Ansible is not a full-fledged programming language, but it does have several programming language features, and one of the most important of these is variable substitution.

There are different types of variables available in Ansible (you can click on individual link which will take you to the respective official documentation page from docs.ansible.org:

* Special variables
* Inventory variables
* Host variables
* Group variables
* Facts and local facts
* Connection variables
* and so on..

**Creating valid variable names**

* Before you start using variables, it’s important to know what are valid variable names.
* The name of the variable must only include letters, underscores, and numbers—spaces are not allowed.
* The name of the variable can only begin with a letter—they can contain numbers, but cannot start with one.

For example, the following are good variable names:

http\_port

server\_hostname\_db1

The following examples are all invalid, however, and cannot be used:

dbserver-east-zone

app server ip

web.server.port

01dbserver

**Built-in variables**

Ansible defines several variables that are always available in a playbook

| **Parameter** | **Description** |
| --- | --- |
| hostvars | A dictionary whose keys are Ansible host names and values are dictionaries that map variable names to values |
| inventory\_hostname | Name of the current host as known by Ansible |
| group\_names | A list of all groups that the current host is a member of |
| groups | A dictionary whose keys are Ansible group names and values are a list of hostnames that are members of the group. Includes all and ungrouped groups: {"all": […], "web": […], "ungrouped": […]} |
| play\_hosts | A list of inventory hostnames that are active in the current play |
| ansible\_version | A dictionary with Ansible version info: {'string': '2.9.13', 'full': '2.9.13', 'major': 2, 'minor': 9, 'revision': 13} |

**Defining variables in inventory**

There are two types of variables which you can define in an inventory i.e. host variables and group variables. Let us understand about each of them individually:

**Host variables**

We will use our default inventory from /etc/ansible/hosts which has below entries

server1

server2

server3

In this inventory I will define 2 variables for server2 in the following format

Advertisement

server1

server2 http\_port=8080 pkg=httpd

server3

So here I have defined two custom variables which are applicable only for server2. Now we will use this variable with ansible playbook. Here I have written a small playbook host\_vars.yml which will use those variables from the inventory so we can know where the variables are working or not:

---

- name: Verifying host variable

hosts: server2

tasks:

- name: Testing first variable (http\_port)

debug:

msg: "The HTTP PORT is {{ http\_port }}"

- name: Testing second variable (pkg)

debug:

msg: "The package name is {{ pkg }}"

Let us execute this playbook:

[ansible@controller ~]$ ansible-playbook host\_vars.yml

PLAY [Verifying host variable] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [server2]

TASK [Testing first variable (http\_port)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [server2] => {

"msg": "The HTTP PORT is 8080"

}

TASK [Testing second variable (pkg)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [server2] => {

"msg": "The package name is httpd"

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

server2 : ok=3 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

In the output from TASK you can see that the variable http\_port is replaced with 8080 and pkg is replaced with httpd.

**Group Variables**

I have already explained you about groups in an inventory. So group variables can be assigned to a complete group and all the hosts part of that group instead of individual hosts. To demonstrate the let me modify my inventory file and divide the list of hosts into group

[app]

server1

server3

[db]

server2

[db:vars]

http\_port=8080

pkg=httpd

Here I have divided my managed nodes into two groups and have assigned the variables to db group using db:vars. I will make some changes to our playbook file i.e. replace server2 with the group name db and some minor text changes:

---

- name: Verifying group variable

hosts: db

tasks:

- name: Testing first variable (http\_port)

debug:

msg: "The HTTP PORT is {{ http\_port }}"

- name: Testing second variable (pkg)

debug:

msg: "The package name is {{ pkg }}"

Let us execute the group\_vars.yml playbook and verify the group\_vars:

[ansible@controller ~]$ ansible-playbook group\_vars.yml

PLAY [Verifying group variable] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [server2]

TASK [Testing first variable (http\_port)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [server2] => {

"msg": "The HTTP PORT is 8080"

}

TASK [Testing second variable (pkg)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [server2] => {

"msg": "The package name is httpd"

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

server2 : ok=3 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

**Defining variable in project**

If you are using ansible playbook under some project then it is not recommended to modify the inventory file. We have a cleaner solution available to define host and group variables in Project.

To define host variable we can create a sub-directory host\_vars and similarly to define group variable we can create a sub-directory group\_vars inside the main project directory.

Advertisement

For example I will create a project "lab1" and copy the ansible.cfg from default directory /etc/ansible/

[ansible@controller ~]$ mkdir lab1

[ansible@controller ~]$ cd lab1/

Copy /etc/ansible/ansible.cfg to the project directory

[ansible@controller lab1]$ cp /etc/ansible/ansible.cfg .

