Ansible, automatisation avancée

version 1.0

Sébastien VINCENT

2025

$ /usr/bin/whoami

- Consultant et formateur indépendant
- Github : https://github.com/s-vincent
- E-mail : sebastien@vincent-netsys.fr

$ /usr/local/bin/whoareyou

- Tour de table !
  - Présentation
  - Expériences sur le sujet ?
  - Attentes ?
  - ...

Fonctionnement de la session

- Respecter les horaires
  - du matin, début d'après-midi
  - revenir après les pauses :)
- Signer la feuille de présence
- Concentration / focus
- **Poser des questions**
  - Si les explications ne sont pas claires
  - Pour approfondir
- Donner son avis
- Partager son expérience

Objectifs / prérequis de la session

- Objectifs
  - Aspects avancés d'Ansible
  - Tests unitaires avec Molecule
  - Orchestration
  - Découverte d'Ansible Automation Controller (Tower) / AWX
- Prérequis
  - Connaissance d'Ansible
  - Ecriture de playbooks et de rôles

Citations

- *Apprendre, c'est engranger*
- *Appliquer, c'est comprendre*
- *Documenter, c'est cristalliser*

Rappel : playbook

- Fichier YAML qui décrit un ensemble de tâches à effectuer sur des machines
- Contient variables, cibles, méthodes d'authentification, ...
- A vocation à être réutilisé et stocké dans un gestionnaire de sources (IaC)
- Il doit être idempotent (au maximum) !
- Commande : ansible-playbook

Rappel : paramètres de *play*

- hosts : groupe sur lequel déployer
- vars : variables globales au *play*
- become : élévation de privilèges
- become_user : impersonation d'utilisateur
- tags : ajout de d'étiquettes
- strategy : stratégie de déploiement (linear, free)
- any_errors_fatal : arrêter l'exécution du play si un serveur échoue
- https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html#play

Rappel : répertoires de recherche des fichiers / templates

- ${PWD}/
- ${PWD}/files/
- ${PWD}/templates/
- /path/to/myrole/files/
- /path/to/myrole/templates/

Rappel : tâches

- Action atomique à réaliser
  - Paquet à installer
  - Fichier à modifier
  - Copier un fichier
  - ...
- 99% des cas se base sur un module spécifique
- 1% des cas se base sur les modules shell / command
- Une action peut contenir :
  - module à utiliser + paramètres (exemple : apt, copy, shell)
  - boucles (exemple : liste de paquets à installer)
  - opérateurs de comparaison (exemple : ne faire cette action que si le système est RedHat)
  - variables (exemple : ajout de configuration contenant le nom d'hôte de la machine en cours)

Rappel : paramètres de tâches

- notify : pour les handlers
- when : condition
- loop, with_items, with_nested, with_fileglob : boucles
- vars : variable propre à la tâche
- tags : ajout de d'étiquettes
- become : élévation de privilèges pour la tâche
- become_user : impersonation d'utilisateur pour la tâche
- https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html#tasks

Rappel : inventaire

- Fichier INI, JSON ou YAML qui décrit des groupes
- Un groupe peut contenir soit des machines soit d'autres groupes
- Possibilité d'avoir des plugins d'inventaires dynamiques (AWS, OpenStack, ...)
- Bonnes pratiques : stocker les variables d'infrastructure dans les sous-répertoires group_vars et host_vars
- Commande : ansible-inventory

Rappel : inventaire (2)

```
standalone.example.lan
192.168.0.254

[srvfqdn]
www-sxb.example.lan
db-sxb.example.lan
lweb[1:16].example.lan

[srvip]
192.168.100.[1:32]
192.168.100.[64:96]

[allsrv:children]
srvfqdn
srvip
```

Rappel : ansible.cfg

- Spécifie de nombreux paramètres (inventaire, cache, paramètres systèmes, ...)
- Toutes les variables du fichier de configuration peuvent être utilisé sous la forme d'une variable d'environnement
  - log_path -> ANSIBLE_LOG_PATH
  - inventory -> ANSIBLE_INVENTORY
  - ...
- Commande : ansible-config dump
- https://docs.ansible.com/ansible/latest/reference_appendices/config.html

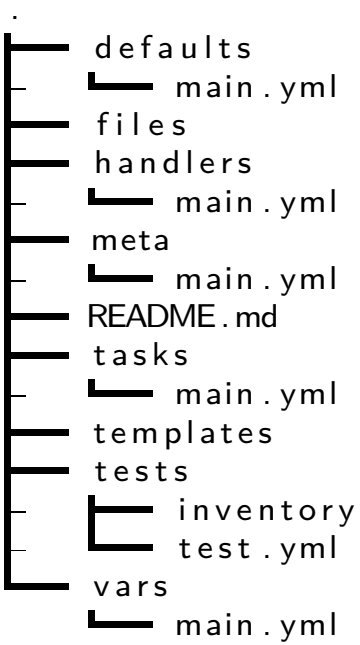
Rappel : ansible.cfg (2)

- Le fichier est pris en compte de la manière suivante :
  - Variable d'environnement ANSIBLE_CONFIG
  - ${PWD}/ansible.cfg (i.e. répertoire courant)
  - ~/.ansible.cfg
  - /etc/ansible/ansible.cfg
- Possible de ne spécifier que quelques options (les autres seront celles par défaut)

Rappel : rôles

- *Playbook* de plus haut niveau afin de viser un objectif fonctionnel
- Tâches, variables, fichiers, notifications propres au rôle
- Il y a un dépôt de rôles Ansible sur https://galaxy.ansible.com
- Commandes :
  - ansible-playbook (pour exécuter)
  - ansible-galaxy (pour créer / récupérer des rôles)

Rappel : rôles (2)



Rappel : variables

- On peut définir des variables dans :
  - un rôle (fichier defaults/main.yml)
  - l'inventaire
  - fichiers dans group_vars/ et host_vars/
  - les facts
  - *play* (vars, vars_file, vars_prompt)
  - un rôle (fichier vars/main.yml)
  - une tâche
  - un fichier inclus par include_vars / import_vars
  - par la ligne de commande (--extra-vars)
  - ...
- https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html

Rappel : variables (2)

```
my_var: Test
my_list:
  - lst1
  - lst2
my_listoneliner: [lst1, lst2]
my_collection:
  attr1: value1
  attr2: value2
my_collectiononeliner: { attr1: value1, attr2: value2 }
```

Rappel : fichiers de variables injectés automatiquement

- ${PWD}/
- inventory/group_vars/all(.yml)
- ${PWD}/group_vars/all(.yml)
- inventory/group_vars/<group>(.yml)
- inventory/host_vars/<server>(.yml)
- ${PWD}/group_vars/<group>(.yml)
- ${PWD}/host_vars/<server>(.yml)
- /path/to/myrole/vars/main.yml

