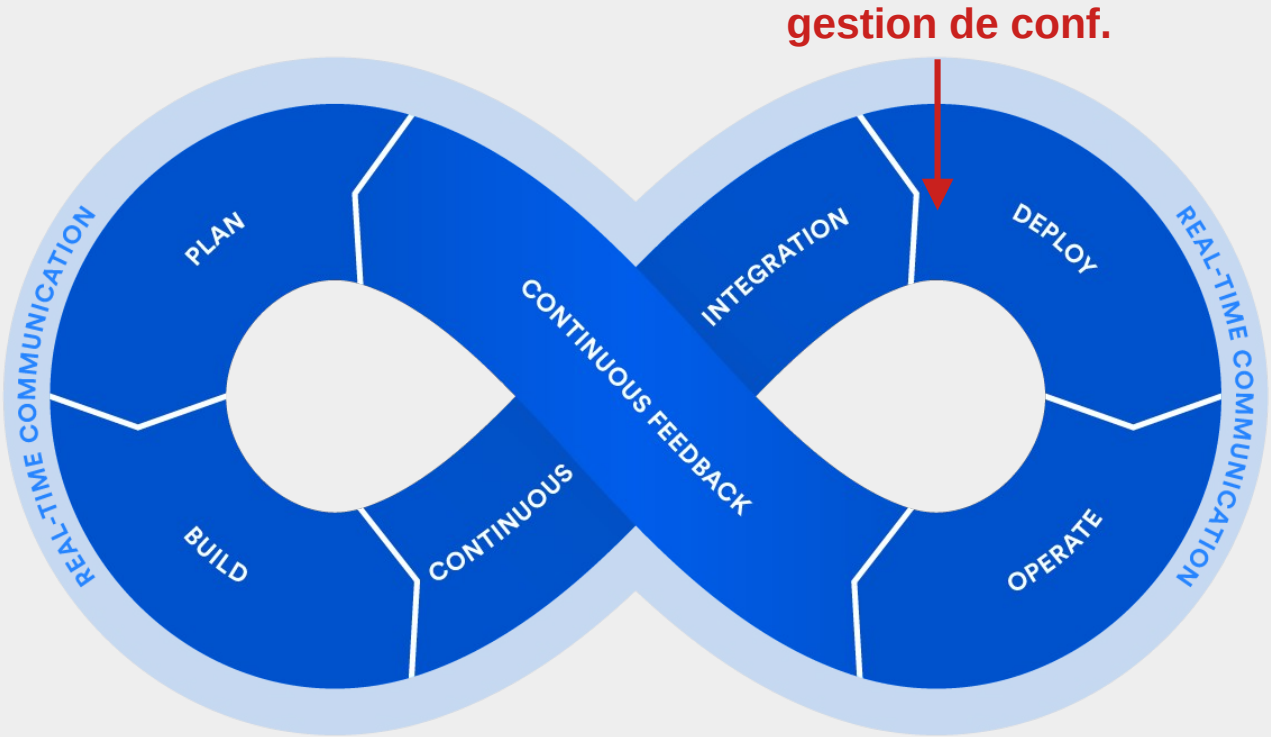
ANSIBLE

INTRODUCTION

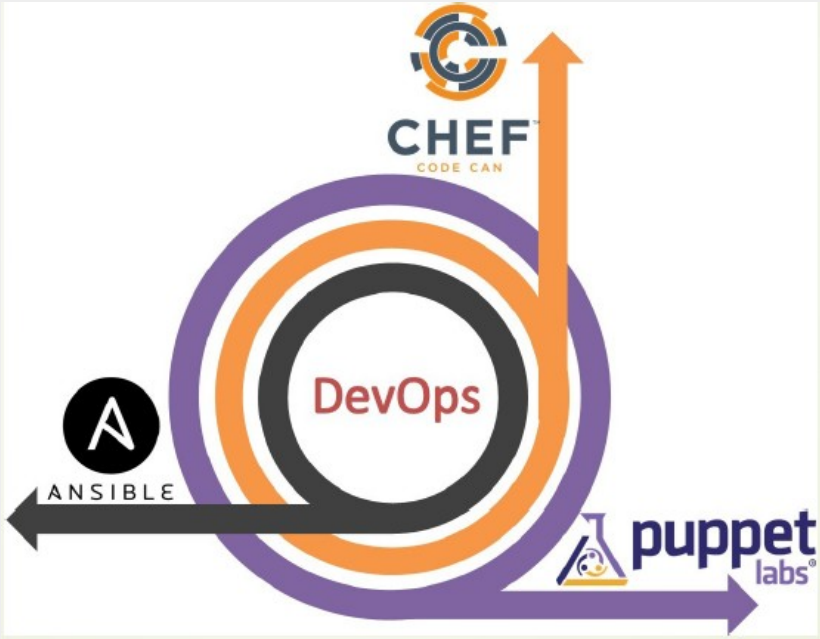
- Solution de gestion de configuration :
  - **processus** de gestion informatique
  - qui vise à assurer la **cohérence**
  - des **actifs informatiques** (logiciels, conteneurs, VMs, serveurs, cluster)

