Formation Ansible, automatiser la gestion des serveurs

Matthieu Falce

Octobre 2022

Matthieu Falce
Devops
Ansible

Au programme I

Devops

Ansible

A propos de moi – Qui suis-je ?

- Qui suis-je ?
  - Matthieu Falce
  - habite à Lille
  - ingénieur en bioinformatique (INSA Lyon)
- Qu'est ce que j'ai fait ?
  - ingénieur R&D en Interaction Homme-Machine (IHM), Inria Lille, équipe Mint puis Mjolnir
  - développeur *fullstack* / *backend* à FUN-MOOC (France Université Numérique)

A propos de moi – Actuellement

- entrepreneur salarié dans une SCOP (Société COOPérative) : MFconsulting
  - conseil en python
  - rédaction de dossier de financement de l'innovation
  - formations
- créateur de *Oh Ce Cours Formation*
- cofondateur / CTO de ExcellencePriority (site de partage exclusif de petites annonces orienté luxe)
- coorganisateur de meetups à Lille
  - python
  - big data et machine learning

Où me trouver ?

- mail: matthieu@falce.net
- github : ice3
- twitter : @matthieufalce
- site: falce.net

Devops

Définition

> Le devops - ou DevOps (selon la graphie habituellement utilisée en langue anglaise) - est un mouvement en ingénierie informatique et une pratique technique visant à l'unification du développement logiciel (dev) et de l'administration des infrastructures informatiques (ops), notamment l'administration système.
>
> https://fr.wikipedia.org/wiki/Devops

> Le mouvement Devops se caractérise principalement par la promotion de l'automation et du suivi (monitoring) de toutes les étapes de la création d'un logiciel, depuis le développement, l'intégration, les tests, la livraison jusqu'au déploiement, l'exploitation et la maintenance des infrastructures
>
> https://fr.wikipedia.org/wiki/Devops

Séparation ?

Mais ? Ca marche sur ma machine !

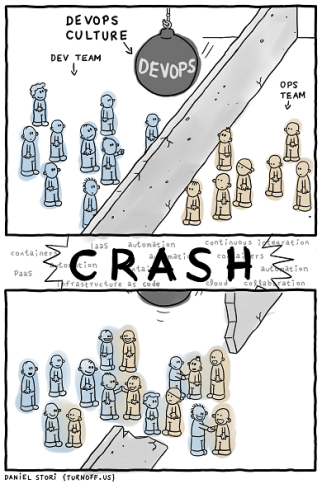
Source : https://aimconsulting.com/insights/blog/devops-architecture-methodoloy-connected/

Pourquoi ?

Séparation dev - ops ?

- les équipes étaient séparées
- spécialisation (compétences différentes)
- "lancer le code au-dessus de la barrière"
- différence d'objectifs
  - ops : stabilité et sécurité du système (SLA, qualité, évolution lente)
  - dev : évolution du système (tickets fermés, fonctionnalités, évolution rapide)

Atteindre la cohésion



Source :
hhttp://www.neilmillard.com/2017/08/19/what-is-devops/

Atteindre la cohésion

Sanjeev Sharma et Bernie Coyne recommandent [1] :

- déploiement régulier des applications / fiabiliser le processus
- tester au plus tôt
- tests dans un environnement similaire à celui de production
- intégration continue incluant des tests continus
- *feed-back* rapide des utilisateurs
- métriques et indicateurs clés pour surveiller l'exploitation

1. https://fr.wikipedia.org/wiki/Devops

Devops un métier ?

Pour moi, le DevOps est un concept, une culture plus qu'un métier (comme "l'agile").

Comme métier il est :

- peu défini
- dépendant des entreprises

Une équipe devient DevOps quand tous ses membres le sont.

Outils du DevOps I

Liste non exhaustive :

- gestion de code source
  - git / SVN
  - gitlab / github
- intégration continue (CI) / déploiement continu (CD)
  - Jenkins / Travis
  - Gitlab CI / Github Actions
- conteneurs
  - docker / Kubernetes / MesOS
- cloud providers
  - AWS / GCP
- automatisation et gestion de configuration / Infrastructure as Code (IaC)
  - ansible / salt / Puppet / Chef
  - Terraform / Packer

Outils du DevOps II

- monitoring et alerting
  - Prometheus / Graphana
  - ELK
- outils de gestion de projet
  - Jira
  - Trello
- gestion des secrets de configuration
  - vault / Secrets

Matthieu Falce

Devops
Définition
Séparation
Outils
Infrastructure as Code (IaC)
Ansible

Infrastructure as Code (IaC)

Mécanismes permettant de gérer une infrastructure virtuelle par

- des fichiers descripteurs
- des scripts

S'applique sur la gestion :

- du DNS,
- du Load-Balancing
- des sous-réseaux
- des groupes de sécurité

Infrastructure as Code (IaC) – avantages

- automatisation des déploiements
  - réduction du coût
  - réduction du risque
  - rapidité d'exécution
- reproductibilité des déploiements
- collaboration au sein de l'équipe

Ansible

Matthieu Falce

Devops

Ansible

Contexte
Installation
Gestion des hôtes
Commander aux hôtes
Playbook
Aller plus loin

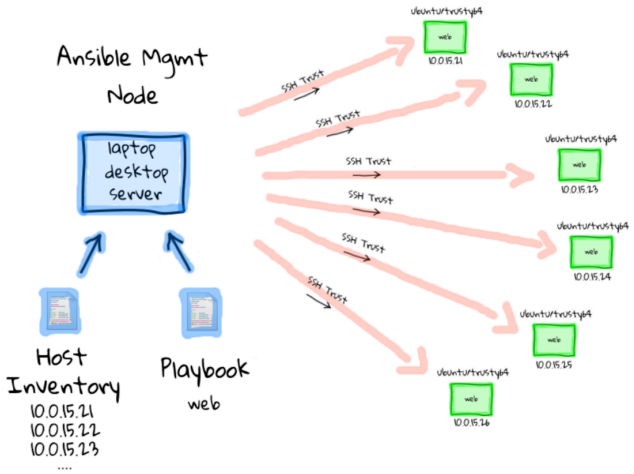
Qu'est-ce que c'est ?

