Support pédagogique Ansible

Avanced

- Délégation
- Tags
- Include et Import
- blocks

Délégation des tâches

Il est parfois utile de faire des actions différentes sur certains des serveurs de l'inventaire, `delegate_to` peut être utilisé pour ça.

> Illustration avec un exemple : on reprend le fichier `template-net.html.j2`, le `playbook` est lancé sur le node manager, les `facts` sont récupérés des deux serveurs (`1-1.practice` et `1-2.practice`), les actions (création de répertoire et exécution du template) sont réalisées sur le serveur `1-1.practice`.

- Le fichier de configuration à jouer `jinja4.yml`

```
- name: "Generate html file for each host"
  hosts: all
  gather_facts: yes
  vars:
    host_inventory: "centos"
    inventory_dir:  "/var/www/html/inventory"
  tasks:
    - name: "Create template directory"
      file:
        path: "{{inventory_dir}}"
        owner: "apache"
        group: "apache"
        mode: "0755"
        state: "directory"
      delegate_to: "{{host_inventory}}"
    - name: "html file generation"
      template:
        src: "template-net.html.j2"
        dest: "{{inventory_dir}}/{{inventory_hostname}}.html"
      delegate_to: "{{host_inventory}}"
```

- Exécution du `playbook`

```
$ ansible-playbook jinja4.yml
```

- Résultat des actions

```
TASK [Gathering Facts]
ok: [1-1.practice-k8s.cloud]
ok: [1-2.practice-k8s.cloud]

TASK [Create template directory]
ok: [1-1.practice-k8s.cloud]
changed: [1-2.practice-k8s.cloud -> 1-1.practice-k8s.cloud]

TASK [html file generation]
changed: [1-2.practice-k8s.cloud -> 1-1.practice-k8s.cloud]
changed: [1-1.practice-k8s.cloud]
```

- Sur le serveur Inventaire : `1-1.practice-k8s.cloud`

```
$ ansible 1-1.practice-k8s.cloud -m shell -a 'grep IP /var/www/html/inventory/*'
```

```
1-1.practice-k8s.cloud | CHANGED | rc=0 >>

/var/www/html/inventory/1-1.practice-k8s.cloud.html:     IP : 10.75.40.101
/var/www/html/inventory/1-2.practice-k8s.cloud.html:     IP : 10.76.132.143
```

- Sur le serveur `1-2.practice-k8s.cloud`, le répertoire n'a pas été créé

```
$ ansible 1-2.practice-k8s.cloud -m shell -a 'grep IP /var/www/html/inventory/*'
```

```
1-2.practice-k8s.cloud | FAILED | rc=2 >>

grep: /var/www/html/inventory/*: No such file or directorynon-zero return code
```

Délégation des tâches

> Afin d'éviter de multiplier les exécutions de taches redondantes, il est possible d'utiliser `run_once: yes` pour forcer l'exécution d'une tache une seule fois.

- Pour l'exemple précédent, la tache "*création du répertoire template*" est executée deux fois (autant de serveur que de tentatives). Ce n'est pas optimal, on perd du temps.

```
TASK [Create template directory]
ok: [1-1.practice-k8s.cloud]
changed: [1-2.practice-k8s.cloud -> 1-1.practice-k8s.cloud]
```

- En ajouter `run_once: yes` dans `jinja5.yml`

```
- name: "Generate html file for each host"
  hosts: all
  gather_facts: yes
  vars:
    host_inventory: "1-1.practice-k8s.cloud"
    inventory_dir:  "/var/www/html/inventory"
  tasks:
    - name: "Create template directory"
      file:
        path: "{{inventory_dir}}"
        owner: "apache"
        group: "apache"
        mode: "0755"
        state: "directory"
      delegate_to: "{{host_inventory}}"

       ==>  Execution une seule fois
      run_once: yes

    - name: "html file generation"
      template:
        src: "template-net.html.j2"
        dest: "{{inventory_dir}}/{{inventory_hostname}}.html"
      delegate_to: "{{host_inventory}}"
```

- Rejouer le playbook

```
$ ansible-playbook  jinja5.yml
```

- La tâche de création du répertoire est exécutée **une seule fois**.

```
TASK [Gathering Facts]
ok: [1-1.practice-k8s.cloud]
ok: [1-2.practice-k8s.cloud]

TASK [Create template directory]
ok: [1-1.practice-k8s.cloud]

TASK [html file generation]
ok: [1-1.practice-k8s.cloud]
ok: [1-2.practice-k8s.cloud -> 1-1.practice-k8s.cloud]
```

Installation d'un serveur Apache et configuration avec un fichier template

- Installation de `Apache` en reprenant les exemples précédents
- Configuration d'un alias avec un fichier `template`
- Démarre `Apache`

