Ansible Tutorial Part 3 – Introduction to Ansible Variables

Introduction to Ansible Variables

Ansible supports variables that can be used to store values than 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 a dynamic values for a given environment in your Ansible project. Example of value that variables might contain include –

- Users to create.
- Packages to install.
- Services to restart.
- Files to remove.
- Archives to retrieve from the Internet.

Variable names must start with a letter and they can only contain letters, numbers, and underscores.

Invalid Variable names	Valid Variable names
web server	web_server
remote.file	remote_file
1st file	file_1 OR file1
remoteserver$1	remote_server_1 OR remote_server1

Defining Variables

Variables can be defined in a variety of places in an Ansible project. However, this can be simplified to three basic scope levels –

- **Global Scope –** Variables set from the command line or Ansible configuration.
- **Play Scope –** Variables set in the play and related structures.

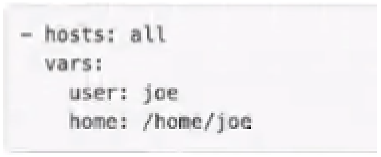
- **Host Scope –** Variables set on host groups and individual hosts by the inventory, fact gathering or registered tasks.

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

Defining Variables in Playbook

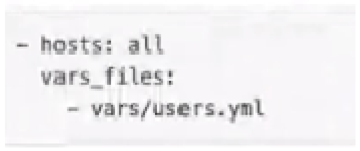
When writing playbooks, you can define your own variables and then invoke those values in a task. For example , a variable named web_package can be defined with a value of httpd. A task can then call the variable using the yum module to install the httpd package.

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



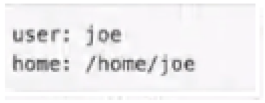
```
- hosts: all
  vars:
    user: joe
    home: /home/joe
```

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



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

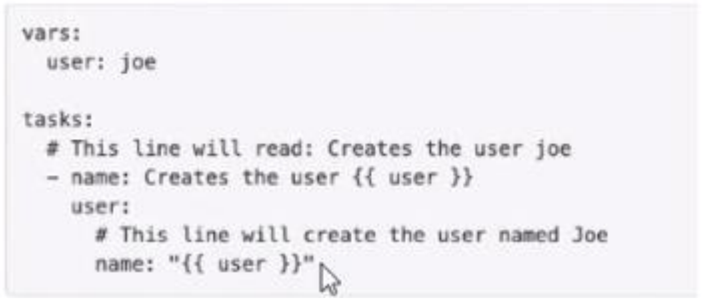
The playbook variables are then defined in that file or those files in YAML format.



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

Using Variables in Playbook

After variables have been declared, administrators can use the variables in the tasks. Variables are referenced by placing the variable name in double curl braces {{ }}. Ansible substitutes the variable with its value when the task is executed. This is a Jinja2 format.



```
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 }}"
```

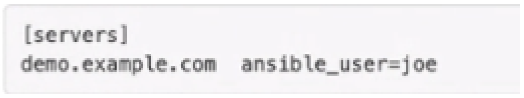
Important – When a variable is used as the first element to start a value, quotes are mandatory. This prevents Ansible from interpreting the variable reference as starting a YAML directory.

Host Variables and Group Variables

Inventory variables that apply directly to hosts fall into two broad categories – host variables apply to a specific host and group variables apply to all hosts in a host group or in a group of host groups. Host variable takes precedence over group variable, but variables defined by a playbook takes precedence over both.

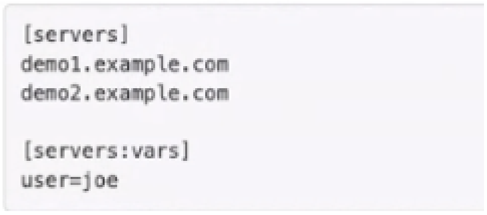
One way to define host variables and group variables is to do it directly from the inventory file. This is an older approach and not preferred but still you may encounter it.

1. Defining the **ansible_user** host variable for demo.example.com



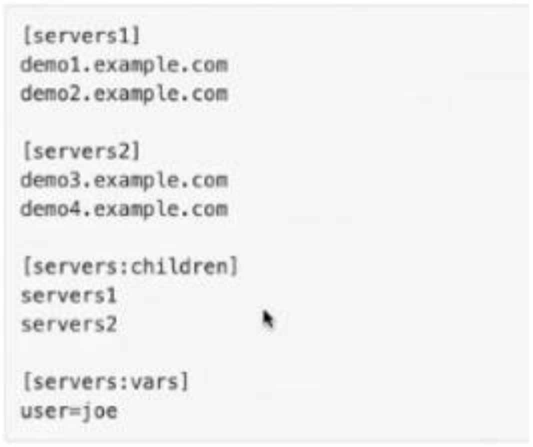
```
[servers]
demo.example.com  ansible_user=joe
```

2. Defining the **user** group variable for the **servers** host group



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

[servers:vars]
user=joe
```

3. Defining the **user** group variable from the **servers** group, which consists of two host groups each with two servers



