Automation

Ansible Variables

DE HOGESCHOOL MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

Playbook Variables

- There are many different ways to define variables for use in tasks.
- Variables can be passed through the CLI when calling ansible-playbook, using the --extra-vars option: ansible-playbook example.yml --extra-vars "foo=bar".
- You can also pass additional variables using quoted JSON, YAML, or even by passing a JSON or YAML file, such as --extra-vars "@even_more_vars.json" or --extra-vars "@even_more_vars.yml".
- Variables can be included inline with the rest of a playbook, in a vars section.

Examples of inline variables and using a file

Ubuntu-22.04 > home > tomc > ansible-vault-test > vars-1.yml

```yaml
---
hosts: example

vars:
  foo: bar

tasks:
  # Prints "Variable 'foo' is set to bar".
  debug:
    msg: "Variable 'foo' is set to {{ foo }}"\
```

Ubuntu-22.04 > home > tomc > ansible-vault-test > vars-2.yml

```yaml
---
# Main playbook file.
- hosts: example

vars_files:
  - vars.yml

tasks:
  - debug:
    msg: "Variable 'foo' is set to {{ foo }}"
```

```
tomc @ DESKTOP-TOMC :: 20:38:28 :: ~/ansible-vault-test
$ ansible-playbook vars-1.yml

PLAY [Playbook] ************************************************************************************************

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

TASK [print variable] ******************************************************************************************
ok: [localhost] => {
    "msg": "Variable 'foo' is set to bar"
}

PLAY RECAP *****************************************************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

tomc @ DESKTOP-TOMC :: 20:38:40 :: ~/ansible-vault-test
$ _
```

Ubuntu-22.04 > home > tomc > ansible-vault-test > vars.yml

```yaml
---
# Variables file 'vars.yml' in the same folder
# as the playbook.
foo: bar
```

Advanced: conditional variable file import

- Variable files can also be conditionally imported.
- E.g. you have one set of variables for RHEL servers (where the Apache service is called **httpd**) and another for Debian servers (where the Apache service is called **apache2**). In this case, you can conditionally include variable files with include_vars.
- If you are following the example, add two files in the same directory as your playbook: **apache_RedHat.yml** and **apache_default.yml**. Define the variable **apache_service_name: httpd** in the CentOS file and **apache_service_name: apache2** in the default file.
- Ansible stores the OS of the server in the predefined **variable ansible_os_family**, and will include the vars file with the resulting name.
- If Ansible cannot find a file with that name, it will use the variables loaded from **apache_default.yml**, using **with_first_found**. So on a Debian or Ubuntu server, Ansible would correctly use apache2 as the service name, even if there is no apache_Debian.yml or apache_Ubuntu.yml file available.

Ubuntu-22.04 > home > tomc > ansible-vault-test > vars-3.yml

```yaml
---
- hosts: example
  pre_tasks:
    - include_vars: "{{ item }}"
      with_first_found:
        - "apache_{{ ansible_os_family }}.yml"
        - "apache_default.yml"

  tasks:
    - name: Ensure Apache is running.
      service:
        name: "{{ apache_service_name }}"
        state: running
```

Inventory variabelen

- Variables can also be added in Ansible inventory files, either inline with a host definition or in a group.

Ubuntu-22.04 > home > tomc > ansible-vault-test > prod-servers.ini

```
1 # Host-specific variables (defined inline).
2 2 [washington]
3 3 app1.example.com proxy_state=present
4 4 app2.example.com proxy_state=absent
5 5
6 6 # Variables defined for the entire group.
7 7 [washington:vars]
8 8 cdn_host=washington.static.example.com
9 9 api_version=3.0.1
```

Ubuntu-22.04 > home > tomc > ansible-vault-test > app1.example.com

```
1 ---
2 foo: bar
3 baz: qux
```

Inventory variables

- You can also use group_vars and host_vars YAML variable files within a specific path, and Ansible will assign them to individual hosts and groups defined in your inventory.
- For example, to apply a set of variables to the host **app1.example.com**, create an empty file named **app1.example.com** at the location **/etc/ansible/host_vars/app1.example.com**, and add variables cf. an included vars_files YAML file.
- To apply a set of variables to the entire **washington group**, create a similar file in the location **/etc/ansible/group_vars/washington**.
- **You can also place these files (with the same name) in host_vars or group_vars directories in the directory of your playbook.** Ansible will use the variables defined in the inventory /etc/ansible/[host|group]_vars directory first (if the appropriate files exist), then it will use variables defined in the playbook directories.

