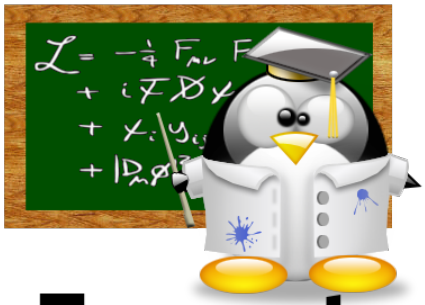

Formatux

Ansible Niveau 2

Table des matières

🎓 Objectifs

- ☐ Utiliser les variables ;
- ☐ Mettre en oeuvre des boucles et des conditions ;
- ☐ Gérer les fichiers ;
- ☐ Envoyer des notifications et réagir ;
- ☐ Gérer les fichiers ;
- ☐ Créer des tâches asynchrones.

Chapitre 1. Les variables

Plus d'informations sur http://docs.ansible.com/ansible/latest/playbooks_variables.html

Sous Ansible, il existe deux types de variables :

- la valeur simple
- le dictionnaire

Une variable peut être définie dans un playbook, dans un rôle ou depuis la ligne de commande.

Par exemple, depuis un playbook :

```
---
- hosts: apache1
  remote_user: root
  vars:
    port_http: 80
    service:
      debian: apache2
      centos: httpd
```

ou depuis la ligne de commande :

```
$ ansible-playbook deploy-http.yml --extra-vars "service=httpd"
```

Une fois définie, une variable peut être utilisée en l'appellant entre doubles accolades :

- {{ port_http }} pour la valeur simple
  - {{ service['centos'] }} ou {{ service.centos }} pour le dictionnaire.

Par exemple :

```
  tasks:
  - name: make sure apache is started
    service: name={{ service['centos'] }} state=started
```

Evidemment, il est également possible d'accéder aux variables globales d'Ansible (type d'OS, adresses IP, nom de la VM, etc.).

1.1. Externaliser les variables

Les variables peuvent être déportées dans un fichier externe au playbook, auquel cas il faudra définir ce fichier dans le playbook avec la directive `vars_files` :

```
---
- hosts: apache1
  remote_user: root
  vars_files:
    - mesvariables.yml
```

Le fichier mesvariables.yml

```
---
port_http: 80
service:
  debian: apache2
  centos: httpd
```

1.2. Afficher une variable

Pour afficher une variable, il faut activer le mode `debug` de la façon suivante :

Afficher une variable

```
- debug:
  msg: "Afficher la variable : {{ service['debian'] }}"
```

1.3. Enregistrer le retour d'une tâche

Pour enregistrer le retour d'une tâche et pouvoir y accéder plus tard, il faut utiliser le mot clef `register` à l'intérieur même de la tâche.

Utilisation d'une variable stockée

```
tasks:
- name: contenu de /home
  shell: ls /home
  register: homes

- name: affiche le premier repertoire
  debug:
    var: homes.stdout_lines[0]

- name: affiche le second repertoire
  debug:
    var: homes.stdout_lines[1]
```

Les chaînes de caractères composant la variable enregistrée sont accessibles via la valeur `stdout` (ce qui permet de faire des choses comme `homes.stdout.find("core") != -1`), de les exploiter en utilisant une boucle (voir `with_items`), ou tout simplement par leurs indices comme vu dans l'exemple précédent.

Chapitre 2. La gestion des boucles

Plus d'informations sur http://docs.ansible.com/ansible/latest/playbooks_loops.html

Il existe plusieurs types de boucles sous Ansible, en fonction de l'objet que vous voulez manipuler :

- `with_items`
- `with_file`
- `with_dict`
- `with_fileglob`
- ...

Exemple d'utilisation, création de 3 utilisateurs :

```
- name: ajouter des utilisateurs
  user:
    name: "{{ item }}"
    state: present
    groups: "users"
  with_items:
     - antoine
     - kevin
     - nicolas
```

Nous pouvons reprendre l'exemple vu durant l'étude des variables stockées pour l'améliorer :

Utilisation d'une variable stockée

```
  tasks:
  - name: contenu de /home
    shell: ls /home
    register: homes

  - name: affiche le nom des repertoires
    debug:
      msg: "Dossier => {{ item }}"
    with_items:
      - "{{ homes.stdout_lines }}"
```

Une fois un dictionnaire créé, celui-ci peut être parcouru en utilisant la variable `item` comme index :

```
---
- hosts: ansiblecli
  remote_user: ansible
  become: true
  vars:
    users:
      antoine:
        group: users
        state: present
      erwan:
        group: users
        state: absent

  tasks:

  - name: creer les utilisateurs
    user:
      name: "{{ item }}"
      group: "{{ users[item]['group'] }}"
      state: "{{ users[item]['state'] }}"
    with_items: "{{ users }}"
```

Chapitre 3. Les conditions

Plus d'informations sur http://docs.ansible.com/ansible/latest/playbooks_conditionals.html

L'instruction `when` est très pratique dans de nombreux cas : ne pas effectuer certaines actions sur certains type de serveur, si un fichier ou un n'utilisateur n'existe pas, etc.

