Managing Variables

Introduction to Ansible Variables

- Ansible Supports variables that can be used to store values that can then be reused throughout files in an **Ansible project**.
- This can simplify the creation and maintenance of a project and reduce the number of errors.
- Variables provide a convenient way to manage dynamic values for a given environment in your Ansible project.
- Examples of values that variables might contain include:
  - Users to create
  - Packages to install
  - Services to restart
  - Files to remove
  - Archives to retrieve from the internet

Naming Variables

- Variable names must start with letter, and they can only contain letters, numbers, and underscores.
- The following table illustrates the difference between invalid variable names.
- **Example of invalid and valid Ansible Variable Names**

Invalid variable names	Valid variable names
Web server	web_server
remote.file	remote_file
1st file	file_1 file1
remoteserver$1	remote_server_1 remote_server1

Defining Variables

Variables can be defined in a variety of places in an **Ansible project**. If a variable is set using the same name in **two places**, and those settings have different values, precedence determines which value is used.

Here is a simplified list of ways to define a variable, ordered from lowest precedence to highest:

- Group variables defined in the inventory.
- Group variables defined in files in a **group_vars** subdirectory in the same directory as the inventory or the playbook.
- Host variables defined in the inventory.

Defining Variables

- Host variables defined in files in a **host_vars** subdirectory in the **same directory** as the **inventory** or the **playbook**.
- **Host facts**, discovered at runtime.
- Play variables in the playbook (**vars** and **vars_files**).
- Task variables.
- Extra variables defined on the command line.

a variable that is set to affect the all host group will be overridden by a variable that has the same name and is set to affect a single host.

sometimes you might want to use precedence to cause different hosts or host groups to get different settings than your defaults.

If the same variable name is defined at more than one level, the level with the highest precedence wins.

Defining Variables

- A narrow scope, such as a host variable or a task variable, takes precedence over a wider scope, such as a group variable or a play variable.
- Variables defined by the inventory are overridden by variables defined by the playbook.
- Extra variables defined on the command line with the **--extra-vars**, or **-e**, option have the highest precedence.

Variables in Playbooks

Variables play an important role in Ansible Playbooks because they ease the management of variable data in a playbook.

- **Defining Variables in Playbooks**

When writing playbooks, you can define your own variables and then invoke those values in a task.

For Example, a variable named **web_package** can be defined with a value of httpd. A task can then call the variable using **yum** module to install the httpd package.

Playbook variables can be defined in multiple ways. One common method is to place a variable in a **vars** block at the beginning of the playbook.

```
- hosts: all
 vars:
   user: joe
   home: /home/joe
```

It is also possible to define playbook variables in external files. In this case, instead of using a vars
block in the playbook, the vars_files directive may be used, followed by a list of names for
external variable files relative to the location of the playbook:

```
- hosts: all
  vars_files:
    - vars/users.yml
```

- The playbook variables are then defined in that file or those files in YAML format:

```
user: joe
home: /home/joe
```

Using Variables in Playbooks

- After variables have been declared, administrators can use the variables in tasks.
- Variables are referenced by placing the variable name in double curly braces ({{}}).
- Ansible substitutes the variable with its value when the task is executed

```
vars:
  user: joe

tasks:
  # This line will read: Creates the user joe
  - name: Creates the user {{ user }}
    user:
      # This line will create the user named Joe
      name: "{{ user }}"
```

Note: when a variable is used as the first element to start a value, quotes are mandatory.

This prevents ansible from interpreting the variable reference as starting a YAML dictionary

For Example, if quotes are missing :

```
yum:
    name: {{ service }}
           ^ here
We could be wrong, but this one looks like it might be an issue with
missing quotes. Always quote template expression brackets when they
start a value. For instance:

    with_items:
      - {{ foo }}
Should be written as:

    with_items:
      - "{{ foo }}"
```

Host Variables and Group Variables

Inventory variables that apply directly to hosts fall into two brands:

- **Host Variables**
- **Group Variables**

Host Variables

Host variables apply to specific hosts

- Defining the ansible_user host variable for demo.example.com:

```
[servers]
demo.example.com ansible_user=joe
```

Group Variables

Group variables apply to all hosts in a host group or in a group of hosts.

- Defining the user group variable for the servers host group.

```
[servers]
demo1.example.com
demo2.example.com

[servers:vars]
user=joe
```

Using Directories to Populate Host and Group Variables

- The preferred approach to defining variables for hosts and host groups is to create two directories, **group_vars** and **host_vars**, in the same working directory as the inventory file or playbook.
- These directories contain files defining group variables and **host variables**, respectively
- To define group variables for the **servers group**, you would create a **YAML file** named **group_vars/servers**, and then the contents of that file would set variables to values using the same syntax as in a playbook:

```
user: joe
```

- To define host variables for a particular host, create a file with a name matching the host in the host_vars directory to contain the host variables.