Create an inventory file, I will just add server2 in my inventory file as that is enough to demonstrate the example here:

[ansible@controller lab1]$ cat inventory

server2

Now we will create host\_vars directory inside lab1

[ansible@controller lab1]$ mkdir host\_vars

Inside host\_vars we will create a new file with the same name as of the server i.e. server2 in my case and define the variables:

[ansible@controller lab1]$ cat host\_vars/server2

http\_port: 8080

pkg: httpd

Now we copy our existing host\_vars.yml to this project's home folder:

Advertisement

[ansible@controller lab1]$ cp ~/host\_vars.yml .

and then execute the playbook:

[ansible@controller lab1]$ ansible-playbook host\_vars.yml

PLAY [Verifying host variable] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [server2]

TASK [Testing first variable (http\_port)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [server2] => {

"msg": "The HTTP PORT is 8080"

}

TASK [Testing second variable (pkg)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [server2] => {

"msg": "The package name is httpd"

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

server2 : ok=3 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

So ansible was able to fetch the variable from server2 file inside host\_vars directory.

**Similarly** we can create group\_vars directory under lab1 and use the name of the group to create a new file which will contain the list of variables. Next modify the hosts: entry with the group name instead of server name in the playbook file.

**Defining variables in playbook**

The simplest way to define variables is to put a vars section in your playbook with the names and values of variables.

In this example playbook-vars.yml I have defined two custom variables under vars and I am using them with debug module. Additionally I am also using to built-in variables

---

- hosts: server2

vars:

port\_no: 80

pkg\_name: httpd

gather\_facts: no

tasks:

- debug:

msg:

- "The value of port is {{ port\_no }}"

- "The value of pkg is {{ pkg\_name }}"

- "The value of hostname is {{ inventory\_hostname }}"

- "Ansible version is {{ ansible\_version }}"

Let us execute this playbook:

[ansible@controller ~]$ ansible-playbook playbook-vars.yml

PLAY [server2] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [server2] => {

"msg": [

"The value of port is 80",

"The value of pkg is httpd",

"The value of hostname is server2",

"Ansible version is {'string': '2.9.13', 'full': '2.9.13', 'major': 2, 'minor': 9, 'revision': 13}"

]

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

server2 : ok=1 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

So the ansible has successfully fetched the values of the variables.

**Defining variables using command line**

Variables set by passing -e or --extra-vars with var=value to ansible-playbook have the highest precedence, which means you can use this to override variables that are already defined

Here I have a sample playbook variable-with-cmd-line.yml where I have defined a variable username with a value as deepak. I am then using debug module to print the value of username variable

---

- hosts: localhost

vars:

username: deepak

tasks:

- debug:

msg: "USERNAME={{ username }}"

Now additionally since this section is about using **command line variables** we will define the same variable using username and assign a different value:

[ansible@controller ~]$ ansible-playbook variable-with-cmd-line.yml -e username=golinuxcloud

PLAY [localhost] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost]

TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost] => {

"msg": "USERNAME=golinuxcloud"

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

As you see from the TASK section output, the command line variable value which we gave with -e has taken precedence compared to the value given by vars inside the playbook.

**Accessing Variables**

There are different ways using which you can access the content of the variable. We have used double curly braces to access all our variables in the examples above. This curly braces syntax is part of Jinja2 template.

Now inside these curly braces we have simply defined the name of the variable and the mapping value was printed on the console. But sometimes the variables can be in complex structure such as the output of "Gathering Facts"

We can access the facts of a system using setup module which gives us a long list of output. Some provided facts, like networking information, are made available as nested data structures. To access them a simple {{ foo }} is not sufficient, but it is still easy to do. Here’s how we get an IP address:

Here in this playbook access-variables.yml I have combined multiple methods using which you can capture the IPv4 address of eth0 using the Ansible Facts

---

- name: Collect IPv4 address of eth0

hosts: localhost

tasks:

- debug:

msg:

- "Method-1 {{ ansible\_eth0.ipv4.address }}"

- "Method-2 {{ ansible\_eth0['ipv4']['address'] }}"

- "Method-3 {{ ansible\_eth0['ipv4'].address }}"

- "Method-4 {{ ansible\_eth0.ipv4['address'] }}"

As you can see under debug module section there are couple of methods using which we can collect the IPv4 address from the facts. Let us execute this playbook:

[ansible@controller ~]$ ansible-playbook access-variables.yml

PLAY [Capture output of command] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost]

TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost] => {

"msg": [

"Method-1 172.31.7.253",

"Method-2 172.31.7.253",

"Method-3 172.31.7.253",

"Method-4 172.31.7.253",

]

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

So we get the same value using all the methods.