- Objectifs
  - surveiller **automatiquement** les mises à jour des données de configuration pour **préparer** le déploiement
  - La finalité de ce processus de **garantir l'état voulu** du système considéré
  - Préserver l'état voulu après des exécutions successives du processus => **idempotence**

- Position dans le cycle devops

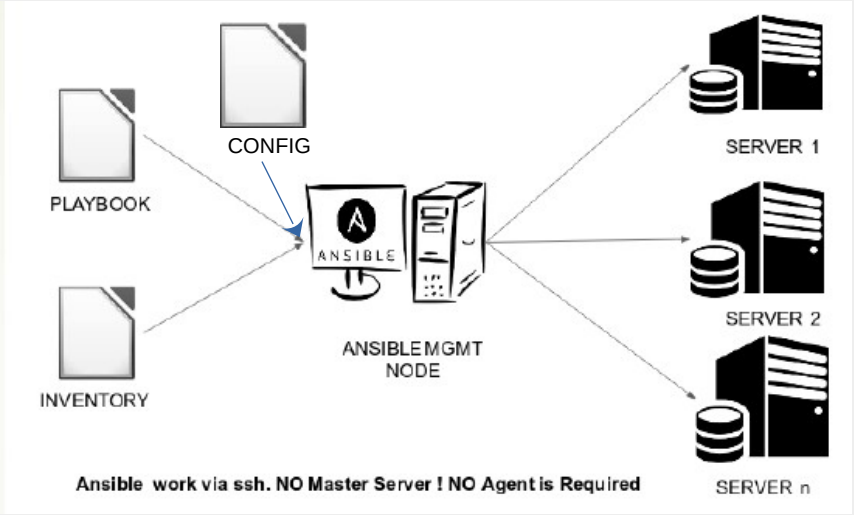


- Position dans le cycle devops
  - Les outils



Mise en oeuvre

- Ansible va automatiser l'administration de tâches
  - exécutées par des processus **python**,
  - décrites dans un document en format YAML « **playbook** »
  - à destination de machines dont les connexions SSH sont décrits dans un **inventaires** .ini ou.yaml
  - en rassemblant des données de contexte dans un fichier « **config.cfg** »



Connexion SSH

- Protocole SSH : « Secure Shell »

  - protocole client / serveur, permet une connexion authentifiée sur un compte **utilisateur** cible
  - par un jeu de **clés privée / publique**

- Création d'une paire de clés côté client : **ssh-keygen**
- Configuration du serveur sshd côté serveur

```
# installation ex debian
sudo apt install -y openssh-server
# configuration dans /etc/ssh/sshd_config
PasswordAuthentication no
PubkeyAuthentication yes
# redémarrer le service sshd
sudo systemctl restart sshd
```

Gérer les clés SSH

- Côté public / serveur

  - ajout du contenu de la clé publique dans le fichier **authorized_keys** du dossier **~/.ssh**
  - le dossier aux droits 700 et le fichier authorized_keys aux droits 600

```
chmod 700 ~/.ssh
chmod 600  ~/.ssh/authorized_keys
```

- Côté privé / client

  - la clé privée aux droits 600 doit être placé dans un dossier aux droits 700

- Test de la connexion sur un compte « ansible » sur le serveur

```
ssh -i /path/to/private_key ansible@host.domain.name
```

