Ansible Workshop - Exercises

Basics

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

4 - Using Variables

Objective

Ansible supports variables to store values that can be used in Ansible playbooks. Variables can be defined in a variety of places and have a clear precedence. Ansible substitutes the variable with its value when a task is executed.

This exercise covers variables, specifically

- How to use variable delimiters `{{` and `}}`
- What `host_vars` and `group_vars` are and when to use them
- How to use `ansible_facts`
- How to use the `debug` module to print variables to the console window

Guide

Variables are referenced in Ansible Playbooks by placing the variable name in double curly braces:

```
This string contains a {{ variable }}
```

Variables and their values can be defined in various places: the inventory, additional files, on the command line, etc.

The recommended practice to provide variables in the inventory, is to define them in files located in two directories: `host_vars` and `group_vars`.

- To define variables for a group "servers", a YAML file named `group_vars/servers.yml` with the variable definitions is created.
- To define variables specifically for a host `node1`, the file `host_vars/node1.yml` with the variable definitions is created.

Tip

Host variables take precedence over group variables (more about precedence can be found in the docs).

Step 1 - Create Variable Files

For understanding and practice let's do a lab. Following up on the theme "Let's build a web server. Or two. Or even more…", you will change the `index.html` to show the development environment (dev/prod) a server is deployed in.

On the ansible control host, as the `student` user, create the directories to hold the variable definitions in `~/ansible_files/`:

```
[student@ansible-1 ansible_files]$ mkdir host_vars group_vars
```

Now create two files containing variable definitions. We'll define a variable named `stage` which will point to different environments, `dev` or `prod`:

- Create the file `~/ansible_files/group_vars/web.yml` with this content:

```
---
stage: dev
```

- Create the file `~/ansible_files/host_vars/node2.yml` with this content:

```
---
stage: prod
```

What is this about?

- For all servers in the `web` group the variable `stage` with value `dev` is defined. So as default we flag them as members of the dev environment.
- For server `node2` this is overridden and the host is flagged as a production server.

Step 2 - Create web.html Files

Now create two files in `~/ansible_files/files/`:

One called `prod_web.html` with the following content:

```
<body>
<h1>This is a production webserver, take care!</h1>
</body>
```

And the other called `dev_web.html` with the following content:

```
<body>
<h1>This is a development webserver, have fun!</h1>
</body>
```

Step 3 - Create the Playbook

Now you need a Playbook that copies the prod or dev `web.html` file - according to the "stage" variable.

Create a new Playbook called `deploy_index_html.yml` in the `~/ansible_files/` directory.

Tip

Note how the variable "stage" is used in the name of the file to copy.

```yaml
---
- name: Webserver configuration
  hosts: web
  become: true
  tasks:
    - name: Copy web.html
      ansible.builtin.copy:
        src: "{{ stage }}_web.html"
        dest: /var/www/html/index.html
        mode: "0644"
        owner: apache
        group: apache
```

- Run the Playbook:

Ansible

```
[student@ansible-1 ansible_files]$ ansible-playbook deploy_index_html.yml
```

Navigator

```
[student@ansible-1 ansible_files]$ ansible-navigator run deploy_index_html.yml
```

Step 4 - Test the Result

The Ansible Playbook copies different files as index.html to the hosts, use `curl` to test it.

For node1:

```
[student@ansible-1 ansible_files]$ curl http://node1
<body>
<h1>This is a development webserver, have fun!</h1>
</body>
```

For node2:

```
[student@ansible-1 ansible_files]$ curl http://node2
<body>
<h1>This is a production webserver, take care!</h1>
</body>
```

For node3:

```
[student@ansible-1 ansible_files]$ curl http://node3
<body>
<h1>This is a development webserver, have fun!</h1>
</body>
```

Tip

If by now you think: There has to be a smarter way to change content in files… you are absolutely right. This lab was done to introduce variables, you are about to learn about templates in one of the next chapters.

Step 5 - Ansible Facts

Ansible facts are variables that are automatically discovered by Ansible from a managed host. Remember the "Gathering Facts" task with the *setup* module we used with the Ansible ad hoc command?
The facts contain useful information stored into variables that administrators can reuse.