Ansible est une plate-forme logicielle libre pour la configuration et la gestion des ordinateurs. Elle combine le déploiement de logiciels multi-nœuds, l'exécution des tâches ad-hoc, et la gestion de configuration.
Elle gère les différents nœuds à travers SSH et ne nécessite l'installation d'aucun logiciel supplémentaire sur ceux-ci. Les modules communiquent via la sortie standard en notation JSON et peuvent être écrits dans n'importe quel langage de programmation.
Le système utilise YAML pour exprimer des descriptions réutilisables de systèmes, appelées playbook.

---

https://fr.wikipedia.org/wiki/Ansible_(logiciel)

En images



Source : https://sysadmincasts.com/episodes/43-19-minutes-with-ansible-part-1-4

Spécificités

Matthieu Falce

Devops

Ansible

Contexte
Installation
Gestion des hôtes
Commander aux hôtes
Playbook
Aller plus loin

- architecture en mode *push* (*agentless*, sans démon à installer)
- utilise SSH / Paramiko (implémentation python de SSH) pour la communication
- possibilité d'utiliser un mode *pull*
- hybride entre procédural (décrire comment obtenir quelque chose) et déclaratif (décrire quoi obtenir)
- codé en python, utilise YAML pour les fichiers de configuration et Jinja2 pour le templating
- open source et racheté par *Red Hat* en 2015

Ecosystème

Ansible n'est pas le seul logiciel d'automatisation de configuration.

"Concurrents" :

- `puppet`
- `saltstack`
- `chef`

Ansible est le plus simple de tous à prendre en main.

Il n'en reste pas moins puissant et utilisé.

Qui l'utilise ?

- Fedora
- HP
- Airbus
- La Poste
- Société Générale
- EDX
- Cisco
- NASA
- ...

Vocabulaire [2]

- `host` : machine à configurer
- `inventory` : fichier listant des `hosts` à configurer
- `module` : unité de code que l'on peut utiliser depuis des tâches ou en ligne de commande et qui sont exécuté sur les hôtes. `Ansible` est livré avec une grande liste de module.
- `tasks` : dans un `playbook`, nous définissons des tâches, elles vont lancer des modules avec des paramètres spécifiques.
- `role` : fichier décrivant un composant indépendant permettant de réutiliser des étapes de configuration
- `playbook` : fichier décrivant la configuration, le déploiement et l'orchestration à `Ansible` en utilisant des rôles et tâches

2. Plus encore ici : https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html

Installation

Etant *agentless* il suffit de configurer le noeud de contrôle.

- nécessite uniquement python pour fonctionner
  - version 2.7
  - version 3.5 ou supérieure
- disponible sur :
-
  - Linux
  - Unix
  - MacOS
  - pas windows

Le noeud de contrôle peut être n'importe quelle machine ayant accès à celles à déployer :

- serveur dédié
- ordinateurs des développeurs
- ...

Installation – 2

Il est possible d'installer :

- avec les gestionnaires de paquets systèmes
  - Ansible suit un cycle de sortie assez rapide → nécessité d'ajouter des dépots tiers (`rpm` ou `ppa`)
  - regarder les documentations pour ajouter les dépots
  - `sudo dnf install ansible`
  - `sudo yum install ansible`
  - `sudo apt install ansible`
- avec `pip` : `pip install –user ansible`

Avantages de `pip` :

- intégration au requirements.txt de projets python
- ne pas nécessiter les droits administrateurs

Configuration des noeuds contrôlés [3]

Des dépendances sont nécessaires sur les noeuds contrôlés :

- serveur SSH
- serveur SFTP (ou SCP sinon) pour les envois de fichiers
- python 2.6 ou supérieur à 3.5

Il est possible de les configurer en utilisant Ansible et son module `raw` :

```
ansible myhost –become -m raw -a ”yum install -y python2”
```

3. https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#managed-node-requirements

Configuration

Ansible va chercher des fichiers de configurations à ces endroits du plus prioritaire au moins prioritaire :

- chemin défini dans `ANSIBLE_CONFIG` si la variable d'environnement existe
- fichier `ansible.cfg` dans le dossier courant (attention ne doit pas être `world readable`)
- fichier ~ `/.ansible.cfg` dans le dossier `home`
- fichier `/etc/ansible/ansible.cfg`

La configuration par défaut est plutôt saine, mais il peut parfois être nécessaire d'adapter des points à notre façon de travailler.

Bonne pratique : garder toute la configuration dans le dossier de déploiement

Configuration – 2

- installation par gestionnaires de paquets
  - fichier de configuration créé
  - situé dans `/etc/ansible`
- installation par `pip`
  - il faut créer le fichier de configuration
  - exemple ici : `https://github.com/ansible/ansible/blob/devel/examples/ansible.cfg`

Plus d'infos ici : `https://docs.ansible.com/ansible/latest/reference_appendices/config.html#ansible-configuration-settings`

Concepts

Ansible doit connaître des informations sur les hôtes pour les contacter (informations de connexion, identifiants, groupe)
Il est possible de regrouper les hôtes par groupes (par exemple : *les serveurs web de production à Paris*)

Cela se fait à plusieurs endroits :

- l'inventaire (*inventory*) : le fichier le plus important, faisant... l'inventaire des hôtes
- `host_vars` : définissant les variables des hôtes
- `group_vars` : définissant les variables des groupes

Plus d'infos : `https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html`

Inventaire

- on définit l'inventory une fois et on y touche que lorsque l'infrastructure est modifiée
- définit des listes ou des listes de listes (groupes) d'hôtes
- fichier d'inventaire
  - `etc/ansible/hosts` par défaut
  - peut être précisé dans la commande : `-i <path>`
- peut être en format INI ou YAML
- statique, mais possibilité d'inventaire dynamique[4] pour le cloud (par exemple)

4. `https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html`

Inventaire – Exemple

Example de fichier `ini` créant 6 hôtes et 2 groupes

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

Par défaut, 2 groupes créés : `all` et `ungrouped`

Variables – inventaires

On peut passer des variables sur des hôtes directement dans l'inventaire

```
localhost             ansible_connection=local
other1.example.com    ansible_connection=ssh        ansible_user=myuser
other2.example.com    ansible_connection=ssh        ansible_user=myotheruser
```