- Création du fichier template `inventory.conf.j2`, qui reprend les configurations à pousser sur le serveur.

```
$ cat inventory.conf.j2
```

```
Alias /inventaire /var/www/html/inventory

# Donne des droits d’accès à tout le monde
<Directory /var/www/html/inventory/>
  Order Allow,Deny
  Allow from All
</Directory>
```

- L'exécution est limitée au serveur `1-1.practice-k8s.cloud` et la copie de la configuration est effectuée via le module `template`

```
$ cat apache-inventaire.conf.yml
```

```
- name: "Apache installation"
  hosts: 1-1.practice-k8s.cloud
  tasks:
    - name: "Apache package installation"
      yum:
        name: "httpd"
        state: "present"
    - name: "Apache configuration"
      template:
        src: "inventory.conf.j2"
        dest: "/etc/httpd/conf.d/inventory.conf"
        owner: "apache"
        group: "apache"
    - name: "Start apache service"
      service:
        name: "httpd"
        state: "started"
        enabled: yes
```

- Le playbook installe le serveur **Apache2** et configure un alias `inventory` sur le serveur `1-1.practice-k8s.cloud`

```
$ ansible-playbook apache-inventaire.conf.yml
```

- Le contenu du répertoire `inventory` est bien visible sur le serveur

Les tags

Les tags

> Les `tags` vont permettre de découper l'exécution d'un `playbook` en plusieurs sections, chaque section sera repérée par des `Tags`.

- Ajout d'un champs `tags` dans le `apache-inventaire.conf.tag.yml`

Les tags

```
$ cat apache-inventaire.conf.tag.yml
```

```
- name: "Apache installation"
  hosts: 1-1.practice-k8s.cloud
  tasks:
    - name: "Apache package installation"
      yum:
        name: "httpd"
        state: "present"
    - name: "Apache configuration"
      template:
        src: "inventory.conf.j2"
        dest: "/etc/httpd/conf.d/inventory.conf"
        owner: "apache"
        group: "apache"
    - name: "Start apache service"
      # ICI un TAG
      tags: restart
      service:
        name: "httpd"
        state: "started"
        enabled: yes
```

Les tags

- Pour lister tous les `tags` d'un playbook

```
$ ansible-playbook  apache-inventaire.conf.tag.yml --list-tags
```

```
playbook: apache-inventaire.conf.tag.yml

  play 1 (1-1.practice-k8s.cloud): Apache installation  TAGS: []
      TASK TAGS: [restart]
```

Les tags

- Pour inclure ou exclure un `tags` de l'exécution du `playbook`
  - `--tags LIST_TAGS` (la liste des tags est séparée par une ',')
  - `--skip LIST_TAGS`

```
$ ansible-playbook  apache-inventaire.conf.tag.yml --tags restart
```

```
PLAY [Apache installation] ****************************************************************

TASK [Gathering Facts] ********************************************************************
ok: [1-1.practice-k8s.cloud]

TASK [Start apache service] ***************************************************************
ok: [1-1.practice-k8s.cloud]

PLAY RECAP ********************************************************************************
1-1.practice-k8s.cloud     : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Les tags / Exécution conditionnée

> Une tâche sera exécutée si une condition est vérifiée !

- On va utiliser le module `register` et `debug` pour respectivement stocker la sortie de la tâche et l'afficher (`apache-when.yml`)

```
$ cat  apache-when.yml
```

```
- name: "Apache installation"
  hosts: 1-1.practice-k8s.cloud
  tasks:
    - name: "Apache configuration"
      template:
        src: "inventory.conf.j2"
        dest: "/etc/httpd/conf.d/inventory.conf"
        owner: "apache"
        group: "apache"
 #=> register execution result for template creation
      register: apache_conf
 #=> instruction permettant de scruter le contenu de la variable
    - debug: var=apache_conf
...
```

- Lancement du `playbook`

```
$ ansible-playbook apache-when.yml
```

- Le fichier **inventaire** est déjà présent, il n'y a donc pas de changement d'état (*"changed": false*).