Derrière l'instruction `when` les variables ne nécessitent pas de doubles accolades (il s'agit en fait d'expressions Jinja2...).

```
tasks:
  - name: "ne redemarre que les OS Debian"
    command: /sbin/shutdown -r now
    when: ansible_os_family == "Debian"
```

Les conditions peuvent être regroupées avec des parenthèses :

```
tasks:
  - name: "ne redémarre que les CentOS version 6 et Debian version 7"
    command: /sbin/shutdown -r now
    when: (ansible_distribution == "CentOS" and ansible_distribution_major_version ==
"6") or
          (ansible_distribution == "Debian" and ansible_distribution_major_version ==
"7")
```

Les conditions correspondant à un ET logique peuvent être fournies sous forme de liste :

```
tasks:
  - name: "ne redémarre que les CentOS 6"
    command: /sbin/shutdown -r now
    when:
      - ansible_distribution == "CentOS"
      - ansible_distribution_major_version == "6"
```

Le résultat de la variable peut aussi être utilisé conjointement aux conditions `when` et son absence de contenu évalué :

```
tasks:

- name: check if /data exists
  command: find / -maxdepth 1 -type d -name data
  register: datadir

- name: print warning if /data does not exist
  debug: msg="Le dossier /data n'existe pas..."
  when: datadir.stdout == ""
```

Chapitre 4. La gestion des fichiers

Plus d'informations sur http://docs.ansible.com/ansible/latest/list_of_files_modules.html

En fonction de votre besoin, vous allez être amenés à utiliser différents modules Ansible pour modifier les fichiers de configuration du système.

4.1. Le module `ini_file`

Lorsqu'il s'agit de modifier un fichier de type INI (section entre [] puis paires de clef=valeur), le plus simple est d'utiliser le module ini_file.

Le module nécessite :

- La valeur de la section
- Le nom de l'option
- La nouvelle valeur

Exemple d'utilisation :

```
- name: change value on inifile
  ini_file: dest=/path/to/file.ini section=SECTIONNAME option=OPTIONNAME
value=NEWVALUE
```

4.2. Le module `lineinfile`

Pour s'assurer qu'une ligne est présente dans un fichier, ou lorsqu'une seule ligne d'un fichier doit être ajoutée ou modifiée.

Voir http://docs.ansible.com/ansible/latest/lineinfile_module.html.

Dans ce cas, la ligne à modifier d'un fichier sera retrouvée à l'aide d'une regexp.

Par exemple, pour s'assurer que la ligne commençant par 'SELINUX=' dans le fichier /etc/selinux/config contiennent la valeur `enforcing` :

```
- lineinfile:
    path: /etc/selinux/config
    regexp: '^SELINUX='
    line: 'SELINUX=enforcing'
```

4.3. Le module `copy`

Lorsqu'un fichier doit être copié depuis le serveur Ansible vers un ou plusieurs hosts, dans ce cas il est préférable d'utiliser le module `copy` :

```
- copy:
    src: /data/ansible/sources/monfichier.conf
    dest: /etc/monfichier.conf
    owner: root
    group: root
    mode: 0644
```

4.4. Le module `fetch`

Lorsqu'un fichier doit être copié depuis un serveur distant vers le serveur local.

Ce module fait l'inverse du module `copy`.

```
- fetch:
    src: /etc/monfichier.conf
    dest: /data/ansible/backup/monfichier-{{ inventory_hostname }}.conf
    flat: yes
```

4.5. Le module `template`

Ansible et son module `template` utilisent le système de template Jinja2 (http://jinja.pocoo.org/docs/) pour générer des fichiers sur les hôtes cibles.

```
- template:
    src: /data/ansible/templates/monfichier.j2
    dest: /etc/monfichier.conf
    owner: root
    group: root
    mode: 0644
```

Il est possible d'ajouter une étape de validation si le service ciblé le permet (par exemple apache avec la commande `apachectl -t`) :

```
- template:
    src: /data/ansible/templates/vhost.j2
    dest: /etc/httpd/sites-available/vhost.conf
    owner: root
    group: root
    mode: 0644
    validate: '/usr/sbin/apachectl -t'
```

4.6. Le module `get_url`

Pour télécharger vers un ou plusieurs hôtes des fichiers depuis un site web ou ftp.

```
- get_url:
    url: http://site.com/archive.zip
    dest: /tmp/archive.zip
    mode: 0640
    checksum: sha256:f772bd36185515581aa9a2e4b38fb97940ff28764900ba708e68286121770e9a
```

En fournissant un checksum du fichier, ce dernier ne sera pas re-téléchargé s'il est déjà présent à l'emplacement de destination et que son checksum correspond à la valeur fournie.