```
[servers1]
demo1.example.com
demo2.example.com

[servers2]
demo3.example.com
demo4.example.com

[servers:children]
servers1
servers2

[servers:vars]
user=joe
```

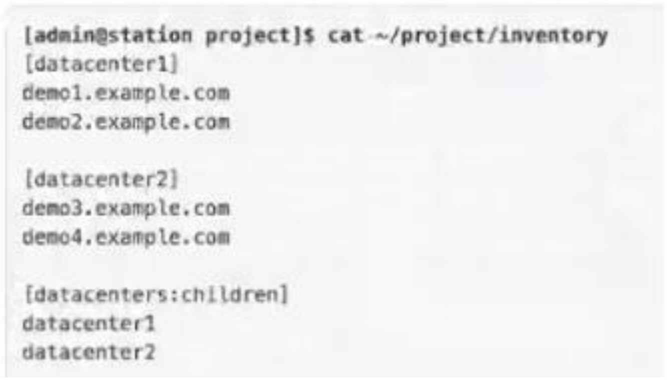
Some disadvantages of this approach are that it makes the inventory file more difficult to work with, it mixes information about hosts and variables in the same file and uses an obsolete syntax.

Using directories to populate Host and Group variables

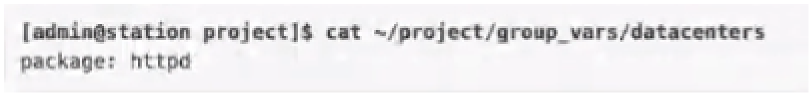
The preferred approach to defining variables for hosts and host groups is to create two directories, **group_vars** and **host_vars,** in the same working directory as the inventory file or directory. These directories contain files defining group variables and host variables, respectively. Important, the recommended practice is to define inventory variables using host_vars and group_vars directories and not to define them directly in the inventory files.

To define group variables for the **servers** group, you would create a YAML file named group_vars/servers and then the contents of the file would set variables to values using the same syntax in the playbook
user:joe
Likewise, 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.

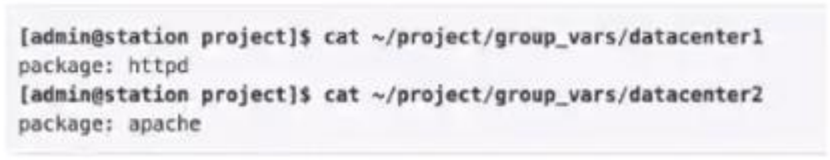
The following example illustrates this approach in more detail. Consider a scenario where there are two data centers to manage and the data center hosts are defined in
the **~/project/inventory** inventory file


```
[admin@station project]$ cat ~/project/inventory
[datacenter1]
demo1.example.com
demo2.example.com

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

[datacenters:children]
datacenter1
datacenter2
```

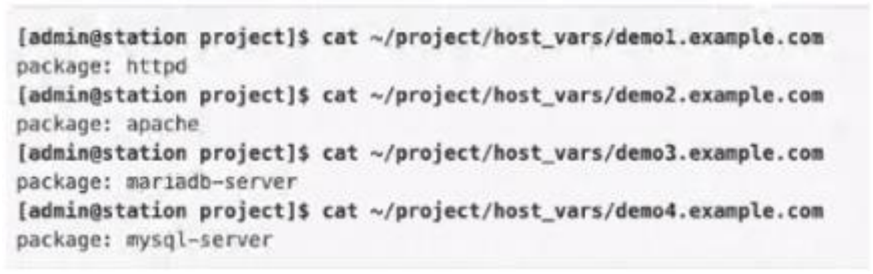
Now, if you need to define a general value for all servers in both data centers, set a group variable for the **datacenters** host group:


```
[admin@station project]$ 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:


```
[admin@station project]$ cat ~/project/group_vars/datacenter1
package: httpd
[admin@station project]$ cat ~/project/group_vars/datacenter2
package: apache
```

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

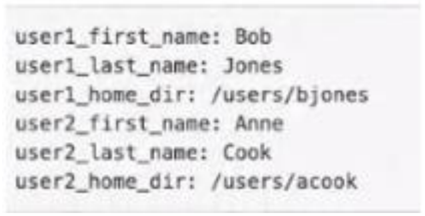

```
[admin@station project]$ cat ~/project/host_vars/demo1.example.com
package: httpd
[admin@station project]$ cat ~/project/host_vars/demo2.example.com
package: apache
[admin@station project]$ cat ~/project/host_vars/demo3.example.com
package: mariadb-server
[admin@station project]$ cat ~/project/host_vars/demo4.example.com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

Overriding variables from the Command Line

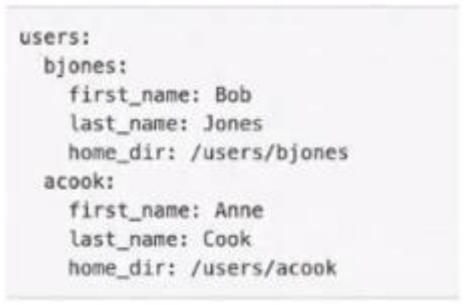
Inventory variables can be overridden by variables set in a playbook, but both kinds of variables may be overridden through arguments passed to the **ansible** or **ansible-playbook** command on the command line. Variables set on the command line are called **extra variables.**
ansible-playbook main.yml -e “package=mysql”

Using Arrays as Variables

Instead of assigning configuration data the relates to the same element (a list of packages, a list of services, a list of users), to multiple variables, administrators can use arrays.
For example, consider the following snippet –

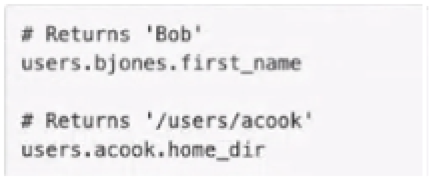