Variables – inventaires

On peut également passer des variables sur des groupes depuis l'inventaire

```
[atlanta]
host1
host2

[atlanta:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

Variables de groupe et d'hôte [5]

On peut définir des variables pour des hôtes ou des groupes dans des dossiers spéciaux

- `/etc/ansible/group_vars/`
- `/etc/ansible/host_vars`
- dossier `group_vars/` et `host_vars/` dans le dossier du playbook

Il faut mettre des fichiers avec le nom du groupe

- par exemple pour le groupe `webservers`
- le fichier YML `/etc/ansible/group_vars/webservers` (extension `'.yml'`, `'.yaml'`, or `'.json'` optionnelle)

Les variables `group_vars` et `hosts_vars` peuvent aussi être définies dans le dossier du `playbook` ou l'`inventory`

5. https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#group-and-host-variables

Priorités des variables

Dans certains cas, nous pouvons avoir des variables déclarées à plusieurs endroits.

Comment se passe la résolution dans ce cas ?

Endroit de définition moins prioritaire au plus prioritaire (la ligne du dessous écrase ce qui est au-dessus):

- groupe `all` (parent de tous les autres)
- groupe parent
- groupe enfant (dans le cas de plusieurs niveaux de groupes)
- `host`

Plus d'informations ici :
`https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#how-variables-are-merged`

Variables modifiables I

Quelques options que nous pouvons régler pour modifier le comportement d'Ansible lors des connexions / commandes aux hôtes :

- connexion à un hôte :
  - `ansible_connection` (connexion à utiliser : `smart`, `ssh`, `paramiko`)
  - `ansible_host` (le nom de l'hôte à connecter)
  - `ansible_port` (le port du serveur SSH)
  - `ansible_password` (le mot de passe de connexion SSH, à demander à l'exécution ou stocker dans un `Vault`)
- gestion du SSH
  - `ansible_ssh_private_key_file` (chemin de la clé SSH privée à utiliser)
  - `ansible_ssh_common_args` (paramètres à passer aux connexions SSH, SFTP, SCP, ...)
- escalade de privilèges

Variables modifiables II

- `ansible_become_user` (l'utilisateur ayant les droits d'administrateur)
- `ansible_become_password` (le mot de passe administrateur, à demander à l'exécution ou stocker dans un `Vault`)

- paramètres d'environnement distant
  - `ansible_python_interpreter` (le chemin vers l'interpréteur python distant)
- ...

Il est également possible de créer ses propres variables que l'on utilisera dans des playbooks / rôles.
Plus d'informations ici : `https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#connecting-to-hosts-behavioral-inventory-parameters`

Organisation de l'inventaire

Plutot que de créer des groupes de groupes, d'autres techniques existent

- un inventaire par environnement (préprod, prod, testing...)
- grouper par localisation géographique (datacenter Paris, datacenter Gravelines)
- grouper par fonction de serveur (web, base de données, analytics)

Comment redémarrer les machines qui sont :

- des serveurs web à Paris en prod ?
- toute la préprod ?

Concepts

Ansible dispose d'un ensemble de modules (le plus souvent *idempotents*) configurables qui sont des unités de codes que l'on peut appeler en ligne de commande ou depuis un playbook.

Exemples :

- `copy` : permet de copier un fichier sur l'hôte
- `service` : pour gérer des deamons système (initV ou systemD)
- `ping` : permet... d'effectuer un ping
- ...

Et beaucoup, beaucoup d'autres : `https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html`

Commander aux `hosts`

Il existe 2 façons de commander à un `host` :

- par une commande *ad hoc* : qui n'exécute qu'une chose
- par un `playbook` qui va regrouper plusieurs commandes (ou tâches) à effectuer

Appeler un module depuis la ligne de commande

Concept :
`ansible [cible] -m [module] -a "[parametres du module]"`

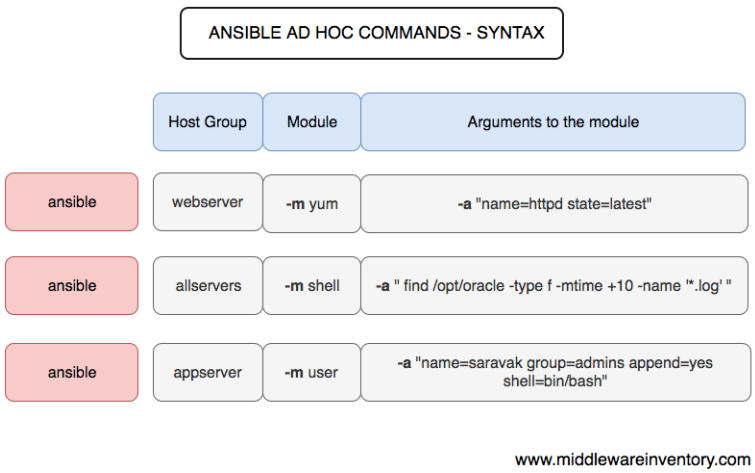
Exemples :

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

La cible désigne des hôtes ou groupes définis dans l'inventaire

Appeler un module depuis la ligne de commande

Décomposition d'une commande



Composition des paramètres d'une commande (source : https://www.middlewareinventory.com/blog/ansible-ad-hoc-commands/)

Changement

Les commandes / tâches Ansible cherchent à être *idemptotente*.
L'idemptotence implique que lancer une tâche plusieurs fois se passe sans effet de bord.

Les commandes peuvent avoir 2 statuts de succès :

- `changed` : une action a été effectuée et quelque chose a changé
- `OK` : une action a été effectuée et rien n'a été modifié

Dans certains cas on peut carrément sauter des actions après des `OK` car on sait quelles ont déjà été faites.

Par exemple : arrêter un déploiement si le code est déjà à jour.

Escalade de privilège [6]

Certaines commandes vont nécessiter d'avoir des droits d'administration ou celui d'un autre utilisateur.

Pour cela, Ansible utilise le concept de `become`. Ainsi nous devenons un autre utilisateur (mécanisme utilisant selon la plateforme : `sudo`, `su`, `pfexec`, `doas`, `pbrun`, `dzdo`, `ksu`, `runas`, `machinectl`, ...)

Variables à régler :

- `become` / `ansible_become` (booléen) : si nous utilisons le changement ou pas
- `become_user` / `ansible_become_user` : utilisateur à incarner (par défaut `root`)

6. https://docs.ansible.com/ansible/latest/user_guide/become.html

Matthieu Falce
Devops
Ansible
Contexte
Installation
Gestion des hôtes
Commander aux hôtes
Playbook
Aller plus loin

Escalade de privilèges – exemple

Sans escalade de privilèges :