Installation d'Ansible via python

- Exemple d'une distribution debian 12

  - python 3.11 est installé par défaut

  - on doit installer le gestionnaire de paquets **pip** via le paquet apt-get **python3-pip**

  - on doit installer ansible dans un **environnement virtuel** python via le paquet apt-get **python-venv**

  - un environnement virtuel est un **dossier** qui va centraliser la **version** et les **dépendances** de python pour un projet donné

```
# install du v. env.
sudo apt-get update
sudo apt-get install -y python3-pip python3-venv
# création du dossier v. env.
python3 -m venv ansible_venv
# activation du v. env.
# => les commandes python3 et pip3 viennent des binaires du dossier v.env.
source /path/to/ansible_venv/bin/activate
# installation d'ansible dans le v. env.
pip3 install ansible
```

Configuration du fichier d'inventaire

- Exemple de fichier « inventory »

```
## définition des machines distantes
# <ip or host>
host.domain.name
# <alias> ansible_host=<ip or host>
staging ansible_host=host.domain.name
preprod ansible_host=preprod.domain.name
prod ansible_host=prod.domain.name

## groupes de machines
[dev]
staging
preprod

[production]
prod

## variables arbitraires attachées à un groupe
[dev:vars]
remote_user=ansible
# exécutable à utiliser sur les machines distantes
ansible_python_interpreter=/usr/bin/python3
```

Tester la connexion ansible à travers ssh

- La commande ansible avec le module « ping »

```
ansible \
--private-key
/path/to/private_key \
-u ansible \
-i  /path/to/inventory \
-m ping \
staging
```

Clé privée pour la cnx ssh

Utilisateur pour la cnx ssh

Fichier contenant les données des machines distantes

Module ping : vérification de la cnx ssh

Nom d'une machine dans l'inventaire

- Configurer La commande ansible avec le fichier « ansible.cfg »

```
# ~/.ansible.cfg
[defaults]
private_key_file = /path/to/private_key
inventory = /path/to/inventory
remote_user = ansible
# bypass de la vérification du fichier "known hosts" pour un script
# automatisé
host_key_checking = False

# section élévation de privilèges : passage à l'utilisateur root
[privilege_escalation]
become_method = sudo
```

- Tester la connexion avec le fichier ansible.cfg

```
ansible -m ping staging
```

commandes ansible « ad-hoc »

- Commandes en ligne avec des modules remarquables

```
ansible -m [module] -a "[module options]" staging
```

- module « **ansible.builtin.raw** », par défaut, pour exécuter une commande à distance

```
ansible  -a "ls -al /home/ansible" staging
```

- module « **ansible.builtin.shell** », pour exécuter une commande à distance dans un shell (redirections, …)

```
ansible -m shell -a 'echo "hello" > /home/ansible/file.txt' staging
```

- module « **ansible.builtin.setup** » pour requêter et afficher les métadonnées « **facts** » de la machine distante

```
ansible -m ansible.builtin.setup staging
```

```
…
"ansible_facts": {
  …
  "ansible_os_family": "Debian",
  …
}
```

Présentation de playbooks

- Document YAML structurant le processus de gestion de configuration

  - utilise les données de connexions configurées dans l'inventaire et le fichier config.cfg

  - séquence les **modules ansible** réalisant les **tâches d'administration** réalisant elles même l'**état voulu** de la machine distante

```
---
# le document commence avec une liste de playbooks
- name: <playbook_name>
  # hosts: noms de machines / groupes de l'inventaire
  hosts: staging
  # compte utilisateur distant
  remote_user: ansible

  # tâches à réaliser sur les machines distantes
  tasks:
    - name: <playbook_name> | <task_name>
      # module à exécuter et ses options
      ansible.builtin.shell:
        cmd: ls -l | grep log
        chdir: somedir/

    - name: <playbook_name> | <task_name2>
```

```
ansible-playbook /path/to/playbook.yml
```

Écriture déclarative & Idempotence des modules ansible

- Écriture déclarative : **présentation** d'une configuration reflétant l'état voulu après la tâche exécutée
- Idempotence : capacité d'un module ansible à garantir l'état quelque soit l'état de départ