Using variables

- Plain variables (collected by Ansible, defined in inventory files, or defined in playbook or variable files) can be used as part of a task with a syntax like **`{{ variable }}`**.
- Eg:

```
- command: "/opt/my-app/rebuild {{ my_environment }}"
```

- When the command is executed, Ansible replaces the contents of **`my_environment`** in **`{{ my_environment }}`**. So the resulting command would be something like **`/opt/my-app/rebuild dev`**.

lists

- Many variables you will use are structured as arrays (or 'lists').

```
foo_list:
- one
- two
- three
```

- Access to list elements:
- `foo[0]`
- `foo|first`

Complex variables

- For larger and more structured arrays (e.g. when retrieving the IP address of the server using the facts Ansible collects from the server), you can access any part of the array, either using parentheses `[' ']` or dot `.` syntax.
- If you know the general structure of the variable, you can use one of the following techniques to retrieve just the server's IPv4 address:

```
{{ ansible_eth0.ipv4.address }}
{{ ansible_eth0['ipv4']['address'] }}
```

Ubuntu-22.04 > home > tomc > ansible-vault-test > vars-4.yml

```yaml
---
  - name: Playbook
    hosts: database

    vars:
      foo: bar

    tasks:
      - name: prints eth0 variable
        debug:
          var: ansible_eth0
```

```
tomc @ DESKTOP-TOMC :: 20:43:15 :: ~/ansible-vault-test
$ ansible-playbook vars-4.yml

PLAY [Playbook] ************************************************************************************************************************

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

TASK [prints eth0 variable] ************************************************************************************************************
ok: [localhost] => {
    "ansible_eth0": {
        "active": true,
        "device": "eth0",
        "ipv4": {
            "address": "172.22.189.234",
            "broadcast": "172.22.191.255",
            "netmask": "255.255.240.0",
            "network": "172.22.176.0",
            "prefix": "20"
        },
        "ipv6": [
            {
                "address": "fe80::215:5dff:fe5c:3427",
                "prefix": "64",
                "scope": "link"
            }
        ],
        "macaddress": "00:15:5d:5c:34:27",
        "module": "hv_netvsc",
        "mtu": 1500,
        "pciid": "b3370019-24ed-4275-afd8-fa442e15d3eb",
        "promisc": false,
        "speed": 10000,
        "type": "ether"
    }
}

PLAY RECAP *****************************************************************************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

tomc @ DESKTOP-TOMC :: 20:43:45 :: ~/ansible-vault-test
$ _
```

Automatically loaded group_vars and host_vars

- Ansible searches the same directory as your inventory file (or in **`/etc/ansible`** if you use the default inventory file in **`/etc/ansible/hosts`**) for two specific **directories**: **group_vars** and **host_vars**.
- You can place YAML files in these directories, with the filename being the group name or host name defined in your inventory file.
- This is useful when you want to package your entire playbook and infrastructure configuration (including all host/group specific configuration) into a source control repository.

Ubuntu-22.04 > home > tomc > ansible-vault-test > group_vars > group1

```
1 ---
2 # File: ./group_vars/group1
3 admin_user: john
```