```
ok: [1-1.practice-k8s.cloud] => {
    "apache_conf": {
        "changed": false,
        "checksum": "d59d9bb024b6b5f9fd0d52b8090c9133f00aae4a",
        "dest": "/etc/httpd/conf.d/inventory.conf",
        "diff": {
            "after": {
                "path": "/etc/httpd/conf.d/inventory.conf"
            },
            "before": {
                "path": "/etc/httpd/conf.d/inventory.conf"
            }
        },
        [...]
```

- La condition `When` est ajoutée sur la variable `apache_conf.changed`, le redémarrage du service se fera uniquement si la configuration de **Apache** a changée. `apache-when.yml`

```
- name: "Apache installation"
  hosts: 1-1.practice-k8s.cloud
  tasks:
    - name: "Apache configuration"
      template:
        src: "inventory.conf.j2"
        dest: "/etc/httpd/conf.d/inventory.conf"
        owner: "apache"
        group: "apache"
    #=> register execution result for template creation
      register: apache_conf
    #=> instruction permettant de scruter le contenu de la variable
    - debug: var=apache_conf

    - name: "Restart apache service"
      service:
        name: "httpd"
        state: "restarted"
        restart if necessary
      when: apache_conf.changed
```

- Relance le `playbook`

```
$ ansible-playbook apache-when.yml
```

- Pas de changement constaté, donc le redémarrage du service n'est pas effectué

```
TASK [debug] *******************************************************************
ok: [1-1.practice-k8s.cloud] => {
    "apache_conf": {
        "changed": false,
[...]

TASK [Start apache service] ****************************************************
skipping: [1-1.practice-k8s.cloud]

PLAY RECAP *********************************************************************
1-1.practice-k8s.cloud     : ok=4    changed=0    unreachable=0    failed=0    skipped=1
```

Action Conditionnée au contexte des serveurs

> Parfois, en fonction du type de système ou de version présents sur le serveur, les actions à effectuer peuvent être différentes.

- Un exemple de configuration qui conditionne l'action en fonction de la version du système

```
- hosts: all
  tasks:
    - name: Upgrade packages Wheezy
      apt: upgrade=safe
      when: ansible_distribution_major_version == "7"

    - name: Upgrade packages Jessie
      shell: "apt-get update && apt-get upgrade -y"
      when: ansible_distribution_major_version == "8"
```

Include et Import

- `import*` : les déclarations sont prétraitées au moment où les playbooks sont analysés.
  - import_playbook
  - import_role
  - import_tasks

Include et Import

- `include*` : les déclarations sont traitées au fur et à mesure qu'elles sont rencontrées lors de l'exécution du playbook (permet une gestion dynamique des variables).
  - include_vars
  - include_playbook
  - include_role
  - include_tasks

Include et Import / Exemple

```
- name: Play to install packages on different os families
  hosts: dev
  gather_facts: true
  become: yes
  tasks:
     - import_tasks: install_RedHat_Pkg.yml
       when: ansible_facts.os_family == "RedHat"
     - import_tasks: install_Debian_Pkg.yml
       when: ansible_facts.os_family == "Debian"
     # It wont work - import_tasks: install_{{ansible_facts.os_family}}_Pkg.yml

     - include_tasks: install_RedHat_Pkg.yml
       when: ansible_facts.os_family == "RedHat"
     - include_tasks: install_Debian_Pkg.yml
       when: ansible_facts.os_family == "Debian"
    -  include_tasks: install_{{ansible_facts.os_family}}_Pkg.yml
```

> `include_tasks: install_{{ansible_facts.os_family}}_Pkg.yml` permet de selectionner en une ligne le playbook corespondant à la famille de l'OS. Donc preferer include à import.

Include et Import / Exemple

> includes-and-imports.yaml

```
- name: setup a service
  hosts: ansible2
  tasks:
  - name: include the services task file
    include_tasks: tasks/service.yaml
    vars:
      package: httpd
      service: httpd
    when: ansible_facts['os_family'] == 'RedHat'
  - name: include the firewall file
    import_tasks: tasks/firewall.yaml
    vars:
      firewall_package: firewalld
      firewall_service: firewalld
      firewall_rules:
      - http
      - https
```

Filtres / tests valeurs

> filter-1.yml

```
- hosts: localhost
  vars_prompt:
  - name: filesize
    prompt: "specify a file size in megabytes"
    private: no
  tasks:
  - name: check if file size is valid
    assert:
      that:
      - "{{ (filesize | int) <= 100 }}"
      - "{{ (filesize | int) >= 1 }}"
      fail_msg: "file size must be between 0 and 100"
      success_msg: "file size is good, let's continue"
  - name: debug
    debug:
      msg:  "filesize : {{ filesize }}"
```

Filtres / tests valeurs

> assert_filter.yaml

```
- name: demonstrate filters
  hosts: localhost
  tasks:
  - name: manipulate a list and check
    assert:
      that:
      - "{{ [ 2, 3, 1, 2, 4, 2 ] | sort | unique | list }} is eq( [ 1, 2, 3, 4 ] )"
```

Filtres / Modification de variables / set_fact