- to define host variables for a particular host, create a file with a name matching the host in the **host_vars** directory to contain the host variables.

```
[root@ansimaster ~]# cat ~/project/inventory
[datacenter1]
demo1.example.com
demo2.example.com

[datacenter2]
demo3.example.com
demo4.example.com

[datacenters:children]
datacenter1
datacenter2
```

- If you need to define a general value for all servers in both data centers, set a group variable for the datacenters host group:

```
[root@ansimaster ~]# cat ~/project/group_vars/datacenters
package: httpd
```

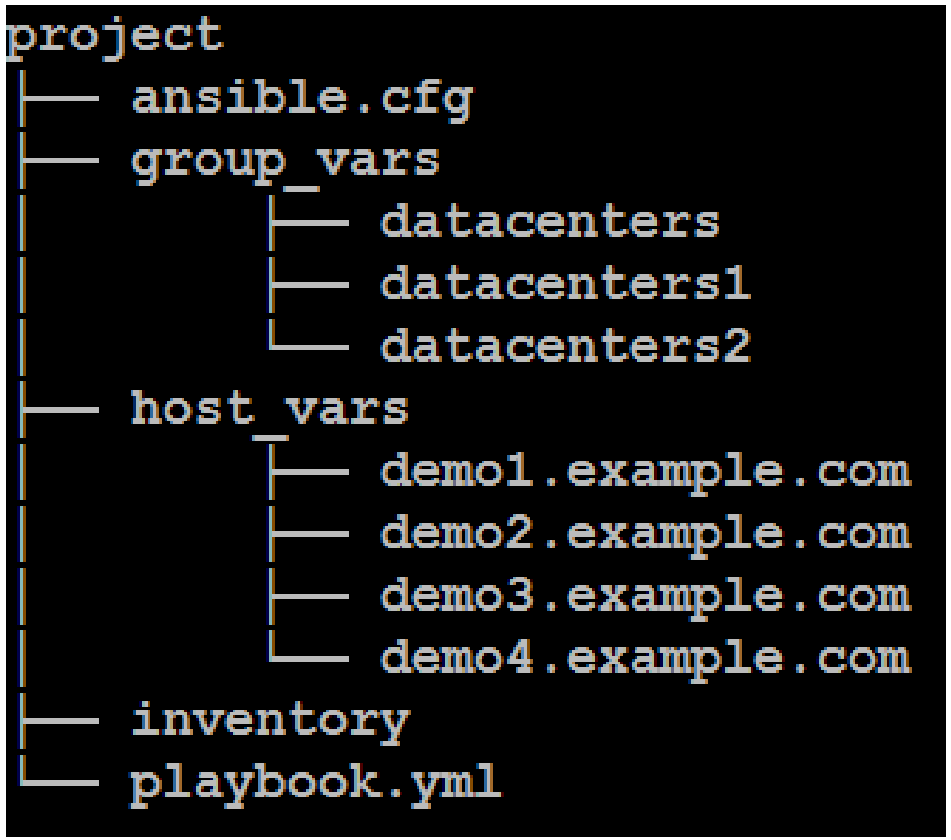
- If the value to define varies for each data center, set a group variable for each data center host group:

```
[root@ansimaster ~]# cat ~/project/group_vars/datacenter1
package: httpd
[root@ansimaster ~]# cat ~/project/group_vars/datacenter2
package: apache
```

- If the value to be defined varies for each host in every data center, then define the variables in separate host variable files:

```
[root@ansimaster ~]# cat ~/project/host_vars/demo1.example.com
package: httpd
[root@ansimaster ~]# cat ~/project/host_vars/demo2.example.com
package: apache
[root@ansimaster ~]# cat ~/project/host_vars/demo3.example.com
package: mariadb-server
[root@ansimaster ~]# cat ~/project/host_vars/demo4.example.com
package: mysql-server
```

- The directory structure for the example project, project, if it contained all the example files above, would appear as follows:


project
ansible.cfg
group_vars
datacenters
datacenters1
datacenters2
host_vars
demo1.example.com
demo2.example.com
demo3.example.com
demo4.example.com
inventory
playbook.yml

Overriding Variables from the Command Li

- Inventory variables are **overridden** by variables set in a **playbook**, but both kinds of variables may be overridden through **arguments** passed to the **ansible** or **ansible-playbook** commands on the command line. Variables set on the command line are called extra variables.
- Extra variables can be useful when you need to override the defined value for a variable for a oneoff run of a playbook. For example:

```
[user@demo ~]$ ansible-playbook main.yml -e "package=apache"
```