**Accessing dictionary keys in a variable**

When we execute the setup module it gives us a long bunch of output with system information. This information is by default stored into ansible\_facts variable. Individually you can also search for other variables with stores different values of your system, to get this list of variables from setup module you can execute:

[ansible@controller ~]$ ansible localhost -m setup | grep ansible

"ansible\_facts": {

"ansible\_all\_ipv4\_addresses": [

"ansible\_all\_ipv6\_addresses": [

"ansible\_apparmor": {

"ansible\_architecture": "x86\_64",

"ansible\_bios\_date": "08/24/2006",

"ansible\_bios\_version": "4.2.amazon",

"ansible\_cmdline": {

"ansible\_date\_time": {

"ansible\_default\_ipv4": {

"ansible\_default\_ipv6": {},

"ansible\_device\_links": {

"ansible\_devices": {

"ansible\_distribution": "CentOS",

"ansible\_distribution\_file\_parsed": true,

"ansible\_distribution\_file\_path": "/etc/redhat-release",

"ansible\_distribution\_file\_variety": "RedHat",

"ansible\_distribution\_major\_version": "8",

"ansible\_distribution\_release": "Core",

"ansible\_distribution\_version": "8.2",

...

"ansible\_user\_shell": "/bin/bash",

"ansible\_user\_uid": 1001,

"ansible\_userspace\_architecture": "x86\_64",

"ansible\_userspace\_bits": "64",

"ansible\_virtualization\_role": "guest",

"ansible\_virtualization\_type": "xen",

So you can use any of these variables to collect system information.

If a variable contains a dictionary, you can access the keys of the dictionary by using either a dot (.) or a subscript ([]). This is the trimmed sample output of setup module which contains the interface and IP address information which we have used to get the data in previous example:

Advertisement

"ansible\_eth0": {

"active": true,

"device": "eth0",

"features": {

"esp\_hw\_offload": "off [fixed]",

...

"tls\_hw\_rx\_offload": "off [fixed]",

"tls\_hw\_tx\_offload": "off [fixed]",

},

"hw\_timestamp\_filters": [],

"ipv4": {

"address": "172.31.7.253",

"broadcast": "172.31.15.255",

"netmask": "255.255.240.0",

"network": "172.31.0.0"

},

Here the output is in dictionary format and we need to access the Keys of this dictionary. So we first take the variable name ansible\_eth0 followed by the key so using **dot** notation the variable becomes ansible\_eth0.ipv4.addess

Now instead of dot you can also use subscript[] and modify the variable to ansible\_eth0['ipv4']['address']

Finally to access the variable inside the ansible playbook we use Jinja2 template using double curly braces {{}}

**Using register module to store output of any command**

Registered variables are similar to facts, with a few key differences. Like facts, registered variables are host-level variables. However, registered variables are only stored in memory. (Ansible facts are backed by whatever cache plugin you have configured.) Registered variables are only valid on the host for the rest of the current playbook run. Finally, registered variables and facts have different precedence levels.

I have a sample playbook file register\_variable.yml with following content:

---

- name: Capture output of command

hosts: localhost

gather\_facts: false

tasks:

- name: Collect ip address

shell: "ifconfig eth0"

This playbook will execute "ifconfig eth0" on localhost.

[ansible@controller ~]$ ansible-playbook register\_variable.yml

PLAY [Capture output of command] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Collect ip address] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [localhost]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

localhost : ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

But here our intention is to **get the IP Address** used by eth1 interface so we must capture the output of "ifconfig eth1"

To capture the output we use register module. The value of a variable set using the register clause is always a dictionary, but the specific keys of the dictionary are different, depending on the module that was invoked.

So we will update our playbook to use register module and store the output of "ifconfig eth0" into ip\_addr variable

---

- name: Capture output of command

hosts: localhost

gather\_facts: false

tasks:

- name: Collect ip address

shell: "ifconfig eth0"

register: ip\_addr

- debug: var=ip\_addr

Next we will execute our playbook which now contains a long list of output:

[ansible@controller ~]$ ansible-playbook register\_variable.yml

PLAY [Capture output of command] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Collect ip address] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [localhost]

TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost] => {

"ip\_addr": {

"changed": true,

"cmd": "ifconfig eth0",

"delta": "0:00:00.004912",

"end": "2020-09-23 08:16:01.349273",

"failed": false,

"rc": 0,

"start": "2020-09-23 08:16:01.344361",

"stderr": "",

"stderr\_lines": [],

"stdout": "eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 9001\n inet 172.31.7.253 netmask 255.255.240.0 broadcast 172.31.15.255\n inet6 fe80::4c:ff:feb4:e3c prefixlen 64 scopeid 0x20\n ether 02:4c:00:b4:0e:3c txqueuelen 1000 (Ethernet)\n RX packets 12128 bytes 967043 (944.3 KiB)\n RX errors 0 dropped 0 overruns 0 frame 0\n TX packets 9008 bytes 1183882 (1.1 MiB)\n TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0",

"stdout\_lines": [

"eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 9001",

" inet 172.31.7.253 netmask 255.255.240.0 broadcast 172.31.15.255",

" inet6 fe80::4c:ff:feb4:e3c prefixlen 64 scopeid 0x20",

" ether 02:4c:00:b4:0e:3c txqueuelen 1000 (Ethernet)",

" RX packets 12128 bytes 967043 (944.3 KiB)",

" RX errors 0 dropped 0 overruns 0 frame 0",

" TX packets 9008 bytes 1183882 (1.1 MiB)",

" TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0"

]

}

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

localhost : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

In this output we are only interested in stdout key value so to access stdout without variable we will use ip\_addr.stdout (which we had learned in our last chapter "Accessing dictionary keys in a variable"

We will update our playbook with the following entry:

- debug: var=ip\_addr.stdout

and re-run the playbook:

Advertisement

TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost] => {

"ip\_addr.stdout": "eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 9001\n inet 172.31.7.253 netmask 255.255.240.0 broadcast 172.31.15.255\n inet6 fe80::4c:ff:feb4:e3c prefixlen 64 scopeid 0x20\n ether 02:4c:00:b4:0e:3c txqueuelen 1000 (Ethernet)\n RX packets 12297 bytes 978553 (955.6 KiB)\n RX errors 0 dropped 0 overruns 0 frame 0\n TX packets 9111 bytes 1199660 (1.1 MiB)\n TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0"

}

So now we get the content of stdout only. But this needs formatting and we will perform some additional strip to get the value we want i.e. the IP address of eth0. I will split the content with "\n" (new line) and get the first index value i.e. the second line

- debug: var=ip\_addr.stdout.split("\n")[1]

Let us re-run the playbook:

TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost] => {

"ip\_addr.stdout.split(\"\n\")[1]": " inet 172.31.7.253 netmask 255.255.240.0 broadcast 172.31.15.255"

}

Now we have much cleaner output but we still need some more formatting to get the IP address of eth0. So we will add another split based on whitespace character and then print the first index value i.e. the second line:

- debug: var=ip\_addr.stdout.split("\n")[1].split()[1]

Let us check the output:

TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost] => {

"ip\_addr.stdout.split(\"\n\")[1].split()[1]": "172.31.7.253"

}

Now we get the IP Address of eth0.

**Using set\_fact module to create a new variable**

**How can we assign this IP address we are getting from all these split operation to a new variable**?

vars can be used only when we are statically defining a variable inside the playbook but for dynamic values which are collected runtime we can use set\_fact

Ansible also allows you to set a fact (effectively the same as defining a new variable) in a task by using the set\_fact module. So we will use set\_fact module and define a new variable ip\_addr inside our playbook which will perform all the split operations and then store the final value inside ip\_addr

I have updated my playbook to use set\_fact module:

---

- name: Capture output of command

hosts: localhost

gather\_facts: false

tasks:

- name: Collect ip address

shell: "ifconfig eth0"

register: ip\_addr

- set\_fact:

ip\_addr: "{{ ip\_addr.stdout.split('\n')[1].split()[1] }}"

- debug: var=ip\_addr

Here we are performing the same split operation but instead of performing with debug module we do this wit set\_fact so that we can store the output into ip\_addr variable. Now we can use this variable across playbook when ever required

Let us execute this playbook:

[ansible@controller ~]$ ansible-playbook register\_variable.yml

PLAY [Capture output of command] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Collect ip address] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [localhost]

TASK [set\_fact] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost]

TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost] => {

"ip\_addr": "172.31.7.253"

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

localhost : ok=3 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

So we are properly getting the IP Address value of eth0 interface.

**Prompt for user input with vars\_prompt**

When running a playbook, there may be information that you need to collect at runtime. This may be sensitive information, such as passwords. At other times, this is information that can only be provided by the end user at runtime, such as the password to use for the root user when bootstrapping a system.

You can collect this information from the end user by specifying a vars\_prompt section in your playbook. When you run the playbook, it will ask the questions you’ve specified and record the answers, ready to be used as variables.

