Ansible Workshop - Exercises

Basics

Get to know Ansible and learn to write your first Ansible Playbooks.

6 - Run tasks multiple times

Objective

Get to know the `loop`, `with_<lookup>`, and `until` keywords to execute a task multiple times.

The `loop` keyword is not yet a full replacement for `with_<lookup>`, but we recommend it for most use cases. Both keywords achieve the same thing (although a bit differently under the hood).
You may find e.g. `with_items` in examples and the use is not (*yet*) deprecated - that syntax will still be valid for the foreseeable future - but try to use the `loop` keyword whenever possible.
The `until` keyword is used to retry a task until a certain condition is met. For example, you could run a task up to `X` times (defined by a *retries* parameter) with a delay of `X` seconds between each attempt. This may be useful if your playbook has to wait for the startup of a process before continuing.

Guide

Loops enable us to repeat the same task over and over again. For example, lets say you want to create multiple users. By using an Ansible loop, you can do that in a single task. Loops can also iterate over more than just basic lists. For example, if you have a list of users with their corresponding group, loop can iterate over them as well.

Find out more about loops in the Ansible Loops documentation.

Step 1 - Simple Loops

To show the loops feature we will generate three new users on `node1`. For that, create the file `loop_users.yml` in `~/ansible-files` on your control node as your student user. We will use the `user` module to generate the user accounts.

```
---
- name: Demo playbook for loops
  hosts: node1
  become: true
  tasks:
    - name: Ensure multiple users are present
      ansible.builtin.user:
        name: "{{ item }}"
        state: present
      loop:
        - dev_user
        - qa_user
        - prod_user
```

Understand the playbook and the output:

- The names are not provided to the user module directly. Instead, there is only a variable called `{{ item }}` for the parameter `name`.
- The `loop` keyword lists the actual user names. Those replace the `{{ item }}` during the actual execution of the playbook.
- During execution the task is only listed once, but there are three changes listed underneath it.

Step 2 - Loop complex lists

Sometimes, your list items contain multiple key-value-pairs. Imagine that the users should be assigned to different additional groups, your list may look like this:

```
- username: dev_user
  group: ftp
- username: qa_user
  group: apache
- username: prod_user
  group: admin
```

The `user` module has the optional parameter `groups` which defines the group (or list of groups) the user should be added to. To reference items in a hash, the `{{ item }}` keyword needs to reference the sub-key: `{{ item.group }}` for example.

Hint

By default, the user is **removed** from all other groups. Use the module parameter `append: true` to modify this.

Let's rewrite the playbook to create the users and also add them to specific groups:

```
---
- name: Demo playbook for loops
  hosts: node1
  become: true
  tasks:
    - name: Ensure multiple users are present
      ansible.builtin.user:
        name: "{{ item.username }}"
        state: present
        groups: "{{ item.group }}"
      loop:
        - username: dev_user
          group: ftp
        - username: qa_user
          group: apache
        - username: prod_user
          group: admin
```

Check the output:

- Again the task is listed once, but three changes are listed. Each loop item with its content is shown.

Failure

At least one user was not created because of a missing group, the playbook failed?
Well, we did not create all groups, the *user*-module does not do this! Some groups are already present, either they were present by default or were created when we installed packages, other groups must be created before we can use them.

Success

To ensure all groups are created, before you reference them, add one more task which creates the groups for you! Use the `ansible.builtin.group` module and loop over the same list as the task which creates the users, this list contains all groups which need to be created.