Chapitre 5. Les handlers

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

Les handlers permettent de lancer des opérations, comme relancer un service, lors de changements.

Un module, étant idempotent, un playbook peut détecter qu'il y a eu un changement significatif sur un système distant, et donc déclencer une opération en réaction à ce changement. Une notification est envoyée à la fin d'un bloc de tâche du playbook, et l'opération en réaction ne sera déclenchée qu'une seule fois même si plusieurs tâches différentes envoient cette notification.

Par exemple, plusieurs tâches peuvent indiquer que le service httpd nécessite une relance à cause d'un changement dans ses fichiers de configuration. Mais le service ne sera redémarré qu'une seule fois pour éviter les multiples démarrages non nécessaires.

```
- name: template configuration file
  template: src=modele-site.j2 dest=/etc/httpd/sites-availables/site-test.conf
  notify:
     - restart memcached
     - restart httpd
```

Un handler est une sorte de tâche référencé par un nom unique global :

- Il est activé par ou un plusieurs notifiers.
- Il ne se lance pas immédiatement, mais attend que toutes les tâches soient complètes pour s'exécuter.

Exemple de handlers

```
handlers:
    - name: restart memcached
      service: name=memcached state=restarted
    - name: restart httpd
      service: name=httpd state=restarted
```

Depuis la version 2.2 d'Ansible, les handlers peuvent se mettre directement à l'écoute des tâches :

```
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"
```

Chapitre 6. Les rôles

Plus d'informations sur http://docs.ansible.com/ansible/latest/playbooks_reuse_roles.html

Un rôle Ansible est une unité favorisant la réutilisabilité des playbooks.

Un squelette de rôle, servant comme point de départ du développement d'un rôle personnalisé, peut être généré par la commande `ansible-galaxy` :

```
$ ansible-galaxy init claranet
```

La commande aura pour effet de générer l'arborescence suivante pour contenir le rôle `claranet` :

```
$ tree claranet
claranet/
├── defaults
│   └── main.yml
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Les rôles permettent de s'affranchir totalement de l'inclusion des fichiers. Plus besoin de spécifier les chemins d'accès aux fichiers, les directives `include` dans les playbook. Il suffit de spécifier une tâche, ansible s'occupe des inclusions.

6.1. La commande ansible-galaxy

La commande `ansible-galaxy` gère des rôles en utilisant le site galaxy.ansible.com.

Syntaxe de la commande ansible-galaxy

```
ansible-galaxy [import|init|install|login|remove|...]
```

Table 1. Sous-commandes de la commande ansible-galaxy

Sous-commandes	Observations
`install`	installe un rôle
`remove`	retire un ou plusieurs rôles
`init`	génère un squelette de nouveau rôle
`import`	importe un rôle depuis le site web galaxy. Nécessite un login au préalable.

Chapitre 7. Les tâches asynchrones

Plus d'informations sur http://docs.ansible.com/ansible/latest/playbooks_async.html

Par défaut, les connexions SSH vers les hosts restent ouvertes durant l'exécution des différentes tâches d'un playbook sur l'ensemble des noeuds.

Cela peut poser quelques problèmes, notamment :

- si la durée d'exécution de la tâche est supérieure au timeout de la connexion SSH
- si la connexion est interrompue durant l'action (reboot du serveur par exemple)

Dans ce cas, il faudra basculer en mode asynchrone et spécifier un temps maximum d'exécution ainsi que la fréquence (par défaut à 10s) avec laquelle vous allez vérifier le status de l'hôte.

En spécifiant une valeur de poll à 0, Ansible va exécuter la tâche et continuer sans se soucier du résultat.

Voici un exemple mettant en oeuvre les tâches asynchrones, qui permet de relancer un serveur et d'attendre que le port 22 soit de nouveau joignable :

```
# On attend 2s et on lance un reboot
- name: Reboot system
  shell: sleep 2 && shutdown -r now "Ansible reboot triggered"
  async: 1
  poll: 0
  ignore_errors: true
  become: true
  changed_when: False

  # On attend que ça revienne
  - name: Waiting for server to restart (10 mins max)
    wait_for:
      host: "{{ inventory_hostname }}"
      port: 22
      delay: 30
      state: started
      timeout: 600
    delegate_to: localhost
```

Chapitre 8. Connexion à une instance Cloud Amazon ECS

Lors de la création d'une instance Amazon, une clef privée est créée et téléchargée sur le poste local.

Ajout de la clef dans l'agent SSH :

```
ssh-add path/to/fichier.pem
```

Lancement des `facts` sur les serveurs aws :

```
ansible aws --user=ec2-user --become -m setup
```

Pour une image ubuntu, il faudra utiliser l'utilisateur ubuntu :

```
ansible aws --user=ubuntu --become -m setup
```