Rappel : inclusion de fichier YAML

- inclusion de fichier de variables : include_vars
- inclusion de fichier de tâches : include_tasks / import_tasks
- inclusion de rôle : include_role / import_role
- inclusion de playbook : import_playbook
- Différence include / import
  - include_* remplace les références de variables ("{{ myvar }}.yml") par la valeur avant l'inclusion
  - include_* peut utiliser les conditions (when:)
- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_reuse.html#comparing-includes-and-imports-dynamic-and-static-reuse

Rappel : inclusion de fichier YAML (2)

```
---
- name: Playbook
  hosts: web
  become: True

  - name: Include vars
    ansible.builtin.include_vars: "{{ ansible_os_family }}.yml"

  - name: Include role memcached
    ansible.builtin.include_role: geerlingguy.memcached
    vars:
      memcached_listen_ip: 0.0.0.0
    when: memcached_setup | bool

  - name: Import role myapache
    ansible.builtin.import_role: test.myapache
    vars:
      myapache_listen_ip: 0.0.0.0

- name: Import playbook
  ansible.builtin.import_playbook: apache.yml
  vars:
    myapache_port: 8086
```

Rappel : répertoires de recherches des fichier YAML

- fichiers de tâches :
  - ${PWD}/
  - /path/to/myrole/tasks/
- fichiers de variables :
  - ${PWD}/
  - ${PWD}/vars/
  - /path/to/myrole/vars/

Rappel : template Jinja2

- Template Jinja : fichier avec des références de variables, boucles, conditions
- https://jinja.palletsprojects.com/en/2.10.x/
- Lors de l'exécution, le fichier sera généré (variables remplacées, ...)
- Customisation des fichiers par rapport à une machine, groupe, variables => réutilisabilité !
- Toutes les variables d'inventaire, de *play*, *facts*, ... sont disponibles lors du traitement par Jinja2

```
---
- name: Copy template file
  ansible.builtin.template:
    src: myfile.conf.j2
    dest: /etc/myservice/myfile.conf
    owner: root
    group: root
    mode: 0644
```

Rappel : template Jinja2 (2)

```
 <VirtualHost *:80>
  ServerName {{ apache_vhost_domain }}
  DocumentRoot /var/www/html/{{ apache_vhost_domain }}
  CustomLog /var/log/{{ apache }}/{{ apache_vhost_domain }}_access.log combined
  ErrorLog  /var/log/{{ apache }}/{{ apache_vhost_domain }}_error.log

  <Directory />
    Options none
    Allowoverride none
    Require all denied
  </Directory>

  <Directory /var/www/html/{{ apache_vhost_domain }}>
    Require all granted
  </Directory>
</VirtualHost>
```

Rappel : template Jinja2 (3)

```
<VirtualHost *:80>
 ServerName {{ apache_vhost_domain }}
 DocumentRoot /var/www/html/{{ apache_vhost_domain }}
 CustomLog /var/log/{{ apache }}/{{ apache_vhost_domain }}_access.log combined
 ErrorLog  /var/log/{{ apache }}/{{ apache_vhost_domain }}_error.log

 {% for item in apache_dirs  %}
 <Directory /var/www/html/{{ apache_vhost_domain }}/{{ item }}>
   Require all granted
 </Directory>
 {% endfor %}

 <Directory /var/www/html/{{ apache_vhost_domain }}/secure>
 {% if securehtpasswd is defined and securehtpasswd is string  %}
   AuthName "Please authenticate"
   AuthType Basic
   AuthUserFile /etc/httpd/{{ securehtpasswd }}
   require valid-user
 {% else %}
   require all denied
 {% endif %}
 </Directory>

</VirtualHost>
```

Rappel : template Jinja2 et filtres

- Des filtres peuvent être appliqués sur des variables et valeurs
- Valeur par défaut : {{ myvar | default(42) }}
- Omission d'une valeur si non défini : {{ myvar | default(omit) }}
- Validation : {{ myvar | ipaddr }} (renvoi False si ce n'est pas une adresse IP valide)
- Obligation : {{ myvar | mandatory }}
- Regex : {{ 'mystring in myarray' | regex_search('in') }}
- Regex : {{ 'mystring in myarray' | regex_replace('in', 'to') }}

Rappel : template Jinja2 et filtres (2)

- Génération mot de passe Unix :

  {{ 'passwordsecret' | password_hash('sha256', 'salt') }}
- Majuscule / minuscule : {{ distribution | upper }} / {{ distribution | lower }}
- Chemin : {{ path | basename }}, {{ path | dirname }}
- Opérateur ternaire : {{ mystate =='install' | ternary('present', 'absent') }}
- Conversion JSON : {{ mydict | to_json }}
- https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html
- https://blog.stephane-robert.info/docs/infra-as-code/gestion-de-configuration/ansible/filtres-jinja/

Rappel : les collections

- Depuis ansible 2.9, on peut utiliser les collections
- But :
  - désengorger le support de certains modules du code de base d'Ansible
  - moyen d'unifier des rôles, modules, filtres, tests unitaires pour les éditeurs 3rd party
- Tout ceci visent un même besoin fonctionnel étendue
- L'outil ansible-galaxy peut également créer et télécharger des *collections*
- Pour des systèmes spécifiques (Cisco, VMware, ...), vérifier si les éditeurs fournissent une collection !

Rappel : les collections (2)

- Chemins de recherche pour les collections :
  - ${PWD}/collections/ansible_collections
  - ~/.ansible/collections/ansible_collections
  - /usr/lib/python3/dist-packages/ansible_collections
- Quelques collections intéressantes :
  - ansible.posix
  - community.general
  - community.crypto
  - community.mysql
  - community.docker
  - ansible.windows
  - community.windows
  - ...