```
- name: Run a command as the apache user
  command: somecommand
```

Avec escalade de privilèges

```
- name: Run a command as the apache user
  command: somecommand
  become: yes
  become_user: apache
```

Escalade de privilèges – risques

`become` peut causer des problèmes :

- si l'on devient un utilisateur avec moins de droits que l'utilisateur connecté → possibilité de ne plus pouvoir lire le code à exécuter
  - utiliser le *pipelining*
  - ne pas devenir un utilisateur avec moins de droits
- il faut passer le mot de passe du compte sans le stocker
  - utiliser `Vault`
  - utiliser le paramètre `-K` ou `–ask-become-password`
- peut être gênant dans certaines configurations
  - si seulement certaines commandes sont sur liste blanche pour `sudo` (on ne sait pas les chemins qu'Ansible va utiliser)
  - faire attention avec les variables définies par `pam_systemd` (par exemple : `XDG_RUNTIME_DIR`)

Facts

Ansible permet de récolter des informations sur les `hosts` : les facts

- appelé en commande `ansible all -m setup`
- appelé par défaut au début d'un playbook

Passage de variables

Nous pouvons rajouter des variables en ligne de commande avec `-e` ou `–extra-vars`:

- au format `ini` (clé=valeur) : `-e "version=1.23.45 other_variable=foo"`
- au format `json` : `-e '"pacman":"mrs","ghosts":["inky","pinky","clyde","sue"]'`
- depuis un fichier (`yaml` ou `json`) : `-e "@some_file.json"`

Priorité des variables I

Vu que l'on peut définir des variables à de plusieurs endroits, il faut regarder la priorité en cas de définitions multiples. Voilà l'ordre, du moins prioritaire au plus prioritaire

- command line values (eg "-u user")
- role defaults
- inventory file or script group vars
- inventory group_vars/all
- playbook group_vars/all
- inventory group_vars/*
- playbook group_vars/*
- inventory file or script host vars
- inventory host_vars/*
- playbook host_vars/*

Priorité des variables II

- host facts / cached set_facts
- play vars
- play vars_prompt
- play vars_files
- role vars (defined in role/vars/main.yml)
- block vars (only for tasks in block)
- task vars (only for the task)
- include_vars
- set_facts / registered vars
- role (and include_role) params
- include params
- extra vars (always win precedence)

Patterns I

Les `patterns` ou motifs permettent de définir les cibles à manipuler.
Ils peuvent toucher les :

- groupes
- hôtes

Patterns II

Quelques exemples :

- tous les hôtes : `all` / `*`
- un hôte préçis : `host1`
- plusieurs hôtes : `host1,host2`
- un groupe : `webservers`
- fusion de groupes : `webservers:dbservers`
- exclusion de groupes : `webservers:!atlanta`
- intersection de groupes : `webservers:&staging`

Patterns III

Pour les playbooks :

- depuis le `host` d'un playbook
- depuis la ligne de commande avec `–limit`

Patterns IV

On peut également savoir lesquels seront considérés depuis la `CLI` : `–list-hosts` (liste les hôtes considérés par le `pattern`)

Intéret

Les playbooks jouent un rôle central dans Ansible.

Ils permettent de définir une configuration comme un ensemble de tâche à effectuer.

En pratique, vous allez en utiliser 99% du temps.

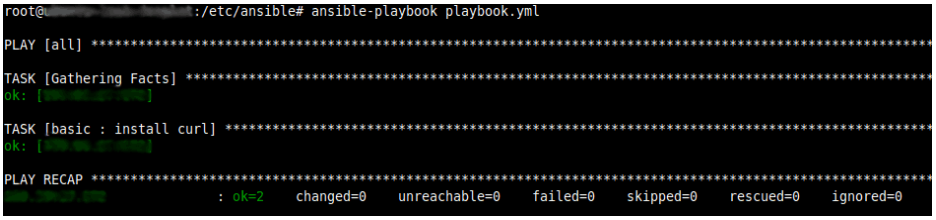
Exemple de playbook


```
---
- hosts: all
  gather_facts: True
  vars:
    upgrade_type: safe
  tasks:
    - name: Upgrade packages
      apt: upgrade={{ upgrade_type }}

    - name: Install base packages
      apt:
        name: # with_items est déprécié
          - fail2ban
          - iptables-persistent
        state: present
        update_cache: yes
```

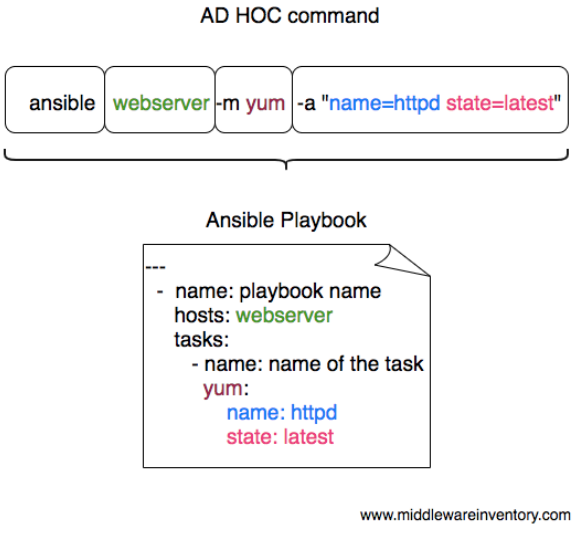
Lancer un playbook

```
ansible-playbook playbook.yml
```



Résultat de l'exécution du playbook (source : https://sysadmincasts.com/episodes/43-19-minutes-with-ansible-part-1-4)

Gestion des paramètres et liens avec les commandes *ad hoc*



Lien des paramètres entre un playbook et une commande *ad hoc* (source : https://www.middlewareinventory.com/blog/ansible-ad-hoc-commands/)

Matthieu Falce

Devops

Ansible

Contexte
Installation
Gestion des hôtes
Commander aux hôtes
Playbook
Aller plus loin

YAML

Le YAML est le format utilisé pour écrire les playbooks (entre autre) dans Ansible.
Concept :

- acronyme récursif depuis sa version 1.12
  - *YAML Ain't Markup Language*
  - « YAML n'est pas un langage de balisage »
- format de représentation de données par sérialisation Unicode
- même concept que JSON, XML, CSV...

Plus d'informations ici : https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html

YAML

Le YAML est le format utilisé pour écrire les playbooks (entre autre) dans Ansible.
Ce que l'on peut stocker :

- des scalaires : valeurs simples
- des listes : plusieurs éléments ordonnés
- des dictionnaires : relation entre une clé et une valeur
- → des mélanges complexes des trois
- des commentaires

Plus d'informations ici : https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html

YAML – exemple

Exemple de liste de dictionnaire qui contiennent des valeurs complexes