```
- name: showing map
  hosts: all
  gather_facts: no
  become: yes
  vars:
  - basedir: /var/www/html
  tasks:
  - name: showing basedir
    debug:
      var: basedir
  - name: showing files in basedir
    find:
      paths: "{{ basedir }}"
      recurse: yes
    register: basedir_files
  - name: showing current variable contents to understand why we need map
    debug:
      var: basedir_files
  - name: show files based on map attribute path without list - will show error
    set_fact:
      webfiles: "{{ basedir_files['files'] | map(attribute='path') }}"
  - debug:
      var: webfiles
  - name: show files based on map attribute with list
    set_fact:
      webfileslist: "{{ basedir_files['files'] | map(attribute='path') | list }}"
  - debug:
      var: webfileslist
```

Blocks

Les blocks dans Ansible sont des constructions qui permettent de regrouper un ensemble de tâches.

Avantages des Blocks

- **Gestion des erreurs :** Les blocks permettent d'appliquer des directives de gestion des erreurs sur un groupe de tâches, rendant votre playbook plus robuste et facile à débugger.
- **Exécution conditionnelle :** Vous pouvez exécuter un block de tâches basé sur une condition spécifique, ce qui simplifie la logique de vos playbooks.

Blocks

- Ansible permet de structurer ses tâches avec des blocs.
- Les `block` sont utilisés pour :
  - Organiser les tâches
  - Gérer les erreurs (`rescue`)
  - Exécuter du code de récupération (`always`)

Blocks

```
- name: Exemple de bloc
  block:
    - name: Tâche 1
      debug: msg="Dans le bloc"
```

- Structure simple pour grouper plusieurs tâches.

Blocks

```
- block:
    - name: Tâche critique
      command: /bin/false
  rescue:
    - name: Gestion de l'erreur
      debug: msg="Erreur capturée"
```

- Le bloc `rescue` est exécuté si une tâche échoue.

Blocks `always`

```
- block:
    - name: Tâche principale
      command: /bin/false
  rescue:
    - name: On gère l’erreur
      debug: msg="Erreur gérée"
  always:
    - name: Toujours exécutée
      debug: msg="Nettoyage"
```

- Le bloc `always` s'exécute toujours.

Blocks : Cas d’usage typique

- Déploiement d’un service :
  - Bloc pour les étapes critiques
  - `rescue` pour rollback
  - `always` pour notification

Blocks : Conditions et boucles dans block

```
- block:
    - name: Boucle dans un bloc
      debug: msg="Item: {{ item }}"
  loop: [1, 2, 3]
  when: some_condition
```

- `loop` et `when` sont valides à l'extérieur du `block`.

Blocks : Variables et scopes

- Variables définies dans un bloc :
  - Disponibles uniquement dans ce bloc
- Utiliser `set_fact` si persistance requise

Blocks : Exemple avancé

```
- block:
    - name: Créer utilisateur
      user: name=testuser state=present
    - name: Attribuer droits
      file: path=/home/testuser/.ssh mode=0700
  rescue:
    - name: Rollback
      user: name=testuser state=absent
  always:
    - name: Logging
      debug: msg="Tentative de création utilisateur"
```

Blocks : Meilleures pratiques

- Grouper logiquement les tâches
- Toujours utiliser `rescue` et `always` pour robustesse
- Utiliser des noms explicites

Blocks : Conclusion

- `block`, `rescue`, `always` sont essentiels pour les playbooks robustes
- Aident à créer du code maintenable et résilient
- À maîtriser pour devenir expert Ansible

Blocks : exemple

```
- name: Validate nginx installation and if not there then install it
   hosts: localhost
   gather_facts: false
   tasks:
    - block:
         - name: Validating nginx installation
           ansible.builtin.command:
             cmd: 'ls -lrt /usr/sbin/nginx'
       rescue:
         - name: Installing nginx
           ansible.builtin.yum:
             name: nginx
             state: present
           become: yes
       always:
         - name: Displaying message
           ansible.builtin.debug:
             msg: "Now we have nginx on"
    - name: Finding the status
      ansible.builtin.service_facts:
    - name: Displaying status of nginx
      ansible.builtin.debug:
        msg: "{{ansible_facts.services['nginx.service']}}"
```

Blocks : Contexte de l’exercice

Vous devez créer un playbook Ansible qui :

1. Crée un utilisateur `webadmin`
2. Crée son répertoire `.ssh`
3. Copie une clé publique dans `authorized_keys`
4. Si une erreur survient, supprimer l’utilisateur
5. Toujours écrire un log dans un fichier `/tmp/ansible.log`

Blocks : Objectif pédagogique

- Utiliser `block`, `rescue` et `always`
- Gérer les erreurs proprement
- Appliquer les conditions et bonnes pratiques
- Structurer le playbook comme un expert