Ubuntu-22.04 > home > tomc > ansible-vault-test > hosts_vars > host1

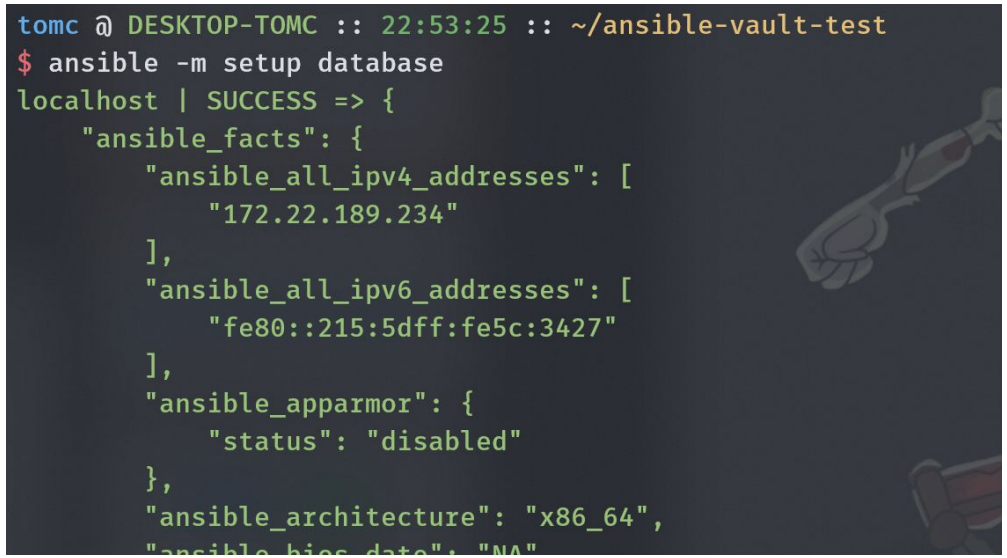
```
1 ---
2 # File: ./host_vars/host1
3 admin_user: jane
```

Magic variables using host and group variables

- If you need to get the variables of a specific host from another host, Ansible provides a magic **`hostvars`** variable that contains all defined host variables (from inventory files and all discovered YAML files in host_vars directories).
    - ```
      # works from any host, returns "jane".
      {{ hostvars['host1']['admin_user'] }}
      ```
- **`groups`**: a list of all group names in the inventory
- **`group_names`**: a list of all groups to which the current host belongs
- **`inventory_hostname`**: the hostname of the current host, according to the inventory
- **`inventory_hostname_short`**: the first part of inventory_hostname, up to and including the first point
- **`play_hosts`**: all hosts on which the current play is running.
- see
- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_vars_facts.html#information-about-ansible-magic-variables
      ```

Facts (variables derived from system information)

- When you run an Ansible playbook, by default Ansible first gathers information or **facts** about each host in the playbook. ("GATHERING FACTS")
- To get a list of all the gathered facts available, you can use the ansible command with the setup module.
  - **`ansible -m setup ..`**
- Often, playbooks use facts such as **`ansible_os_family`**, **`ansible_hostname`**, and **`ansible_memtotal_mb`** to register new variables or in combination with when, to determine whether certain tasks should be run.



Variables precedence

1. --extra-vars passed in via the command line (these always win, no matter what).
2. Task-level vars (in a task block).
3. Block-level vars (for all tasks in a block).
4. Role vars (e.g. [role]/vars/main.yml) and vars from include_vars module.
5. Vars set via set_facts modules.
6. Vars set via register in a task.
7. Individual play-level vars: 1. vars_files 2. vars_prompt 3. vars
8. Host facts.
9. Playbook host_vars.
10. Playbook group_vars.
11. Inventory: 1. host_vars 2. group_vars 3. vars
12. Role default vars (e.g. [role]/defaults/main.yml).

Variables good practices

- Roles should provide sane default values via the role's 'defaults' variables. These variables will be the fallback in case the variable is not defined anywhere else in the chain.
- Playbooks should rarely define variables (e.g. via set_fact), but rather, **variables should be defined in included vars_files** or, less often, via inventory.
- Only truly host- or group-specific variables should be defined in host or group inventories.
- Dynamic and static inventory sources should contain a minimum of variables, especially as these variables are often less visible to those maintaining a particular playbook.
- Command line variables (**-e**) should be avoided when possible. One of the main use cases is when doing local testing or running one-off playbooks where you aren't worried about the maintainability or idempotence of the tasks you're running.