Rappel : Ansible Vault

- Création : ansible-vault create myvault.yml
- Edition : ansible-vault edit myvault.yml
- Visualisation : ansible-vault view myvault.yml
- Chiffrer un fichier existant : ansible-vault encrypt myvault.yml
- Utilisation (interactif) :

  ansible-playbook myplaybook.yml --vault-id=@prompt
- Utilisation (fichier) :

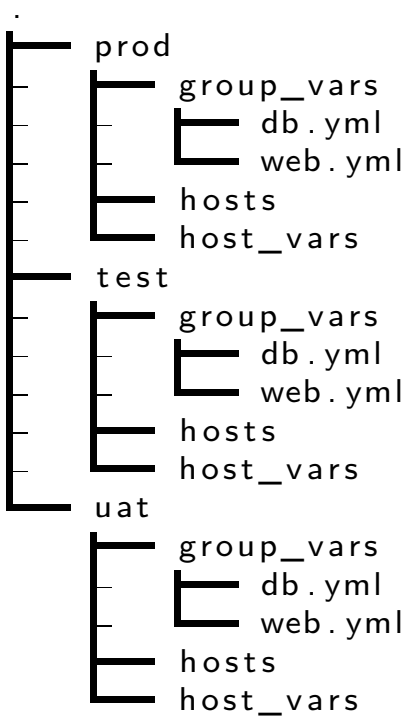
  ansible-playbook myplaybook.yml --vault-id=/path/to/file
- Répéter autant de --vault-id que de fichiers chiffrés (avec des mots de passe différent)

- TP 1

Arborescence d'inventaire

- Un répertoire avec un inventaire par environnement
- $+$ les variables d'infrastructure de cet environnement
- Un répertoire unique avec tous les environnements ?
- Un dépôt unique avec un sous-module Git par environnement ?

Arborescence d'inventaire (2)



Plugin d'inventaire dynamique

- Utilisation d'un plugin existant (via une collection) pour générer l'inventaire
- L'activation peut se faire via le paramètre enable_plugins de la section [inventory] du fichier ansible.cfg
  - aws_ec2, openstack, vmware_vm_inventory, ...
- La configuration du plugin est un fichier YAML qui est utilisé à la place du paramètre inventory (ansible-playbook -i libvirt.yml)

Script d'inventaire dynamique

- Executable (Python, bash, ...) qui va générer l'inventaire
  - récupération des serveurs via base de données, LDAP, AD, ...
- Ecriture sur la sortie standard (stdout) de l'inventaire au format JSON
- Utile pour des cas complexe quand il n'existe pas de plugins ou les inventaires dynamiques / statiques sont insuffisants

Script d'inventaire dynamique (2)

```
#!/usr/bin/env python3
import json

output = {
    "web": { "hosts": [] },
    "db": { "hosts": [] }
}

webs = ["vmdebian", "vmalma"]
dbs = ["localhost"]

for server in webs:
    output["web"]["hosts"].append(server)

for server in dbs:
    output["db"]["hosts"].append(server)

print(json.dumps(output))
```

Inventaire et serveur de rebond

- Connexion à travers un serveur de rebond
- Paramètre *ansible_host* et *ansible_port*
- Ansible utilisera *ansible_host*:*ansible_port* pour se connecter

```
[web]
lweb[1:16].example.lan  ansible_host=10.67.0.11

[db]
db1-sxb.example.lan     ansible_host=10.67.0.21 ansible_port=2222
db2-sxb.example.lan     ansible_host=10.67.0.21 ansible_port=2222
```

Ajout d'hôtes / groupes dynamiques

- Module ansible.builtin.add_host
- Ces groupes / hôtes resteront valides jusqu'à la fin du *playbook*

```
---
- name: Adds hosts
  hosts: localhost

  tasks:
    - name: Adds host to inventory
      ansible.builtin.add_host:
        name: "{{ srvad01_address }}"
        groups: ad
        # variable
        myadvar: agdlp
        ansible_port: 2222
```

Création de groupes dynamiques

- Module ansible.builtin.group_by
- Création de nouveaux groupes suivant des variables (*facts*)
- Les hôtes existants seront automatiquement ajoutés au groupe correspondant
- Sous-groupes possibles
- Ces groupes resteront valides jusqu'à la fin du *playbook*

Création de groupes dynamiques (2)

```
---
- name: Adds hosts
  hosts: localhost
  connection: local

  tasks:
    - name: Create groups based on the machine architecture
      ansible.builtin.group_by:
        # machine_x86_64, machine_x86, machine_aarch64, machine_arm, ...
        key: machine_{{ ansible_machine }}

    - name: Create nested groups
      ansible.builtin.group_by:
        key: el{{ ansible_distribution_major_version }}-{{ ansible_architecture }}
        parents:
          - el{{ ansible_distribution_major_version }}

- name: Adds hosts
  hosts: machine_x86_64

  tasks:
    - name: Print
      ansible.builtin.debug:
        var: ansible_hostname
```

Factorisation

- Pour un même besoin fonctionnel répété plusieurs fois dans un playbook
  - on créé un rôle
  - on l'ajoute dans la section role ou include_role
  - on passe des variables **au niveau du rôle**
- Pour une suite de tâches répétée plusieurs fois dans un playbook ou un rôle
  - on créé un fichier séparé
  - on inclut
  - on passe des variables **au niveau de la tâche**

Factorisation (2)

```
---
- name: Backup files
  vars:
    mydirectory: /home
    mybackup_location: /mnt/nas/backup/prod/
  ansible.builtin.include_tasks: backup.yml

- name: Install webapp
  vars:
    tomcat_webapp: "{{ item }}"
    tomcat_root: /var/www/tomcat/
  ansible.builtin.include_role:
    name: tomcat
  with_items:
    - restaurant
    - conges
```

Factorisation (3)

```
---
- name: Play
  hosts: web
  become: True

  roles:
    - name: tomcat
      vars:
        tomcat_webapp: restaurant
        tomcat_root: /var/www/tomcat
```

Arguments par défaut de modules

- Paramètre de *play*, tâche ou *block* : module_defaults
- Dictionnaire de modules avec les valeurs par défaut des arguments
- Utile quand plusieurs tâches utilisent
  - le même module
  - les mêmes arguments

Arguments par défaut de modules (2)

- Possible de surcharger les valeurs par défaut
- Dans un rôle ?
  - Module block avec le paramètre *module_defaults* + les sous-tâches

Arguments par défaut de modules (3)

```
- name: Play
  hosts: all
  become: True

  module_defaults:
    ansible.builtin.file:
      owner: apache
      group: apache
      mode: 0640

  tasks:
    - name: Create file
      ansible.builtin.file:
        path: /tmp/illbebach.txt
        state: touch

    - name: Create file
      ansible.builtin.file:
        path: /tmp/turbotwins
        state: directory
        mode: 0700
```

Ancres / alias

- Référence d'un bloc YAML
- Evite de réécrire un bloc YAML complet à chaque fois
- Disponible entre les différents *play* d'un même playbook
- Utile pour définir la même valeur d'une variable
- Possible de surcharger les valeurs du bloc

Ancres / alias (2)

- myvar : &myaliasname : copie l'intagralité de myvar
- myothervar: *myaliasname : colle la valeur dans myothervar
- <<: *myaliasname : fusionne le bloc (en cas de surcharge)
- Attention : la fusion de liste va générer un dictionnaire
- Attention : dans un rôle, ne pas définir les ancres dans vars/main.yml
  - définir les ancres dans la section vars d'une tâche et elles seront disponibles pour toutes les autres taĉhes !