```
---
# liste (avec le "-"" et même niveau indentation)
-  martin: # dictionnaire
    name: Martin Dev # clé : valeur
    job: Dev
    skills: # liste avec le -
      - python
-  tabitha:
    name: Tabitha
    job: Dev
    skills:
      - lisp
      - fortran
```

YAML – exemple

Représentation sous forme de *flow collection*

```
---
- martin: {name: Martin D, job: Dev, skill: [python]}
- tabitha: {name: Tabitha, job: Dev, skill: [lisp, fortran]}
```

YAML – Gotchas

Quelques sources d'erreurs :

- il faut échapper correctement les scalaires avec des guillemets
  - si la valeur fini par un ":" (par exemple `c:`)
  - si la valeur commence par "{{ }}" (si c'est une variable)

Templates / Jinja2

Les templates sont utilisés pour les *expressions dynamiques* et *l'accès aux variables* dans

- les `playbooks`
- les fichiers copiés avec le module `copy`

Le moteur de *template* est `Jinja2`, qui permet :

- d'accéder à des variables et les modifier avec des filtres
  - il y a beaucoup de filtres possibles : `https://docs.ansible.com/ansible/latest/user_guide/_filters.html`
- de faire des boucles
- de faire des tests
- ...

Plus d'informations ici : `https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html`

Variables

Les variables peuvent être utilisées à plusieurs endroits :

- tâches / `roles` / `playbooks`
- *templates*

`Ansible` fait une évaluation paresseuse des variables : elles ne sont calculées qu'au moment de l'usage → une variable peut en contenir d'autres

Prompt

Dans certains cas nous voulons demander à l'utilisateur des variables à l'exécution du `playbooks`. On utilise un `prompt` pour cela.

```
---
- hosts: all
  vars_prompt:
    - name: username
      prompt: "What is your username?"
      private: no
    - name: password
      prompt: "What is your password?"
  tasks:
    - debug:
        msg: 'Logging in as {{ username }}'
```

Prompt – remarques

- si les variables sont déjà définies, `Ansible` ne les demandera pas
- on peut écrire des valeurs secrètes avec `private: yes`, on peut également utiliser des variables `Vault`
- faire attention à quelles variables demander pour avoir une traçabilité des déploiements

Vault

Le `Vault` permet de stocker des variables de façon sécurisée. C'est utile pour les secrets (mots de passe, clé d'API, ...) que l'on ne veut pas stocker en clair.
Il y a deux façons de l'utiliser :

- chiffrer directement une variable :
  - permet de faciliter les `diffs` (on voit que la variable a changé)
  - ne permet pas facilement de changer de mot de passe (toutes les variables sont chiffrées séparément)
- chiffrer un fichier (contenant plusieurs variables)
  - permet de changer facilement de mot de passe
  - plus difficile de gérer les `diffs` (le nom de la variable est chiffré également)

Plus d'informations ici : `https://docs.ansible.com/ansible/latest/user_guide/vault.html`

Vault – chiffrer un fichier I

1. création du fichier de variables : `ansible-vault create secrets.yml`
   - entrer le mot de passe utilisé pour le déchiffrement
   - éditer le fichier `secrets.yml` avec les variables et leur valeur
2. éditions suivantes : `ansible-vault edit secrets.yml`
3. passage des variables au `playbook`
   - passage par la CLI `ansible-playbook playbook.yml -e@secrets.yml`
   - dans le `playbook` en utilisant `include_vars: secrets.yml`
4. déchiffrement (on utilise la même clé que pour chiffrer)
   - demander à l'utilisateur `–ask-vault-pass`
   - utiliser le mot de passe écrit dans un fichier `–vault-password-file` (ou changer la variable `DEFAULT_VAULT_PASSWORD_FILE`)

Vault – chiffrer une variable

```
ansible-vault encrypt_string 'foobar' –name
'the_secret'
```

Résultat :

```
the_secret: !vault |
        $ANSIBLE_VAULT;1.1;AES256
        30313332353763383265656130396566626438 36
        34373764343465623937303338303835366138 35
        62666431353932616164643664613563303031 30
        316331330a6465356538653332333663306233 37
        30666533363564653837396362656130316430 37
        38383564306335316364646663337663735346 3
        34363737393432313 70a65333531656564386337
        31626163373530333833353234383337306337 32
        6235
```

Matthieu Falce
Devops
Ansible
Contexte
Installation
Gestion des hôtes
Commander aux hôtes
Playbook
Aller plus loin

Debugging

Parfois, nous avons des problèmes avec nos `playbooks`, nous voulons voir ce qu'ils vont exécuter ou quelle est la valeur de certaines variables.

`Ansible` nous fourni des aides :

- `dry run`
- `–step` : permet de lancer un `playbook` "pas à pas", pour chaque tâche, on nous demande si on veut l'exécuter
- la commande `debug`
- le mot clé `debugger` (exécution pas à pas) [7]

7. https://docs.ansible.com/ansible/latest/user_guide/playbooks_debugger.html

Debugging – `dry run`

Le `check mode` ou `dry run` permet

- de ne pas réellement faire de modifications à la machine distante.
- de montrer les changements qui auraient été faits

Intéressant de l'utiliser avec `–diff` pour les changements qui auraient été faits

```
ansible-playbook foo.yml –check –diff –limit
foo.example.com
```

Debugging – Debug


```
# affiche une variable
- debug:
    msg: >
      System {{ inventory_hostname }}
      has uuid {{ ansible_product_uuid }}