Need help?

```
---
- name: Demo playbook for loops
  hosts: node1
  become: true
  tasks:
    # Looping over same list as the next task, but only using/referencing the groups key
    - name: Ensure groups are present
      ansible.builtin.group:
        name: "{{ item.group }}"
        state: present
      loop:
        - username: dev_user
          group: ftp
        - username: qa_user
          group: apache
        - username: prod_user
          group: admin

    - name: Ensure multiple users are present
      ansible.builtin.user:
        name: "{{ item.username }}"
        state: present
        groups: "{{ item.group }}"
      loop:
        - username: dev_user
          group: ftp
        - username: qa_user
          group: apache
        - username: prod_user
          group: admin
```

Instead of repeating the list in the loop, you can (and should!) relocate the loop content to a variable and reference this one. Take a look at the following playbook:

```
---
- name: Demo playbook for loops
  hosts: node1
  become: true
  vars:
    user_and_group_list:
      - username: dev_user
        group: ftp
      - username: qa_user
        group: apache
      - username: prod_user
        group: admin
  tasks:
    - name: Ensure groups are present
      ansible.builtin.group:
        name: "{{ item.group }}"
        state: present
      loop: "{{ user_and_group_list }}"

    - name: Ensure multiple users are present
      ansible.builtin.user:
        name: "{{ item.username }}"
        state: present
        groups: "{{ item.group }}"
      loop: "{{ user_and_group_list }}"
```

Run the playbook again to ensure all users (and groups) are created!

Afterwards, verify that the user `prod_user` was indeed created on `node1` using the following playbook, name it `user_id.yml`:

```yaml
---
- name: Get user ID play
  hosts: node1
  vars:
    myuser: "prod_user"
  tasks:
    - name: Get info for {{ myuser }}
      ansible.builtin.getent:
        database: passwd
        key: "{{ myuser }}"

    - name: Output info for {{ myuser }}
      ansible.builtin.debug:
        msg: "{{ myuser }} uid: {{ getent_passwd[myuser][1] }}"
```

Ansible

```
$ ansible-playbook user_id.yml

PLAY [Get user ID play]
*********************************************************************************************

TASK [Gathering Facts]
*********************************************************************************************
ok: [node1]

TASK [Get info for prod_user]
*********************************************************************************************
ok: [node1]

TASK [Output info for prod_user]
*********************************************************************************************
ok: [node1] => {
    "msg": [
        "prod_user uid: 1002"
    ]
}

PLAY RECAP
*********************************************************************************************************************
node1                      : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

Navigator

Hint

It is possible to insert a *string* directly into the dictionary structure like this (although it makes the task less flexible):

```
- name: Output info for user
  ansible.builtin.debug:
    msg: "{{ myuser }} uid: {{ getent_passwd[myuser]['prod_user'][1] }}"
```

As you can see the *value* (`prod_user`) of the variable `myuser` is used directly. It must be enclosed in single quotes. You can't use normal quotation marks, as these are used outside of the whole variable.

Step 3 - Loops with *list*-variable

Up to now, we always provided the list to loop in the *loop* keyword directly, most of the times you will provide the list with a variable.

```
1  ---
2  - name: Use Ansible magic variables
3    hosts: control
4    tasks:
5      - name: Show all the hosts in the inventory
6        ansible.builtin.debug:
7          msg: "{{ item }}"
8        loop: "{{ groups['all'] }}"
9
10     - name: Show all the hosts in the current play
11       ansible.builtin.debug:
12         msg: "{{ item }}"
13       loop: "{{ ansible_play_hosts }}"
```

This playbook uses two *magic variables*, these variables cannot be set directly by the user and are always defined. The second task for example, uses the special variable `ansible_play_hosts` , which contains a **list** of hosts in the current play run, failed or unreachable hosts are excluded from this list. The first task uses the special variable `groups` , this contains a **dictionary** with all the groups in the inventory and each group has the **list** of hosts that belong to it.
Copy the contents to a file `special-variables.yml` and run the playbook.
We can use the playbook to display that the *loop* keyword needs *list*-input, if you provide otherwise, Ansible will display an error message.

```
fatal: [node1]: FAILED! => {"msg": "Invalid data passed to 'loop', it requires a list, got this
instead: {'all': ['node1', 'node2', 'node3'], 'ungrouped': [], 'web': ['node1', 'node2',
'node3']}. Hint: If you passed a list/dict of just one element, try adding wantlist=True to your
lookup invocation or use q/query instead of lookup."}
```

You can provoke this, if you change line 8 to `loop: "{{ groups }}"` . With that change you would try to loop a dictionary, this obviously fails.