Ancres / alias (3)

```
- name: Play
  hosts: all
  become: True

  vars:
    mystring: &mystr_opts Not to be

    mystring2: *mystr_opts

    myvhost: &myvhost_opts
      port: 80
      tls_port: 443
      user: www-data
      group: www-data
      path: /var/www/html/myvhost

    myothervhost:
      # inherit all children elements of myvhost (port, tls_port, ...)
      <<: *myvhost_opts
      # overload
      path: /var/www/html/myothervhost
```

Ancres / alias (4)

```
- name: Play
  hosts: all
  become: True

  vars:
    mylist: &mylist_opts
    - el1
    - el2

    mylist2: *mylist_opts
    # copy of mylist

    mylist2:
      - *mylist_opts
      - el3
    # WARNING: mylist2 has TWO elements an array (containing el1 and el2) + el3

    mylist2:
      *mylist_opts
      - el3
    # ERROR: syntax error
```

Les *facts*

- Les *facts* sont des informations récupérées sur les cibles :
  - nom d'hôte, OS, adresses IP, CPU, mémoire, ...
- L'étape du *gathering facts* se fait au début d'un *play*
- Pour voir les *facts* : ansible host -m setup
- Ces valeurs sont disponibles comme variables
- Ajout de variables facts : module *ansible.builtin.set_fact*

```
---
- name: Set variable facts
  ansible.builtin.set_fact:
    my_var: Test
    my_list:
      - lst1
      - lst2
    my_collection:
      attr1: value1
      attr2: value2
    my_user: {{ res.stdout }}
```

Les *facts* (2)

- Désactiver la génération : *gather_facts: False* dans un *play*
- Générer uniquement un sous-ensemble des facts:
  - *gather_facts: False*
  - une tâche avec le module ansible.builtin.setup + argument *gather_subset*)
  - léger coût en temps et on a accès aux variables que l'on souhaite

```
---
- name: Retrieves distribution facts
  ansible.builtin.setup:
    gather_subset:
      - '!all'
      - distribution
```

Les *facts* (3)

- Possibilité de mise en cache des facts : accélère le traitement des *playbooks* !
  - configuration dans le fichier de configuration ansible.cfg
  - paramètres *gathering*, *fact_caching*, *fact_caching_connection*, *fact_caching_timeout*
  - Stockage en base Redis ou fichier JSON
- Vidage du cache : ansible-playbook --flush-cache playbook.yml

```
[defaults]
# ... other configuration
gathering = smart
fact_caching = jsonfile
fact_caching_connection = /tmp/ansible/cache
```

Les *custom facts*

- *Custom facts* : informations à rajouter aux *facts*
- Ajouter des fichiers *.fact* dans /etc/ansible/facts.d/
  - Fichier au format INI ou JSON
  - Exécutable qui affiche du texte au format INI ou JSON
- Ces variables seront dans la section *ansible_local* des facts

```
$ cat /etc/ansible/facts.d/site.fact
{
  "myfactstr": "test",
  "myfactlist": ["element1", "element2"],
  "myfactdict": {
    "name": "vhost",
    "path": "/var/www/html"
  }
}
```

Les *custom facts* (2)

```
$ ansible -i inventory/hosts -m setup vmdebian
  ...
        "ansible_local": {
            "site": {
                "myfactdict": {
                    "name": "vhost",
                    "path": "/var/www/html"
                },
                "myfactlist": [
                    "element1",
                    "element2"
                ],
                "myfactstr": "test"
            }
        },
  ...
```

Modules *service_facts* et *package_facts*

- Module *ansible.builtin.service_facts* pour avoir des informations sur les services du système (liste + état)
- Commande ad-hoc : *ansible -m service_facts mygroup*
- Module *ansible.builtin.package_facts* pour avoir des informations sur les paquets installés
- Commande ad-hoc : *ansible -m package_facts mygroup*
- Bien sûr, ces modules peuvent être utilisés dans un *playbook* (avec un *register*) afin d'avoir ces informations en YAML pour produire des audits ou s'en servir comme conditions

Paramètre *delegate_to*

- Il est parfois nécessaire de déléguer une tâche d'un serveur à une autre machine
- Exemple : sortir une machine de la répartition, rsync d'un répertoire local
- Le nom de la machine en cours sera {{ inventory_hostname }}
- Les variables seront aussi celles de la machine en cours
- Si le "délégué" est localhost, on peut utiliser *local_action*

Paramètre *delegate_to* (2)

```
- name: Remove node from load-balancer
  ansible.builtin.command: /usr/local/bin/lb_remove_node {{ inventory_hostname }}
  delegate_to: 127.0.0.1

- name: Remove node from load-balancer
  local_action: ansible.builtin.command /usr/local/bin/lb_remove_node {{ inventory_hostname }}

- name: recursively copy files from management server to target
  local_action: ansible.builtin.command rsync -a /path/to/files {{ inventory_hostname }}:/path/to/target/
```

Paramètre *delegate_facts*

- On peut aussi déléguer l'exécution du module setup (*facts* à d'autres machines
- On aura donc accès aux *facts* de ces machines pour notre *play* même si elles ne font pas partie des hôtes !
- Exemple : avoir accès aux *facts* des machines du groupe database (adresse IP, ...) tout en exécutant un *playbook* sur le groupe web

Paramètre *delegate_facts* (2)

```
- name: Playbook
  hosts: app_servers
  become: True

  tasks:
    - name: gather facts from db servers
      ansible.builtin.setup:
      delegate_to: "{{ item }}"
      delegate_facts: True
      loop: "{{ groups['dbservers'] }}"
```

Lookups

- Les *lookups* sont utilisés pour récupérer des données *externes* **depuis le contrôleur**
- Cela peut être un fichier en local, enregistrement DNS, *credentials* AWS, mot de passe aléatoire, ...
- Utilisable directement dans une argument de module, via une variable ou dans un fichier template Jinja
- Les structures *with_* utilisent un *lookup* (items, nested, fileglbo, dict, cvsfile, ...)
- https://docs.ansible.com/ansible/latest/playbooks_lookups.html

Lookups (2)

```
---
- name: Set authorized key for user root copying it from current user
  ansible.posix.authorized_key:
    user: root
    state: present
    key: "{{ lookup('ansible.builtin.file', lookup('ansible.builtin.env','HOME') + '/.ssh/id_rsa.pub') }}"

- name: Create a user with random password
  vars:
    mypassword: "{{ lookup('password', '/dev/null', chars='ascii_letters') | password_hash('sha512') }}"
  ansible.builtin.user:
    name: user
    password: "{{ mypassword }}"
    shell: /bin/bash
    group: wheel
    update_password: on_create

- name: Unvault an encrypted text file
  vars:
    mypassword: "{{ lookup('ansible.builtin.unvault', '/tmp/password.txt') }}"
  ansible.builtin.debug:
    msg: "{{ mypassword }}"
```