To get an idea what facts Ansible collects by default, on your control node as your student user create the playbook `setup.yml` and run it to get the setup details of `node1`:

```
---
- name: Capture and output facts
  hosts: node1
  gather_facts: false
  tasks:
    - name: Collect only hardware facts, do not gather any other facts
      ansible.builtin.setup:
        gather_subset:
          - '!all'
          - '!min'
          - hardware

    - name: Output facts content
      ansible.builtin.debug:
        msg: "{{ ansible_facts }}"

    - name: Get python interpreter with command and store as variable during runtime
      ansible.builtin.command: which python3
      register: command_output

    - name: Output python interpreter path
      ansible.builtin.debug:
        msg:
          - "Python3 interpreter path from command output: {{ command_output.stdout }}"
          - "Python3 interpreter path from ansible fact: {{
ansible_facts.discovered_interpreter_python }}"
```

Ansible

```
[student@ansible-1 ansible_files]$ ansible-playbook setup.yml
```

Navigator

```
[student@ansible-1 ansible_files]$ ansible-navigator run setup.yml -m stdout
```

This might be a bit too much, you can use filters to limit the output to certain facts, the expression is shell-style wildcard within your playbook. Create a playbook labeled `setup_filter.yml` as shown below. In this example, we filter to get the `eth0` facts as well as memory details of `node1`.

```
---
- name: Capture and output facts
  hosts: node1
  gather_facts: false
  tasks:
    - name: Collect only specific facts
      ansible.builtin.setup:
        filter:
          - 'ansible_eth0'
          - 'ansible_*_mb'

    - name: Output variable content
      ansible.builtin.debug:
        msg: "{{ ansible_facts }}"
```

Run the playbook:

Ansible

```
[student@ansible-1 ansible_files]$ ansible-playbook setup_filter.yml
```

Navigator

```
[student@ansible-1 ansible_files]$ ansible-navigator run setup_filter.yml -m stdout
```

Step 6 - Challenge Lab: Facts

- Try to find and print the OS **family** (RedHat) of your managed hosts using a playbook, it should output only this single fact.

Tip

Use an *ad-hoc* command to output all facts, maybe even filter the output by using `grep` to find the appropriate fact.

Solution

```
$ ansible node1 -m setup | grep family
    "ansible_os_family": "RedHat",
```

```
---
- name: Capture and output facts
  hosts: node1
  gather_facts: false
  tasks:
    - name: Collect only specific facts, this task can be removed when enabling 'gather_facts' again.
      ansible.builtin.setup:
        filter:
          - '*family'

    - name: Output variable content
      ansible.builtin.debug:
        msg: "{{ ansible_os_family }}"
```

Run the playbook:

```
[student@ansible-1 ansible_files]$ ansible-playbook setup_filter.yml
```

Optionally, run the playbook with the *ansible-navigator*:

```
[student@ansible-1 ansible_files]$ ansible-navigator run setup_filter.yml -m stdout
```

Step 7 - Using Facts in Playbooks

Facts can be used in a Playbook like variables, using the proper naming, of course. Create this Playbook as `facts.yml` in the `~/ansible_files/` directory:

```
---
- name: Output facts within a playbook
  hosts: all
  tasks:
    - name: Prints Ansible facts
      ansible.builtin.debug:
        msg: From a total of {{ ansible_memtotal_mb }} MB the server {{ ansible_fqdn }} has {{ ansible_memfree_mb }} MB RAM left.
```

Tip

The "debug" module is handy for e.g. debugging variables or expressions.

Execute it to see how the facts are printed:

Ansible

Examine the output:

```
PLAY [Output facts within a playbook]
*****************************************************************************

TASK [Gathering Facts]
********************************************************************************************
ok: [node2]
ok: [node3]
ok: [node1]
ok: [ansible-1]

TASK [Prints Ansible facts]
***************************************************************************************
ok: [ansible-1] => {
    "msg": "From a total of 7937 MB the server ansible-1 has 2856 MB RAM left."
}
ok: [node1] => {
    "msg": "From a total of 7937 MB the server node1 has 3152 MB RAM left."
}
ok: [node2] => {
    "msg": "From a total of 7937 MB the server node2 has 3138 MB RAM left."
}
ok: [node3] => {
    "msg": "From a total of 7937 MB the server node3 has 3247 MB RAM left."
}

PLAY RECAP
************************************************************************************************
********
ansible-1                  : ok=2    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
node1                      : ok=2    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
node2                      : ok=2    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
node3                      : ok=2    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```