080
```

UTILISATION CONDITIONNELLE DE RÔLES

La structure de contrôle **when** est utilisable avec les rôles

```
- hosts: myhosts
  tasks:
    - include_role:
        name: myrole
        when: "ansible_facts['os_family'] == 'RedHat'"
```

LES MODULES "BUILT-IN"

Les commandes ad-hoc et les playbooks utilisent des modules

```
$ ansible myhost -m service -a "name=httpd state=started"
$ ansible myhost -m ping
$ ansible myhost -m command -a "/sbin/reboot -t now"
...
```

La liste des modules "built-in" est longue

[Liste des modules par
catégorie]<https://docs.ansible.com/ansible/latest/modules/mod>

[Tous les
modules]https://docs.ansible.com/ansible/latest/modules/list_

ANSIBLE GALAXY

Ansible vient avec l'exécutable ansible-galaxy

Celui ci permet de télécharger des rôles écrits par la communauté et donc d'accélérer l'écriture de vos playbooks

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

UTILISATION D'ANSIBLE GALAXY

Pour installer un rôle dans ansible, une ligne de commande suffit

Exemple pour le rôle [insspb.postgresql](#)

```
$ ansible-galaxy install insspb.postgresql
```

Une fois le rôle installé, on peut l'utiliser directement dans un playbook

```
- hosts: myhosts
  gather_facts: true
  roles:
    - insspb.postgresql
```

[6] FONCTIONS AVANCÉES

- Les tags
- Le vault
- Les lookups
- Les modules custom
- Les filtres custom

LES TAGS

Les tags servent à filtrer les playbooks pour n'en jouer qu'une partie

Définition des tags

```
tasks:
  - yum:
      name: "{{ item }}"
      state: installed
    loop:
      - nginx
      - memcached
    tags:
      - install_packages
```

Lancement de cette partie seulement

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

LE VAULT

Stocke de manière chiffrée les données sensibles

- Mot de passes
- Clés SSH
- Toute autre donnée sensible ...

Protégé lui même par mot de passe ou par une autre clé

Lors du lancement d'une commande ansible ou ansible-playbook, on devra alors ajouter l'option **--ask-vault-pass** ou **--vault-password-file**

GESTION DES FICHIERS CHIFFRÉS

Pour créer un vault (cela demandera un nouveau mot de passe)


```
$ ansible-vault create myvault.yml
$ cat myvault.yml
$ANSIBLE_VAULT;1.1;AES256
646266356633646266363436346338386365356534383733633633333564376163616!
3530353430623639393165366565376462366561656131340a3333646238656564306!
616633326335616334643864376663373064623835613835303331366535323331636!
3337653961393230390a3235383235323763366434343331353337633930363363363!
39653461353662356466666233613666393836636430366466653665653339303630
```

Pour l'éditer

```
$ ansible-vault edit myvault.yml
Vault password:
```

LES LOOKUPS

Récupère des données sur la machine cliente, qui sont hors du projet

 Exécuté sur la machine cliente, et non sur la machine distante

```
vars:
  motd_value: "{{ lookup('file', '/etc/motd') }}"
tasks:
  - debug:
      msg: "motd value is {{ motd_value }}"
```

LES MODULES CUSTOM

Si les modules built-in ne suffisent pas (très rare)

Prérequis

```
$ sudo apt update  
$ sudo apt install build-essential libssl-dev libffi-dev python-dev
```

CODE PYTHON D'UN MODULE CUSTOM (1/2)

Définition des arguments et du module

```
from ansible.module_utils.basic import AnsibleModule

def run_module():
    # Définition des paramètres que l'utilisateur peut renseigner
    module_args = dict(
        var1=dict(type='str', required=True),
        var2=dict(type='bool', required=False, default=False)
    )
    # Définition du module
    module = AnsibleModule(
        argument_spec=module_args,
        supports_check_mode=True
    )
    ''' code du module ... '''
```

CODE PYTHON D'UN MODULE CUSTOM (2/2)

```
# Retour du module
result = dict(
    changed=False,
    original_message='',
    message=''
)
# Gestion du dry-run
if module.check_mode:
    return result

''' code du module ....
    utilisation des variables et modification du result'''

module.exit_json(result)
```

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

LES FILTRES CUSTOM

Les filtres custom sont aussi écrits en Python, et se développent d'une façon similaire

Etude des filtres du core d'Ansible

[Lien

github]<https://github.com/ansible/ansible/blob/devel/lib/ansible>

ANSIBLE TOWER

- Interface Web, facilite l'utilisation d'Ansible
- S'appuie sur les Templates
- Ansible Tower
 - Centralise l'infrastructure avec contrôle d'accès basé sur les rôles
 - Planification des tâches et la gestion des stocks graphique
 - Payant

DASHBOARD ANSIBLE TOWER



Projects Inventories Job Templates Jobs

admin



87

Hosts

0

Failed Hosts

2

Inventories

0

Inventory Sync Failures

3

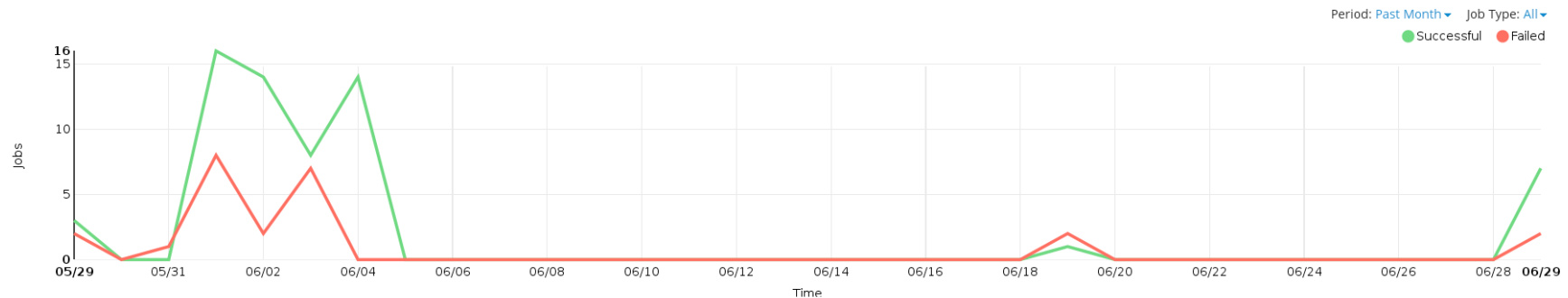
Projects

0

Projects Sync Failures

Job Status

Host Status



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

DASHBOARD ANSIBLE TOWER

Le Dashboard view est divisé en 3 parties

- La partie supérieure comprend
 - Hosts
 - Inventaires
 - Projets
- Le milieu montre
 - le statut des jobs
 - les derniers jobs lancés
 - les derniers Templates utilisés

GUIDE UTILISATEUR ANSIBLE TOWER

[Lien user guide] <https://docs.ansible.com/ansible-tower/latest/pdf/AnsibleTowerUserGuide.pdf>

RAPPELS DES BONNES PRATIQUES

- Automatiser les builds
- Tester dans les builds
- Le build doit être aussi rapide que possible
- Tester les builds dans des environnements clone de la production
- Faciliter l'accès à vos builds
- Automatiser les déploiements
- Idempotence

FIN DE LA FORMATION

Merci pour votre participation

LIENS UTILES

Ansible

- <https://www.ansible.com>
- <https://docs.ansible.com/ansible/latest/index.html>

OUTILS ALTERNATIFS

- <https://www.rudder.io>
- <https://www.chef.io>
- <https://puppet.com>
- <https://www.saltstack.com>