# Affiche le résultat de la dernière commande
- shell: /usr/bin/uptime
  register: result

- debug:
    var: result
    verbosity: 2
```

Gestion des erreurs

Matthieu Falce

Devops

Ansible

Contexte
Installation
Gestion des hôtes
Commander aux hôtes
Playbook
Aller plus loin

Par défaut `Ansible` s'arrête à la première erreur rencontrée. Dans certains cas, on doit cependant changer ce comportement.

Voilà des possibilités :

- `ignore_errors: yes` : l'erreur ne compte pas
- `failed_when` : permet de changer la définition d'une erreur (selon la strderr / le code de retour)
- `changed_when` : permet de changer la définition d'un changement (selon stderr / code de retour)
- `any_errors_fatal: true` : permet d'arrêter complètement le `play` en cours. Sans cela, le `play` ne s'arrête que sur l'hôte ayant une erreur

Plus d'informations : https://docs.ansible.com/ansible/latest/user_guide/playbooks_error_handling.html

Gestion des erreurs – `failed_when`


```
- name: Check if a file exists in temp and fail task if it does
  command: ls /tmp/this_should_not_be_here
  register: result
  failed_when:
    - result.rc == 0
    - '"No such" not in result.stdout'

- name: example of many failed_when conditions with OR
  shell: "./myBinary"
  register: ret
  failed_when: >
    ("No such file or directory" in ret.stdout) or
    (ret.stderr != '') or
    (ret.rc == 10)
```

Ne pas exécuter toutes les tâches

Nous pouvons inclure / exclure certaines tâches de nos `playbooks`

- depuis un playbook
  - when : n'exécute la tâche que si la condition est remplie (soit sur une variable, le résultat d'une tâche précédente ou un fact)
  - `tag` : n'exécute la tâche que si le tag est demandé avec `–tags` ou explicitement ignoré avec `–skip-tags` (utile pour les playbooks très gros)
- depuis la ligne de commande
  - `–start-at-task` : on précise le nom d'une tâche et ça lance à partir d'elle

On peut vérifier les tâches qui vont tourner avec `–list-tasks`

Tags

```
tasks:
- yum:
    name:
    - httpd
    - memcached
    state: present
  tags:
  - packages

- template:
    src: templates/src.j2
    dest: /etc/foo.conf
  tags:
  - configuration
```

Plus d'informations ici : https://docs.ansible.com/ansible/latest/user_guide/playbooks_tags.html

Handlers

Les `handlers` permettent d'exécuter une tâche quand nécessaire → exécution conditionnelle à certaines actions.

Plusieurs actions peuvent notifier un `handlers`

- qui ne sera déclenché qu'une fois
- à la fin du bloc de tâches
- si l'action a été effectivement effectuée

Exemple d'utilisation

- recharger un démon quand un fichier de configuration a changé

Plus d'informations : https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html#handlers-running-operations-on-change

Matthieu Falce

Devops

Ansible

Contexte
Installation
Gestion des hôtes
Commander aux hôtes
Playbook
Aller plus loin

Handlers – exemples

Tâche :

```
- name: template configuration file
  template:
    src: template.j2
    dest: /etc/foo.conf
  notify:
     - restart memcached
     - restart apache
```

Section handler :

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

Les services ne seront redémarrés que si le fichier est copié.

Handlers – gestion

Les `handlers` peuvent s'abonner à des sujets de messages.

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

tasks:
    - name: restart everything
      command: echo "this task will restart the web services"
      notify: "restart web services"
```

Organisation

Pour l'instant, nous n'avons vu que des `playbooks` avec des `tasks`.

Cela n'est pas très pratique pour la maintenabilité et la documentation du *code*.

- comment répéter plusieurs actions en changeant 1 seul paramètre ?
- comment ré-utiliser du code partagé ?
- ...

Organisation

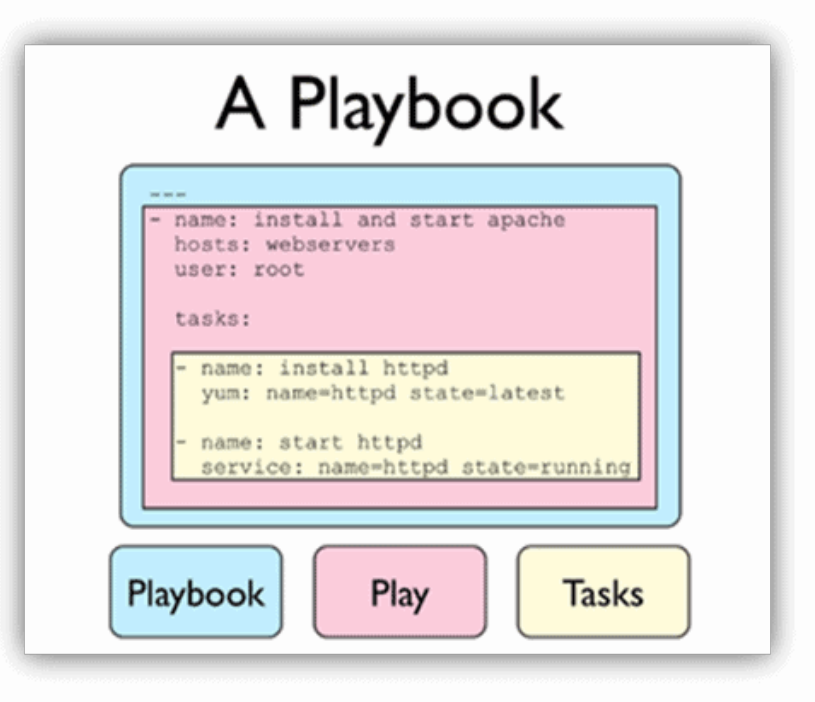
Il existe 2 moyens d'alléger un `playbook` :

- les `includes`
- les `roles`

La plupart du temps, vous utiliserez des `roles`.

Plus d'informations ici : `https://docs.ansible.com/ansible/2.3/playbooks_roles.html`

Organisation – Composition d'un playbook



Décomposition d'un `playbook` (source : `https://k21academy.com/devops-foundation/ansible-playbook-galaxy-tower/`)

Organisation – `include`

Matthieu Falce

Devops

Ansible

Contexte
Installation
Gestion des hôtes
Commander aux hôtes
Playbook
Aller plus loin

Les includes permettent d'importer un fichier `YAML` de tâches depuis un playbook.
`task includes` et `play includes`
Les `playbooks` ont un mot clé `include` mais la liste des tâches aussi.

- les `play includes` : ne fonctionnent que pour importer des `playbook` et seulement à un niveau
- les `task includes` : peuvent prendre des paramètres mais ne sont qu'une liste de tâches

Les `play includes` sont plutôt limités.