Lookups (3)

- Quelques *lookup* intéressants:
  - ansible.builtin.file
  - ansible.builtin.password
  - ansible.builtin.unvault
  - ansible.builtin.env
  - ansible.builtin.url
  - ansible.builtin.first_found
  - ansible.builtin.template
  - community.general.dig
  - kubernetes.core.k8s
  - kubernetes.core.kustomize
- Liste des *lookups* en local : ansible-doc -t lookup -l
- https://docs.ansible.com/ansible/latest/collections/index_lookup.html
- https://docs.ansible.com/ansible/latest/plugins/lookup.html

Lookups et query

- Lot mot clé *lookup* peut retourner une chaine de caractère où les différentes valeurs sont séparées par des virgules
- Impossible d'itérer dessus avec un *with_items* / *loop* / ...
- Si on souhaite une liste, il faut ajouter le paramètre wantlist=True
- Ou remplacer *lookup* par *query* (alias *q*)

Lookups et query (2)

```
---
- name: Print variable with lookup
  ansible.builtin.debug:
    msg: "{{ lookup('community.general.dig', 'www.lafibre.info', qtype='NS') }}"

- name: Print variable with lookup and wantlist=True
  ansible.builtin.debug:
    msg: "{{ lookup('community.general.dig', 'www.lafibre.info', qtype='NS', wantlist=True) }}"

- name: Print variable with query
  ansible.builtin.debug:
    msg: "{{ query('community.general.dig', 'www.lafibre.info', qtype='NS') }}"

- name: Print variable with query alias
  ansible.builtin.debug:
    msg: "{{ q('community.general.dig', 'www.lafibre.info', qtype='NS') }}"
```

Include spécifiques

```
---
- name: Include first matching vars files
  vars:
    params:
      - "{{ ansible_os_family }}.yml"
      - "{{ ansible_distribution }}.yml"
      - default.yml
  ansible.builtin.include_vars: "{{ lookup('ansible.builtin.first_found', params) }}"

- name: Include all files from includes/ directory
  ansible.builtin.include_vars:
    dir: includes/
```

Filtres avancés

- Itérations sur une liste d'éléments et ne garde que ceux qui respecte une condition
  - select
  - reject
- Itérations sur une liste de dictionnaire et ne garde que ceux qui respecte une condition sur **UN** attribut
  - selectattr
  - rejectattr
  - map(attribute='*attrname*')
- JSON : json_query

Filtres avancés (2)

- Filtres de transformation sur les éléments
  - map('*filtername*')
  - map('upper'), map('regex_replace', 'terraform', 'opentofu'), ...
- https://jinja.palletsprojects.com/en/stable/templates/#jinja-filters.map
- https://docs.ansible.com/ansible/latest/playbook_guide/complex_data_manipulation.html
- Ne pas hésiter à utiliser des variables pour stocker le résultat (évite la duplication de code)

Filtres avancés (3)

```
---
- name: Test
  hosts: localhost
  connection: local

  vars:
    nic:
      - eth0
      - eth1
      - wlan0
      - wlan1

    numa:
      - name: cpu1
        core: 6
      - name: cpu2
        core: 8
      - name: cpu3
        core: 4
      - name: cpu4
        core: 16

  tasks:
    - name: Print wired NIC
      ansible.builtin.debug:
        msg: "{{ item }}"
      loop:
        - "{{ nic | select('match', 'eth[0-9]+') }}"
        - "{{ numa | selectattr('core', '>', 6) }}"
        - "{{ numa | map(attribute='core') | select('lessthan', 8) }}"
        - "{{ nic | map('upper') }}"
        - "{{ numa | map(attribute='name') | map('upper') }}"
        - "{{ ['terraform', 'aws', 'openstack'] | map('regex_replace', '^terraform', 'opentofu') }}"
```

Dépendances de rôles / collections

- Fichier requirements.yml pour spécifier les rôles et collections requis pour un projet playbook
  - ${PWD}/
  - ${PWD}/roles/
  - ${PWD}/collections/
  - /path/to/myrole/

Dépendances de rôles / collections (2)

- Installation : ansible-galaxy -r requirements.yml
- Attention : ansible / ansible-playbook ne récupèrent **PAS** automatiquement les dépendances
- Note : AWX / AAP va charger le fichier requirements.yml contenu dans ${PWD}/roles/ et ${PWD}/collections/ d'un projet

Dépendances de rôles / collections (3)

```
---
collections:
  - name: ansible.posix
  - name: community.crypto
  - name: community.general
    version: ">=10.1.0"
  - name: myname.myrepo
    source: https://github.com/myname/myrepo.git
    version: main

---
roles:
  - name: geerlingguy.memcached
  - name: myname.myrole
    src: https://github.com/myname/myrole.git
    version: main
```

Dépendances de rôles / collections (4)

- Si un rôle dépend d'un autre rôle, il faut le spécifier dans meta/main.yml
- Attention : ansible / ansible-playbook ne récupèrent **PAS** automatiquement les dépendances
- Il faudra faire soi-même un *ansible-galaxy install -r requirements.yml*

```
...
dependencies:
  - role: geerlingguy.memcached
  - role: geerlingguy.apache
...
```

Responsabilité et inversion des dépendances de rôles

- En cas de dépendances avec des rôles (exemple reverse-proxy et application)
- Chaque rôle va installer ses propres fichiers de configuration
- Utilisation de variables
  - pour activer ou non certaines fonctionnalités
  - pour savoir le nom de tel ou tel service (reverse-proxy nginx ou apache ?)
- https://blog.ght1pc9kc.fr/2022/stratégie-des-rôles-ansible/

- TP 2

Qualité / tests

- Outils :
  - yamllint
  - ansible-playbook --syntax-check
  - ansible-lint
  - molecule
- Il est fortement recommendé d'utiliser ces outils pour vérifier la syntaxe et les bonnes pratiques
- ansible-lint permet de détecter des pratiques à risques (exemple: "mode: 644" à la place de "mode: **0**644")

Qualité / tests (2)

- Fichier .yamllint et .ansible-lint pour relâcher les règles
- A mettre à la racine des projets playbook et à des rôles
- Installation
  - Paquet ansible-lint ou python3-ansible-lint
  - ou python3 -m pip install ansible-lint
  - ou pipx install ansible-lint