  - en particulier, en cas **exécution successives**

- Exemple du module ansible.builtin.file

```
## le module file est IDEMPOTENT i.e si le dossier existe, nul besoin d'exécuter
- name: playbook | add directory
  ansible.builtin.file:
    path: /home/ansible/some_dir
    # directory => mkdir, file => edit, touch => touch, absent => rm (-r),hard => ln, link => ln -s
    state: directory ( | file |touch | absent | hard | link )
    mode: '0755'
    owner: ansible
    group: ansible
```

Etats d’une tâche au moment de l’exécution

- Chaque tâche exécutée

```
PLAY [PLAYBOOK_NAME] ***********************************************************
TASK [Gathering Facts] *********************************************************
ok: [staging]                  ## exécution du module setup par defaut

TASK [PLAYBOOK_NAME | tâche qui ne change pas l’état OU idempotence ]******
ok: [staging]

TASK [PLAYBOOK_NAME | tâche non exécutée programmatiquement ] ***********
skipping: [staging]

TASK [PLAYBOOK_NAME | tâche qui change l’état] *********************************
changed: [staging]

TASK [PLAYBOOK_NAME | tâche en erreur] *****************************************
fatal: [staging]: FAILED! => {"msg": "error message"}
```

Infrastructure As Code

- techniques d'**automatisation** et de **dynamisation** des tâches d'administrations pour réaliser l'état voulu de la cible
- Ces techniques permettent de manipuler des fichiers de configurations à la façon du développement sur les fichiers sources
- Dans le cas d'Ansible, les fonctionnalités d'« IaC » du playbook :
  - **Variabiliser** les contenus en dur dans les tâches
  - **Conditionner** l'exécution d'une tâche selon le contexte « **facts** » de la cible et les variables précitées
  - **Itérer** l'exécution d'une tâche selon des variables
  - **injecter** des fichiers de configuration en dur
  - injecter de **templates** de configuration au format **jinja2** python (variables, conditions, boucles)

Paramétriser un playbook

- On peut injecter de variables à la façon du moteur de templating jinja2 python dans les playbook

```
key: "{{ my_var }}"
```

- La variable « my_var » peut être spécifier dans le playbook

```
- name: <playbook_name>
  hosts: staging
  remote_user: "{{ remote_user }}"
  vars:
    my_var: some thing

  tasks:
    - name:
      …
      key: "{{ my_var }}"
```

- La variable « remote_user » peut être spécifiée dans l'inventaire (cf slide p9) ou dans une variable d'environnement dans la commande ansible_playbook **-e remote_user=ansible**
- En réalité, il n'y a 16 endroits où l'on peut définir des variables (cf infra)

Sauvegarder l'état d'une tâche dans une variable dans un playbook

- La clé « register » dans une tâche permet de créer une variable réutilisable dans les tâches suivantes
- La variable contient un **objet json** présentant **l'état final** après la tâche

```
# stat: informations sur le chemin en paramètre
- name: PLAYBOOK | vérifier si python3.11 est installé
  ansible.builtin.stat:
    path: /usr/bin/python3.11
  register: check_python311

# debug: affichage de la variable register précédente
- name: DEBUG | check_python311 variable value
  debug:
    var: check_python311
...
{"check_python311": {
   "changed": false,        # la tâche ne change pas l'état
   "failed": false,         # la tâche n'est pas en échec
   "stat": {
      ...
      "exists": true,       # information voulue
      …}}
}
```

Conditionner une tâche à l'évaluation de variables

- La clé « when » dans une tâche permet d'évaluer une expression booléenne python
- Cette expression est nourrie par les variables globales, variables register, facts
- La tâche est exécutée si « when » est vrai

```
- name: PLAYBOOK | install python3.11
  ansible.builtin.apt:
    name: python3.11
    state: present
    update_cache: true
  when: not check_python311.stat.exists and ansible_os_family == "Debian"
```

- Itérer une tâche selon une liste de données
  - La clé « loop » dans une tâche contient une liste **d'objets yaml** permet d'exécuter autant de fois que l'on doive qu'il y ait des élements dans la liste
  - La liste peut être une variable globale
  - Les valeurs de la clé « loop » vont être injectées successivement dans les éléments de la tâche elle même en tant que variable locale « item »

```
vars :
  prerequisites :
    - ca-certificates
    - curl

tasks :
  - name: PLAYBOOK | installer des prérequis
    ansible.builtin.apt:
      name: "{{ item }}"
      state: present
      update_cache: true
    loop: "{{ prerequisites }}"
```