```
user1_first_name: Bob
user1_last_name: Jones
user1_home_dir: /users/bjones
user2_first_name: Anne
user2_last_name: Cook
user2_home_dir: /users/acook
```

This could be rewritten as an array called **users**



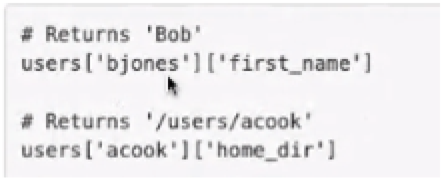
```
users:
  bjones:
    first_name: Bob
    last_name: Jones
    home_dir: /users/bjones
  acook:
    first_name: Anne
    last_name: Cook
    home_dir: /users/acook
```

You can then use the following variables to access user data



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

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

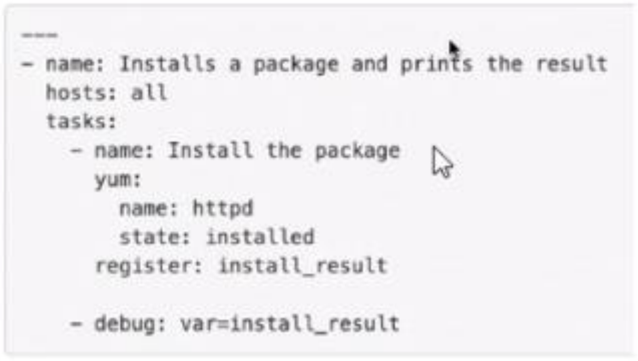
Because the variable is defined as a Python *dictionary,* an alternative syntax is available



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

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

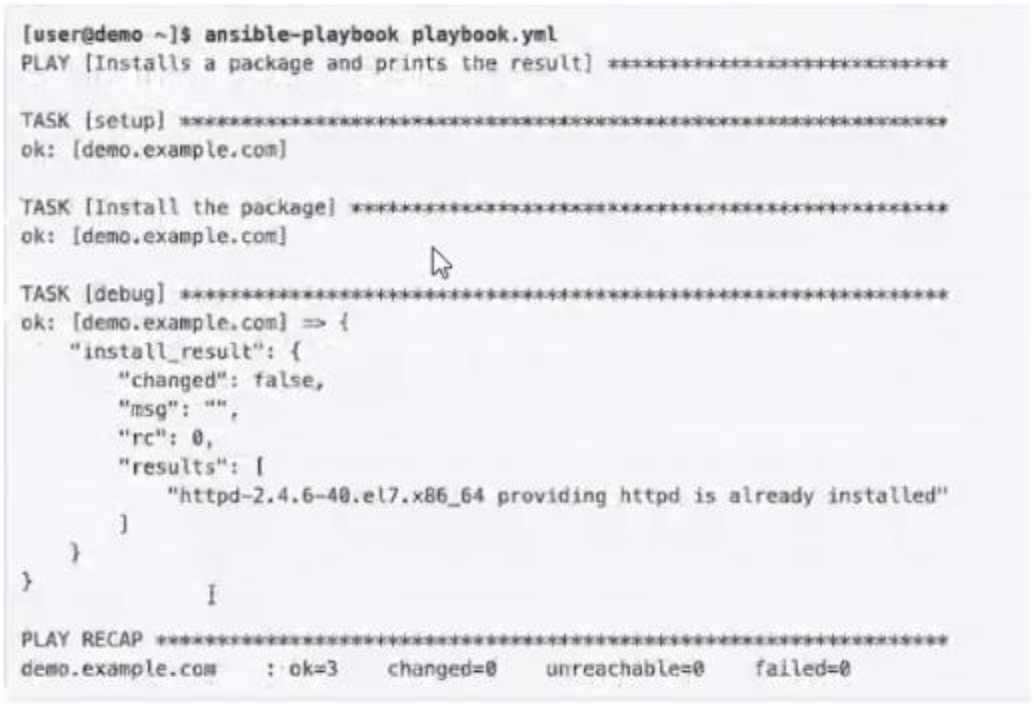
Capturing Command Output with Registered Variables

Administrators 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 or to achieve something else, such as a particular configuration based on a command's output.



```
---
- 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 to **install_result** registered variable to the terminal.



```
[user@demo ~]$ 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
```