Fichier .yamllint

```
---
extends: default

rules:
  line-length:
    max: 150

  empty-lines:
    max-end: 1

  braces:
    min-spaces-inside: 0
    max-spaces-inside: 1

  brackets:
    min-spaces-inside: 0
    max-spaces-inside: 1

  truthy:
    allowed-values:
      - 'True'
      - 'False'
      - 'true'
      - 'false'
      - 'yes'
      - 'no'
```

Fichier .ansible-lint

```
# .ansible-lint
skip_list:
  - yaml[line-length]

warn_list:
  - package-latest
```

Assertion

- Module *ansible.builtin.assert*
- Vérification à l'exécution de condition(s) / pré-requis
- Idéal pour des tests unitaires ou dans un playbook
- arguments fail_msg et success_msg pour personnaliser les messages en cas d'échec ou de succès

Assertion (2)

```
---
- name: Assert
  ansible.builtin.assert:
    that:
      - ansible_os_family == "Debian"
      - ansible_distribution_major_version | int == 12
```

Validation

- Module *ansible.builtin.validate_argument_spec*
- Vérification **avant l'exécution** de condition(s) / pré-requis
- Idéal pour vérifier qu'un playbook / rôle a toutes les variables nécessaires
- Ou qu'une variable a une valeur bien précise (i.e. started, stopped, restarted, reloaded)

Validation (2)

```
---
- name: Validate
  vars:
    required_data:
      myrole_host:
        description: "VirtualHost required"
        type: str
        required: true
      myrole_state:
        description: "State"
        choices: ['process', 'thread', 'coroutine']
        type: str
        default: process
  ansible.builtin.validate_argument_spec:
    argument_spec: "{{ required_data }}"
```

Validation des variables de rôle

- Fichier meta/argument_specs.yml dans l'arborescence du rôle
- Défini les variables attendues (type, choix de valeur, description, ...)
- ansible-doc peut également générer la documentation au format *man* à partir de ces données
- Attention : disponible à partir de la version 2.11

Validation des variables de rôle (2)

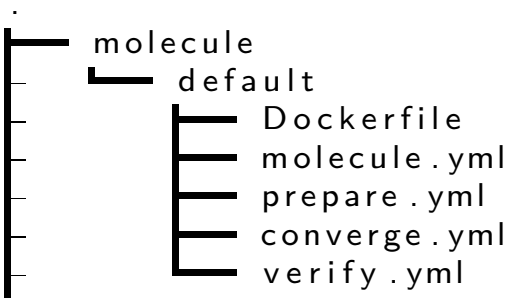
```
---
argument_specs:
  main:
    short_description: "Create some stuff."
    description:
      - Create some stuff on the server
    author: Arnold Blackedecker

    options:
      myrole_state:
        type: str
        choices: ['process', 'thread', 'coroutine']
        required: true
        description:
          - Method of parallelism
      myrole_hosts:
        type: list
        required: true
        description:
          - Method of parallelism
        elements: dict
        options:
          name:
            description:
              - VirtualHost FQDN
            type: str
            required: true
          path:
            description:
              - VirtualHost path (i.e. /var/www/html)
            type: str
            default: /var/www/html/
```

Molecule

- Framework pour les tests unitaires des rôles
- *Test-Driven Development* : écriture des tests unitaires avant le code du rôle
- Répertoire *molecule* à la racine du rôle
- Sous-répertoires pour les scénarios (par défaut *default*)

Molecule (2)


.
molecule
default
Dockerfile
molecule.yml
prepare.yml
converge.yml
verify.yml

Molecule (3)

- Installation
  - python3 -m pip install molecule molecule-plugins
  - pipx install molecule --pip-args molecule-plugins
- Plusieurs plugins de création d'instance : docker, podman, vagrant, ...
- Création de l'arboresence au sein d'un rôle existant
  - *molecule init scenario --driver-name docker*
  - *molecule init scenario --driver-name docker nameofscenario*

Molecule : scenario

- fichier molecule.yml défini les paramètres de l'instance (VM, conteneur)

```
---
dependency:
  name: galaxy
  options:
    # roles only
    role-file: requirements.yml
    # collections + roles only
    requirements-file: requirements.yml
driver:
  name: docker
platforms:
  - name: instance
    image: docker.io/almalinux:9
    privileged: True
    volume_mounts:
      - "/sys/fs/cgroup:/sys/fs/cgroup:rw"
    command: "/usr/sbin/init"
    environment: { container: docker }
    privileged: True
    pre_build_image: true
```

Molecule : séquences d'un scénario

- dependency : installation des dépendances (rôles / collections)
- create : création de l'instance
- prepare : playbook pour installer les prérequis
- converge : playbook pour installer le rôle en lui-même
- verify : playbook pour les tests unitaires
- destroy : destruction de l'instance

Molecule : écriture des tests unitaires

- ansible.builtin.uri avec headers / return_content / register / failed_when : test d'une URL
- ansible.builtin.wait_for : test d'un port TCP
- ansible.builtin.services_facts + assert : test si un service est démarré / activé (vérification du paramètre de retour*state* et *status*)
  - NE PAS mettre de register, la variables *services* sera renseigné automatiquement
  - *that: services["docker.service"]["state"] == "running")*
- ansible.builtin.stat avec register / failed_when : test si un fichier est présent, a les bonnes permissions / propriétaire / groupe, ...

Molecule : écriture des tests unitaires (2)

- Module classique en mode simulation avec failed_when **en cas de changement ou erreur**
  (ansible.builtin.{lineinfile,users,packages,...}, ..)

Molecule : list / dependency

- *molecule list*
  - Liste des scénarios
- *molecule prepare*
  - Installation des dépendances galaxy (rôles, collections)
  - Fichier requirements.yml à la racine du rôle

Molecule : create / prepare

- *molecule create*
  - Créé l'instance (VM, conteneur, ...)
- *molecule prepare*
  - Lance le playbook prepare.yml pour installer les prérequis logiciels

Molecule : converge

- *molecule converge*
  - Lance les séquences *dependency, create et prepare* si ce n'est pas déjà fait
  - Lancement du playbook converge.yml pour déployer le rôle

Molecule : verify / idempotence

- *molecule verify*
  - à lancer **après** un *molecule converge*
  - lance soit le playbook verify.yml qui effectue les tests unitaires
  - soit un script avec le framework testinfra
- *molecule idempotence*
  - A faire après un *molecule converge*
  - Determine si une playbook est idempotent (i.e. aucune tâche dans l'état *changed* après un converge)

Molecule : destroy / reset

- *molecule destroy*
  - Suppression de l'instance
- *molecule reset*
  - Réinitialisation de l'instance