Organisation – include

Voilà à quoi ressemblent des includes :

```
# this is a 'play' include
- include: listofplays

- name: another play
  hosts: all
  tasks:
    - debug: msg=hello

    # this is a 'task' include
    - include: stuff.yml
```

Organisation – include

Voilà à quoi ressemble des includes de tâches :

`Playbook` principal :

```
tasks:
  - include: tasks/foo.yml
```

Liste de tâches (`tasks/foo.yml`)

```
tasks:
  - include: tasks/foo.yml
```

Organisation – include

Avec les includes il est alors possible de réutiliser du code :

```
tasks:
  - include: wordpress.yml wp_user=timmy
  - include: wordpress.yml wp_user=alice
  - include: wordpress.yml wp_user=bob
```

Organisation – role

Les `roles` sont une automatisation pour faciliter les imports (fichiers de variables, `handlers`...)

- ils sont basés sur une architecture de dossier spécifique
- ils permettent de charger tout ce qui est lié à certaines actions
- ils permettent de documenter le type d'actions effectuées
- ils peuvent avoir des dépendances entre eux

Organisation – role


```
site.yml
webservers.yml
fooservers.yml
roles/
   common/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
   webservers/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
```

Organisation – role


```
---
- hosts: webservers
  roles:
      - common
      - webservers
```

Organisation – role

Nous pouvons aussi passer des paramètres spécifiques aux rôles :

```
---

- hosts: webservers
  roles:
    - common
    - role: webserver
      dir: '/opt/a'
      app_port: 5000
    - role: webserver
      dir: '/opt/b'
      app_port: 5001
```

Taches pré et post

Il est possible d'exécuter du code avant et après les rôles.
Pour cela, on utilise les `pre_task` et `post_tasks`.
Voilà l'ordre d'exécution :

- `pre_task` définies dans le `play`
- `handler` déclenchés jusqu'ici
- les rôles vont s'exécuter. Les dépendances seront exécutées avant si besoin
- les tâches définies dans le `playbook`
- `handler` déclenchés jusqu'ici
- `post_task` définies dans le `play`
- `handler` déclenchés jusqu'ici

Taches pré et post


```
---
# source : https://subscription.packtpub.com/
# book/networking_and_servers/9781784398293/2/
# ch02lvl1sec22/adding-pre-tasks-and-post-tasks-to-playbooks
- hosts: www
  remote_user: vagrant
  sudo: yes
  pre_tasks:
     - shell: echo 'I":" Beginning to configure web server..'
  roles:
     - nginx
  post_tasks:
     - shell: echo 'I":" Done configuring nginx web server...'
```

Hôtes multiples

Matthieu Falce

Devops

Ansible

Contexte
Installation
Gestion des hôtes
Commander aux hôtes
Playbook
Aller plus loin

Ansible gère des déploiements sur plusieurs machines. Quand une machine n'est pas atteignable elle est marquée `unreachable` et ses tâches ne sont plus lancées.

- par défaut
  - chaque tâche est lancée en parallèle, les hôtes les plus rapides attendant les autres
  - 5 d'hôtes sont lancés en parallèle (réglable avec l'option `-f` ou en dans le fichier de config [8])
- avec la `strategy` : `free` [9]
  - les hôtes ne s'attendent plus les uns les autres
- avec le mot clé `serial` [10] on peut limiter le nombre de machines déployées en parallèle à la fois
  - utile pour la haute disponibilité (HA)

8. https://docs.ansible.com/ansible/latest/user_guide/playbooks_strategies.html#setting-the-number-of-forks
9. https://docs.ansible.com/ansible/latest/plugins/strategy/free.html#free-strategy
10. https://docs.ansible.com/ansible/latest/user_guide/playbooks_delegation.html#rolling-update-batch-size

Ansible Galaxy

Galaxy est un *hub* permettant de partager des rôles Ansible

Disponible ici : https://galaxy.ansible.com

- les problèmes sont souvent les mêmes entre les devops (installer nginx, wordpress, mettre un serveur à l'heure...)
- il vaut mieux avoir un bon rôle d'installation que des personnels bricolés
- permet d'aller plus vite et de gagner en qualité

N'importe qui peut uploader sur Galaxy, il faut donc faire attention à la qualité.

Plus d'informations ici : https://docs.ansible.com/ansible/latest/galaxy/user_guide.html

Galaxy – Installer un rôle

```
ansible-galaxy install namespace.role_name
```

Par défaut le rôle sera installé dans un de ces fichiers :

- ~/.ansible/roles
- /usr/share/ansible/roles
- /etc/ansible/roles

Il est également possible d'installer plusieurs rôles à la fois, en utilisant un fichier `requirements.yml`.

Galaxy – Utiliser un rôle


```
- hosts: all
  become: yes
  become_method: sudo
  # Liste des rôles à installer
  roles:
  - geerlingguy.apache
  - geerlingguy.php-versions
  - geerlingguy.php
  - geerlingguy.mysql
  - geerlingguy.php-mysql
  - oefenweb.wordpress
  - oefenweb.fail2ban
```

Galaxy – Utiliser un rôle

Matthieu Falce

Devops

Ansible

Contexte
Installation
Gestion des hôtes
Commander aux hôtes
Playbook
Aller plus loin

Il faut cependant gérer les variables qui peuvent être nécessaires.

Par exemple, pour le rôle `Wordpress` plusieurs sont requises : `https://galaxy.ansible.com/oefenweb/wordpress`

Comment les renseigner ?

- soit dans les `vars` du playbook
- soit dans `vars_files` ou `include_vars` dans le playbook

Galaxy – Utiliser un rôle

Passage de variables dans `vars_files`

```
- hosts: all
  become: yes
  become_method: sudo
  # Liste des rôles à installer
  roles:
  - oefenweb.wordpress
  vars_files:
    - main.yml
```

Source : https://les-enovateurs.com/ansible-galaxy-scripts-existants/

Galaxy – Cycle de vie

- recherche d'un rôle : `ansible-galaxy search elasticsearch –author geerlingguy`
- lister ceux installés : `ansible-galaxy list`
- supprimer un rôle : `ansible-galaxy remove namespace.role_name`

Ansible Tower[11] / AWX

API, un service Web et une console Web permettant de centraliser Ansible

- interface graphique pour Ansible
- affichage des jobs en temps réel
- gestion de déploiement complexes
- droits d'accès aux déploiements
- utile pour la collaboration quand beaucoup de serveurs / déploiements / membres d'équipe

Tower est commercialisé par RedHat et AWX est sa version béta gratuite maintenue par RedHat (sans garanties)

11. https://www.ansible.com/products/tower

Et sous Windows ?[12]

C'est possible !