We will write another playbook prompt-variable.yml which will prompt the user for the USERNAME and PASSWORD using vars\_prompt module and then print these values on the console using debug module. The prompt for user\_name is set to private: no so the operator can see the text while for password we have set private: yes so the operator will not see anything while typing the password.

---

- name: Prompt for user input

hosts: localhost

gather\_facts: no

vars\_prompt:

- name: user\_name

prompt: Enter your user name

private: no

- name: password

prompt: Enter your password

private: yes

tasks:

- debug:

msg: "The username is {{ user\_name}} and password is {{ password }}"

Let us execute this playbook and observe the output:

[ansible@controller ~]$ ansible-playbook prompt-variable.yml

Enter your user name: Deepak

Enter your password:

PLAY [Prompt for user input] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost] => {

"msg": "The username is Deepak and password is abcd"

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

localhost : ok=1 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

**Read and access variables from separate YAML or JSON file**

In all the examples above we have defined variable inside inventory, playbook or host\_vars/group\_vars folder. Now in this section we will create a separate YAML or JSON file which will only contain a bunch of variables and using playbook we will refer this YAML or JSON file to read and access the variable value.

This can help you reduce the length of the playbook when working with multiple variables:

**Using YAML File**

We will create a separate YAML file variables.yml with following content:

[ansible@controller ~]$ cat variables.yml

NAME: Deepak

COURSE: Ansible

CHAPTER: Variables

Here I have defined 3 variables and we will access these variables using our playbook read-from-yml-file.yml

---

- name: Collects variable from variables.yml file

hosts: localhost

vars\_files: variables.yml

gather\_facts: false

tasks:

- debug:

msg:

- "My name is {{ NAME }}"

- "This is {{ COURSE }} tutorial"

- "We are learning about {{ CHAPTER }}"

Here we have defined our variables.yml file with vars\_files module. Since this file exists in my current working directory I have not given any path or you must give the full absolute path of the variable file.

Output from this playbook:

[ansible@controller ~]$ ansible-playbook read-from-yml-file.yml

PLAY [Collects variable from variables.yml file] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost] => {

"msg": [

"My name is Deepak",

"This is Ansible tutorial",

"We are learning about Variables"

]

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

localhost : ok=1 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

So the playbook has successfully collected the values from our YAML file.

**Using JSON File**

Similarly we can also create a variable file custom\_vars.json using JSON format

[ansible@controller ~]$ cat custom\_vars.json

{

"a": 50,

"b": [1,2,3,4,5],

"c": { "one":1, "two":2 }

}

Next we will update our playbook file to refer both the variables file. We can place the path and variable file name using separate line and mention multiple files.

---

- name: Collects variable from variables.yml file

hosts: localhost

vars\_files:

- ~/variables.yml

- ~/custom\_vars.json

gather\_facts: false

tasks:

- debug:

msg:

- "My name is {{ NAME }}"

- "This is {{ COURSE }} tutorial"

- "We are learning about {{ CHAPTER }}"

- "The value of b is: {{ b }}"

- "Second index value of b is: {{ b[1] }}"

- "The value of c is {{ c }}"

Let us execute this script:

[ansible@controller ~]$ ansible-playbook read-from-yml-file.yml

PLAY [Collects variable from variables.yml file] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [localhost] => {

"msg": [

"My name is Deepak",

"This is Ansible tutorial",

"We are learning about Variables",

"The value of b is: [1, 2, 3, 4, 5]",

"Second index value of b is: 2",

"The value of c is {'one': 1, 'two': 2}"

]

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

localhost : ok=1 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

So our playbook is able to read from both the variables file.

**Understanding variable precedence**

We’ve covered several ways of defining variables, and it can happen that you define the same variable multiple times for a host, using different values. Avoid this when you can, but if you can’t, then keep in mind Ansible’s precedence rules. When the same variable is defined in multiple ways, the precedence rules determine which value wins.  
For updated list of precedence rules,

The basic rules of precedence are as follows:

* **(Lowest)**command line values (for example, -u my\_user, these are not variables)
* role defaults (defined in role/defaults/main.yml) 1
* inventory file or script group vars 2
* inventory group\_vars/all
* playbook group\_vars/all
* inventory group\_vars/\*
* playbook group\_vars/\*
* inventory file or script host vars
* inventory host\_vars/\*
* playbook host\_vars/\*
* host facts / cached set\_facts
* play vars
* play vars\_prompt
* play vars\_files
* role vars (defined in role/vars/main.yml)
* block vars (only for tasks in block)
* task vars (only for the task)
* include\_vars
* set\_facts / registered vars
* role (and include\_role) params
* include params
* **(Highest)**extra vars (for example, -e "user=my\_user")