Molecule : test

- *molecule test*
  - Lance l'intégralité des séquences de chaque scénario
- Idéal pour valider tout avec une seule commande
- A noter que la commande détruit également les conteneurs / VM à la fin

Tests d'intégration de rôle / playbook : testinfra

- Framework testinfra
- Ecriture des tests unitaires en Python
- Installation
  - Paquet python3-testinfra / python3-pytest-testinfra+ansible
  - ou python3 -m pip install pytest-testinfra
- Fichiers mytest.py dans ${PWD}/tests/

Tests d'intégration de playbook : testinfra (2)

```
def test_apache2_is_running(host):
    web = host.service("apache2")
    assert web.is_running

def test_sshd_config(host):
    sshd_config = host.file('/etc/ssh/sshd_config')
    assert sshd_config.exists
    assert sshd_config.contains('PermitRootLogin without-password')
    assert sshd_config.contains('Port 2222')
```

Molecule et testinfra

- Molecule $<$ 6.x intègre testinfra en dépendance
- Molecule $>=$ 6.x n'intègre plus testinfra mais cela reste possible de l'installer à part et de l'utiliser
- Fichiers mycase.py dans molecule/myscenario/tests/
- Le projet Molecule se concentre pour utiliser Ansible comme outil de vérification

Profiling

- Dans le fichier ansible.cfg, section [default] : callbacks_enabled = timer, profile_tasks, profile_roles
- Fourni des informations sur le temps passé sur la génération des facts, des tâches, des rôles

Debugger

- Dans le fichier ansible.cfg, section [default] : enable_task_debugger = True
- Au niveau d'un *play* ou une tâche, debugger: on_failed
- Variable d'environnement : ANSIBLE_ENABLE_TASK_DEBUGGER=True
- Idéal pour voir les arguments du module / variables globales utilisées pour une tâche

Debugger (2)

- Si une tâche est en erreur, affiche un prompt
  - p <name> : affiche la valeur de <name>
  - <name> = newvalue : spécifie une nouvelle valeur pour <name>
  - u : recrée la tâche avec les variables à jour
  - r : relance la tâche
  - c : lance la prochaine tâche
  - q : quitte le debugger
- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_debugger.html

Debugger (3)

- Le paramètre <name> peut être
  - task.args (avec un point)
    - dictionnaire des arguments **du module**
    - task.args['state'], task.args['regexp'], ...
  - task_vars (avec un underscore)
    - dictionnaire des variables Ansible (facts, play, vars, ...)
    - task_vars['ansible_os_family'], task_vars['memcached_listen_ip']

- TP 3

run_once

- Exécution d'une tâche sur une unique machine d'un groupe
  - la première du groupe
  - sinon utiliser delegate_to pour indiquer la machine
- Cas d'usage : migration / sauvegarde d'une base de données, appel d'API via une machine spécifique, ...

```
---
- name: Play
  hosts: db

  tasks:
    - name: Backup database
      ansible.builtin.command:
        cmd: /opt/backup_db.sh
      run_once: True
```

Parallélisme, batch

- Paramètre *forks* (ansible.cfg) : nombre de machines à traiter en parallèle **par tâche** (i.e. nombre de connexion SSH)
- Stratégie d'exécution (paramètre de *play* ou dans ansible.cfg)
  - *linear* (défaut), attente de la fin d'une tâche **sur toutes les machines** avant de passer à la suivante
  - *free* : dès qu'une machine a fini une tâche, elle passe à la suivante
- S'il y a des dépendance entre des machines, gardez *linear*

```
---
- name: Play
  hosts: all
  strategy: free

  tasks:
```

Parallélisme, batch (2)

- Paramètre de play *serial* : rolling-update
- Exécution d'un *play* sur un certain lot de machines (nombre ou pourcentage)
- Puis on réexecute le *play* sur le prochain lot de machines

```
---
- name: Play
  hosts: all
  become: True
  serial: 2

  tasks:
    - name: Install vim
      ansible.builtin.package:
        name: vim
        state: present
```

Parallélisme, batch (3)

- Paramètre de tâche *throttle* : nombre de machine pouvant exécuter la tâche en parallèle
- Attention : *throttle* doit être plus petit que *forks* et *serial*
  https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_stra

```
---
- name: Play
  hosts: all
  become: True

  tasks:
    - name: Install vim
      throttle: 1
      ansible.builtin.package:
        name: vim
        state: present
```

Tâches asynchrones

- Paramètre de tâche *async* : attente maximum en seconde pour la tâche
- Deux possibilités pour le paramètre de tâche *poll* :
  - *poll* > 0 : attente active, teste toute les X secondes si la tâche est terminée
  - *poll* $= 0$: on passe à la tâche suivante + écriture d'une tâche de synchronisation (module async_status + do / while) pour connaître la fin de la tâche
-

https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_async.html

Tâches asynchrones (2)

```
---
- name: Downloads file
  ansible.builtin.get_url:
    url: https://www.vincent-netsys.fr/bigfile.zip
    dest: /tmp/test.zip
    owner: root
    group: root
    mode: 0644
  async: 600
  poll: 0
  register: job

# ... other tasks

- name: Check async task
  ansible.builtin.async_status:
    jid: "{{ job.ansible_job_id }}"
  register: result
  until: result.finished
  retries: 30
  delay: 5
```

Tâches asynchrones (3)

```
- name: Downloads several files
  ansible.builtin.get_url:
    url: "{{ item }}"
    dest: /tmp/
  async: 600
  poll: 0
  loop:
    - https://www.vincent-netsys.fr/bigfile.zip
    - https://www.vincent-netsys.fr/megabigfile.zip
  register: jobs

# ... other tasks

- name: Check async task
  ansible.builtin.async_status:
    jid: "{{ item.ansible_job_id }}"
  loop: "{{ jobs.results }}"
  register: results
  until: results.finished
  retries: 30
  delay: 5
```

Approche PULL

- Une approche PULL est possible avec Ansible
- Installer Ansible sur toutes les machines cibles
- Stocker les *playbooks* dans un dépôt git
- Utiliser la commande ansible-pull dans un cron
- A interval régulier, la machine va télécharger les *playbooks* du dépôt git et les exécuter en local (sur localhost)

Approche PULL (2)

- Paramètre *hosts* du playbook : mettre à *all* ou *localhost*
- Privilégier des tâches simples et limiter les impacts systèmes
- Intéressant pour du contrôle de conformité (vérification de permissions, démarrage antivirus, mise à jour de paquets, ...)

```
$ ansible-pull -U https://gitlab.com/s-vincent/test.git -C main play.yml
```

Approche PULL : limites

- Gestion des erreurs compliquée, il faut passer sur les machines pour voir si tout s'est bien passé
- Les serveurs doivent avoir accès au(x) dépôt(s) Git (donc les credentials / clés SSH avec accès en lecture seule)
- Pas de synchronisation entre les serveurs d'un groupe
- Pas de variables partagées avec les autres serveurs (hostvars, ...)