- utilise WinRM pour la communication
- possibilité expérimentale d'utiliser un serveur SSH sous Windows
- la plupart des modules existants fonctionnent et certains sont spécifiques à Windows

12. https://docs.ansible.com/ansible/latest/user_guide/windows_setup.html

Lookups [13]

Les lookups permettent de récupérer des variables depuis des sources dynamiques :

- url
- CSV / TSV
- AWS
- fichier
- password

Cela permet d'avoir des comportements dynamiques (attention à la reproductibilité)

13. https://docs.ansible.com/ansible/latest/plugins/lookup.html

Exemple lookups

Exemple d'une génération de mot de passe dynamique :

```
---
  - name: how to use password lookup
  hosts: all
  vars:
    # génère un mot de passe sans le sauvegarder sans
    # créer de fichier et l'affiche
    password: "{{ lookup('password', '/dev/null chars=digits') }}"
  tasks:
    - name: show password file content
      debug:
        msg: "Le mot de passe est : {{ password }}"
```

Source : https://d3vpasha.wordpress.com/2017/06/28/ansible-lookups-acceder-a-des-donnees-depuis-des-sources-exterieures/

Etendre Ansible [14]

Il y a 2 façons d'étendre le code d'Ansible

- avec des modules (langage que l'on veut)
  - scripts réutilisation et indépendants
  - utilisés par l'API, les commandes ou `playbooks`
  - s'exécutent sur le système cible
- avec des plugins (en python)
  - augmentent les fonctionnalités du cœur d'`Ansible`
  - permettent de modifier les fonctionnalités cœur (transformer des données, logger une sortie, manipuler l'inventaire...)
  - s'exécutent sur la machine de commande

14. https://docs.ansible.com/ansible/latest/dev_guide/developing_locally.html

Ecrire ses modules

Les modules peuvent être écrits dans n'importe quel langage, ils prennent un JSON en entrée et en renvoient un en sortie.

- il faut cloner le code d'`Ansible` sur sa machine (python)
- installer et activer l'environnement virtuel
- créer le module `mon_test` en créant le fichier `mon_test.py` au bon endroit
- on peut appeler ce module depuis un `playbook` ou une commande par le nom du module crée

Plus d'informations : https://docs.ansible.com/ansible/latest/dev_guide/developing_modules_general.html

Ecrire ses modules

Code python minimal d'un module :

```python
#!/usr/bin/python

from ansible.module_utils.basic import *


def main():

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


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

Playbook l'appelant :

```
- hosts: localhost
  tasks:
    - name: Test that my module works
      github_repo:
      register: result

    - debug: var=result
```

Tester ses playbooks I

`Molecule` est un outil en python, maintenu par `Ansible` permettant de tester les rôles `Ansible`.
Il permet :

- de tester des instances multiples
- sur des systèmes de virtualisation différents (`Docker`, `Vagrant`…)
- avec des frameworks de tests différents (Test infra, GOSS…)

Il va lancer une séquence d'actions sur la machine :

- `lint` : vérifie que les fichiers YAML sont correctement formatés, selon les *best practices*
- `dependency` : installe les dépendances Ansible-galaxy
- `syntax` : vérifie la syntaxe des rôles `Ansible`
- `create` : crée la VM
- `prepare` : exécute le playbook `prepare.yml`

Tester ses playbooks II

- `converge` : exécute le playbook `playbook.yml`
- `idempotence` : lance le playbook.yml plusieur fois
- `verify` : vérifie que l'on a un serveur qui répond aux attentes en exécutant un playbook spécifique
- `destroy` : supprime la VM

Plus d'infos : `https://molecule.readthedocs.io`

Bonnes pratiques [15]

- commencer par le faire à la main
- ne pas essayer d'être trop malin / générique
- préférer la clarté et la modularité
- ne pas utiliser trop de variables
- stocker les informations sensibles (mots de passes, clé API, ...) dans des `vaults`

15. https://docs.saltstack.com/en/latest/topics/best_practices.html

Bibliographie I

Une sélection des liens utilisés pour préparer ce cours :

- http://willthames.github.io/2018/07/01/connection-local-vs-delegate_to-localhost.html
- https://symfonycasts.com/screencast/ansible/when-changed
- https://symfonycasts.com/screencast/ansible/idempotency-changed-when
- https://blog.eleven-labs.com/fr/getting-start-with-ansible/
- https://blog.toast38coza.me/custom-ansible-module-hello-world/
- https://utux.fr/index.php?article142/ansible-bonnes-pratiques-ep2
- https://www.codeflow.site/fr/article/how-to-automate-installing-wordpress-on-ubuntu-14-04-using-ansible

Bibliographie II

- https://sysadmincasts.com/episodes/43-19-minutes-with-ansible-part-1-4 (et les 3 autres épisodes)
- https://debugthis.dev/ansible/2019-09-29-running-ansible-on-remote-hosts-without-python/
- savoir ce qu'Ansible pousse sur l'hôte: https://serverfault.com/questions/931073/what-does-ansible-push-to-the-remote-host
- https://www.ansible.com/products/tower
- https://stackoverflow.com/questions/37297249/how-to-store-ansible-become-pass-in-a-vault-and-how-to-use-it/37300030
- https://puppet.com/docs/facter/
- https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html
- https://blog.ineat-group.com/2018/09/tester-ses-playbooks-ansible-localement-avec-vagrant/

Matthieu Falce

Devops

Ansible

Contexte

Installation

Gestion des hôtes

Commander aux hôtes

Playbook

Aller plus loin

Bibliographie III

- https://subscription.packtpub.com/book/virtualization_and_cloud/9781789532937/1/ch01lvl1sec12/-ansible-orchestration-and-automation
- https://www.tartarefr.eu/ansible-par-la-pratique-deuxieme-partie-premiers-playbooks-avec-les-roles/
- https://devopssec.fr/article/roles-ansible
- dépôt officiels des exemples de `playbooks` : https://github.com/ansible/ansible-examples
- https://docs.ansible.com/ansible/2.3/playbooks.html