Manipuler un fichier de configuration à distance

- Le module « ansible.builtin.file » permet d'injecter un fichier tel quel dans la machine distant
- Le module « ansible.builtin.template », permet d'injecter un fichier .j2 interpolé par des variables
- Ex

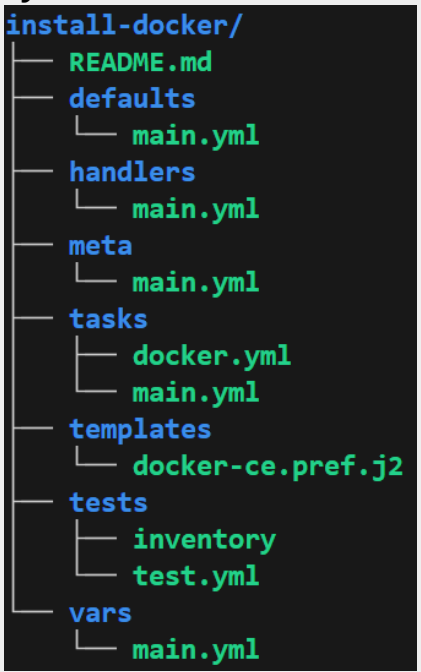
```
# playbook
vars :
  nginx_conf:
    server_name_redirect: formation.lan
    listen: 8080

# default.conf.j2
...
{% if nginx_conf.server_name_redirect is defined %}
server {
    listen       {{ nginx_conf.listen | default('80') }};
    server_name  {{ nginx_conf.server_name_redirect }};
}
{% endif %}
```

```
# default.conf injecté dans la cible
...

server {
    listen       8080;
    server_name  formation.lan;
}
```

- Factoriser un processus donné de gestion de configuration
  - Les rôles « ansible » sont des dossiers composés en
    - tâches
    - variables
    - fichers / templates, ...
  - Présentent une action standard liée à une technologie ou une fonction système
  - Un rôle est créé par la commande **ansible-galaxy role [role_name]**



Composer un playbook

- On peut attacher un autre playbook ou un rôle dans un **playbook principal**

```
- import_playbook: playbook_bootstrap.yml

- name: launch install-docker role
  hosts: dev
  remote_user: "{{ env_user }}"
  roles:
    - name: install-docker

- name: MAIN PLAYBOOK
    hosts: dev
    remote_user: "{{ env_user }}"
    tasks:
      - name: MAIN PLAYBOOK | task
```

Élévation de prilivèges

- La clé « **become** » va demander d'exécuter une tâche donnée en tant que **root**
- La clé « **become_method** » va configurer la méthode du passage à l'utilisateur **root**

  - sudo, su, …
- La commande ansible-playbook principale doit enregister un secret autorisant l'utilisation du become

```
echo  "password" > ./pass
ansible-playbook --become-password-file=pass playbook.yalm

…

- name: BOOTSTRAP | install python3.11 at last
  ansible.builtin.apt:
    name: python3.11
    state: present
    update_cache: true
  when: not check_python311.stat.exists and ansible_os_family == "Debian"
  become: true
```

Surcharge des variables

- Une variable peut être définie dans plusieurs emplacements dans un projet Ansible
- Les emplacements sont du plus fort jusqu'au le plus remplaçable

Variable Precedence v2.0

1. extra vars
2. task vars (only for the task)
3. block vars (only for tasks in block)
4. role and include vars
5. play vars_files
6. play vars_prompt
7. play vars
8. set_facts
9. registered vars
10. host facts
11. playbook host_vars
12. playbook group_vars
13. inventory host_vars
14. inventory group_vars
15. inventory vars
16. role defaults

1. **extra vars** : passable à partir de la command line
4. **role vars** : dans le répertoire /vars d'un rôle
8. **set_facts** : définie dans les set_facts
13. **host_vars** : précisée dans les fichiers individuels des machines de l'inventaire
14. **group_vars** : dans les fichiers de groupes de l'inventaire
16. **role defaults vars** : la variable par défaut