- TP 4

Ansible Automation Controller (Tower)

- Produit commercial + support par Red Hat
- Interface web
- Centralisation des inventaires, authentications, déploiements, ...
- Lancement des playbooks en mode manuel ou planifié
- Utilisation avec **Kubernetes** en production
- Version communautaire : AWX
- https://www.redhat.com/en/technologies/management/ansible/automation-controller
- https://github.com/ansible/awx

Ansible Automation Controller (Tower) (2)

- Role-Based Access Control
- Intégration avec LDAP, ActiveDirectory, ...
- API REST
- Logging (qui a fait quoi, quand et où)
- Déploiement simplifié
- Dashboard des opérations
- Clustering, ...
- Workflow d'opérations

Concepts

- Organisations
- Equipes / utilisateurs
- Projets (dépôts Git)
- Informations d'identification (clé SSH, token, ...)
- Inventaires / hôtes
- Modèle (job de déploiement)

Organisations / équipes / utilisateurs

- Gestion des accès
- Utilisateur associé à une organisation
- Utilisateur associé à une ou plusieurs équipes
- Equipes associée à une organisation
- Possible de configurer un accès LDAP / AD pour récupérer les informations
- https://docs.ansible.com/automation-controller/4.4/html/administration/ent_auth.html

Projets Git

- Les playbooks sont récupérés sur un dépôt de source
- Comment AWX peut connaître / installer les dépendances (collections, rôle à inclure, ...) ?
  - *ansible-galaxy role install ${PWD}/roles/requirements.yml*
  - *ansible-galaxy collection install ${PWD}/collections/requirements.yml*
- https://docs.ansible.com/ansible-tower/latest/html/userguide/projects.html#ansible-galaxy-support

Credentials

- Plusieurs types de *credentials* disponibles : accès machine (SSH, mot de passe), dépôt Git, token GitHub / GitLab, vault, ...
- Plusieurs niveaux d'accrédition : utilisation, lecture, administration
  - Un utilisateur peut utiliser une clé SSH sans jamais à la connaître

Mot de passe des fichiers vault

- Créer vos fichiers vault.yml avec un identification :

  ansible-vault --vault-id clientsprod@prompt create clients.yml
- Dans AWX, information d'identification : *vault* (ou archivage sécurisé en français)
  - ajouter le mot de passe de déchiffrement
  - ajouter l'identificant du fichier
- Possible de faire une demande intéractive au lancement

Inventaires

- A la main en créant les groupes, les serveurs et l'assocation entre groupe et serveurs
- Possibilité d'ajouter variables de groupe dans l'interface
- A partir d'une sources externes (git, ...)
- *Smart Inventory* : sous-ensemble d'un même inventaire (via un filtre de recherche)
- *Constructed Inventory* : sous-ensemble de plusieurs inventaires (via un filtre de recherche)
- https://docs.ansible.com/automation-controller/latest/html/userguide/inventories.html

Modèles de job

- Déploiement d'un playbook sur
  - provenant **d'un projet**
  - sur les machines **un inventaire**
  - avec **un moyen d'authentification** (*credential*)
- L'utilisateur créant un modèle doit bien sûr avoir les accès aux différents composants

Modèles de job (2)

- Customisation de l'exécution : variables, serial, mode verbeux, ...
- Récupération de la sortie d'ansible-playbook
- Lancement manuel
- Lancement périodique (tâche planifiée)
- Exécution du playbook en environnement conteneurisé

Environnements d'exécution

- Image de conteneur pour lancer les déploiements (job)
  - image de base
  - interpréteur Python
  - ansible
  - collections / roles
  - dépendances systèmes des modules (rsync pour synchronize, ...)
  - ansible-runner

Environnements d'exécution (2)

- Création d'un environnement d'exécution Ansible : ansible-builder
- Ficher execution-environment.yml
- Création de l'image Docker :
  ansible-builder build --tag myexec-env:1.0
- https://github.com/ansible-community/images/

Environnements d'exécution (3)

```
---
version: 3

dependencies:
  galaxy: requirements.yml
  python: requirements.txt
  system: bindep.txt

images:
  base_image:
    name: ghcr.io/ansible-community/community-ee-base:2.18.1-1

additional_build_steps:
  prepend_base:
    - RUN yum install -y git
```

Webhooks

- Webhook disponible pour GitHub et GitLab
- Lancement d'un job à réception d'un évènement d'un dépôt GitHub ou GitLab (push, ...)
- https://docs.ansible.com/automation-controller/4.4/html/userguide/webhooks.html

AWX cli

- Existence d'une CLI : awx (paquet Python awxkit)
- Pilotage en ligne de commande : lancement de job, ...

Alternatives

- Sempahore UI
- Interface web pour déployer des playbooks ansible
- ... mais aussi du scripts Terraform / OpenTofu
- Plus léger qu'AWX et ne nécessite pas Kubernetes
- https://semaphoreui.com/
- https://github.com/semaphoreui/semaphore

Alternatives (2)

- RunDeck
- Gitlab CI
- Jenkins

- TP 5

Derniers mots

- Environnement avec des VMs qui changent souvent : utilisez les inventaires dynamiques
- Augmentez la qualité du code avec ansible-lint
- Ajoutez des tests unitaires à vos rôles avec Molecule
- Ajoutez le fichier meta/argument_specs.yml à vos rôle pour valider les variables utilisées dans le rôle
- Faîtes des déploiements de type rolling-update (serial)
- Si beaucoup de monde peut / veut faire des MEP en autonomie : centralisation avec AWX (+ RBAC sur les inventaires, *credentials*, ...)

Liens intéressants

- Site web : https://www.ansible.com
- Documentation : https://docs.ansible.com/ansible
- Modules : htts://docs.ansible.com/ansible/latest/modules/modules_by_category.html
- Ansible dans la vraie vie avec AWX : https://www.youtube.com/watch?v=1-bzgTKHX-s
- Framework Molecule : https://ansible.readthedocs.io/projects/molecule/
- Blog de Stéphane Robert : https://blog.stephane-robert.info
- Playbooks Kubespray : https://github.com/kubernetes-sigs/kubespray
- Playbooks du livre "Haute disponibilité sous Linux" : https://github.com/halnx-todo/ansible-playbooks

Merci pour votre attention.

Questions ?

sebastien@vincent-netsys.fr