Using Arrays as Variables

- Instead of assigning configuration data that relates to the same element (**a list of packages**, **a list of services**, **a list of users**, and so on), to multiple variables, administrators can use arrays.
- One consequence of this is that an array can be browsed.
- For example, consider the following snippet:

```
user1_first_name: Bob
user1_last_name: Jones
user1_home_dir: /users/bjones
user2_first_name: Anne
user2_last_name: Cook
user2_home_dir: /users/acook
```

Using Arrays as Variables

- This could be rewritten as an array called users:

```
user1_first_name: Bob
user1_last_name: Jones
user1_home_dir: /users/bjones
user2_first_name: Anne
user2_last_name: Cook
user2_home_dir: /users/acook
```

- You can then use the following variables to access user data:

```
# Returns 'Bob'
users.bjones.first_name

# Returns '/users/acook'
users.acook.home_dir
```

- Because the variable is defined as a Python *dictionary*, an alternative syntax is available.

```
# Returns 'Bob'
users['bjones']['first_name']

# Returns '/users/acook'
users['acook']['home_dir']
```

Capturing Command Output with Registered Variables

You can use the register statement to capture the output of a command.

The output is saved into a **temporary variable** that can be used later in the **playbook** for either **debugging purposes** such as a particular configuration based on a command's output.

The following playbook demonstrates how to capture the output of a command for debugging purposes:

```
---
- name: Installs a package and prints the result
hosts: all
tasks:
  - name: Install the package
    yum:
      name: httpd
      state: installed
      register: install_result

    - debug:
        var: install_result
```

When you run the playbook, the debug module is used to dump the value of the **install_result** registered variable to the terminal.

```
ansible-playbook playbook.yml
PLAY [Installs a package and prints the result] ****************************

TASK [setup] ***************************************************************
ok: [demo.example.com]

TASK [Install the package] *************************************************
ok: [demo.example.com]

TASK [debug] ***************************************************************
ok: [demo.example.com] => {
    "install_result": {
    "changed": false, "msg": "",
    "rc": 0,
    "results": [
        "httpd-2.4.6-40.el7.x86_64 providing httpd is already installed"
    ]
  }
}

PLAY RECAP *****************************************************************
demo.example.com        : ok=3  changed=0       unreachable=0   failed=0
```

```
[root@ansimaster ~]# ansible demo1.example.com -m setup
demo1.example.com | SUCCESS => {
    "ansible_facts": {
...output omitted...
        "ansible_local": {
            "custom": {
                "packages": {
                    "db_package": "mariadb-server",
                    "web_package": "httpd"
                },
                "users": {
                    "user1": "joe",
                    "user2": "jane"
                }
            }
        },
...output omitted...
    },
    "changed": false
}
```

Using Magic Variables

Some variables are not facts or configured through the setup module, but are also automatically set by Ansible. These magic variables can also be useful to get information specific to a particular managed host.

Four of the most useful are:

- Hostvars
- group_names
- Groups
- inventory_hostname

- **hostvars:** Contains the variables for managed hosts, and can be used to get the values for another managed host's variables. It does not include the managed host's facts if they have not yet been gathered for that host.
- **group_names:** Lists all groups the current managed host is in.
- **groups:** Lists all groups and hosts in the inventory.
- **inventory_hostname:** Contains the host name for the current managed host as configured in the inventory. This may be different from the host name reported by facts for various reasons.

```
[root@ansimaster ~]# ansible localhost -m debug -a 'var=hostvars["localhost"]'
localhost | SUCCESS => {
    "hostvars[\"localhost\"]": {
        "ansible_check_mode": false,
        "ansible_connection": "local",
        "ansible_diff_mode": false,
        "ansible_facts": {},
        "ansible_forks": 5,
        "ansible_inventory_sources": [
            "/home/student/demo/inventory"
        ],
        "ansible_playbook_python": "/usr/bin/python2",
        "ansible_python_interpreter": "/usr/bin/python2",
        "ansible_verbosity": 0,
        "ansible_version": {
            "full": "2.7.0",
            "major": 2,
            "minor": 7,
            "revision": 0,
            "string": "2.7.0"
        },
        "group_names": [],
        "groups": {
            "all": [
                "serverb.lab.example.com"
            ],
            "ungrouped": [],
            "webservers": [
                "serverb.lab.example.com"
            ]
         },
         "inventory_hostname": "localhost",
         "inventory_hostname_short": "localhost",
         "omit": " omit_place_holder 18d132963728b2cbf7143dd49dc4bf5745fe5ec3",
         "playbook_dir": "/home/student/demo"
     }
}
```