**ANSIBLE**

# Chapter 1 & 2 : Introduction & Ad-Hoc command

Two steps you would have to do after install ansible:

* Creating ansible user with sudo privileges
* Generate ssh key for control node then copy that ssh key to all managed hosts
* Add ansible user to sudoer file

**Syntax of sudoer files:**

Users host=(user:group) commands

You can also use alias for user, host… such as:

User\_Alias REBOOT\_USERS = Tim, John, Swith

Cmnd\_Alias REBOOT\_COMMANDS = /sbin/halt, /sbin/init, /sbin/poweroff

Host\_Alias REBOOT\_HOSTS = 192.168.10.10

Runas\_Alias REBOOT\_RUNAS = admins

The alias can then be used

REBOOT\_USERS REBOOT\_HOSTS=(REBOOT\_RUNAS)REBOOT\_COMMANDS

Note: Percentile symbol ‘**%**’ is used to represent a group in Sudoers File

Reference: <https://medium0.com/kernel-space/linux-fundamentals-a-to-z-of-a-sudoers-file-a5da99a30e7f>

**Ansible inventory**

Ansile inventory file is a basically a file that contains list of server, ip, user of servers that you want to managed by control node

Default location of inventory file /etc/ansible/hosts

There are two default groups in Ansible:

1. all - contains all the hosts in the inventory
2. ungrouped - contains all the hosts that are not a member of any group (aside from all).

**Creating a project directory**

**Creating host group, sub group, parent group**

[elliot@control plays]$ cat myhosts

node5

[test]

node1

node2

[prod]

node3

node4

[elliot@control plays]$ ansible ungrouped -i myhosts --list-hosts

hosts (1):

node5

[elliot@control plays]$ cat myhosts

[web\_dev]

node1

[web\_prod]

node2

[db\_dev]

node3

[db\_prod]

node4

[development:children]

web\_dev

db\_dev

[production:children]

web\_prod

db\_prod

[elliot@control plays]$ ansible development -i myhosts --list-hosts

hosts (2):

node1

node3

[elliot@control plays]$ ansible production -i myhosts --list-hosts

hosts (2):

node2

node4

**Configuring Ansible**

By default ansible have 2 import section: [defaults] and [privilege\_escalation]

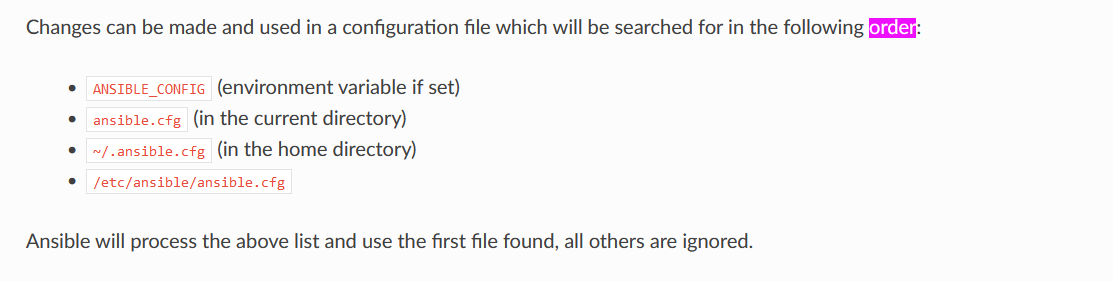
In the **[defaults]** section, here are the most important settings you need to be aware of:

* **inventory -** specifies the path of your inventory file.
* **remote\_user -** specifies the user who will connect to the managed hosts and run the playbooks.
* **forks -** specifies the number of host that Ansible can manage/process in parallel; default is 5.
* **host\_key\_checking -** specifies whether you want to turn on/off SSH key host checking; default is True.

In the **[privilege\_escalation]**section, you can configure the following settings:

* **become -** specify where to allow/disallow privilege escalation; default is False.
* **become\_method -** specify the privilege escalation method; default is sudo.
* **become\_user -** specify the user you become through privilege escalation; default is root.
* **become\_ask\_pass -** specify whether to ask or not ask for privilege escalation password; default is False.

Ansible Configuration file order



**Checking default module in ansible**

[elliot@control plays]$ ansible-doc -l | wc -l

**Workaround of being prompted passphrase**

[elliot@control plays]$ eval `ssh-agent`

Agent pid 218750

[elliot@control plays]$ ssh-add

Enter passphrase for /home/elliot/.ssh/id\_rsa:

Identity added: /home/elliot/.ssh/id\_rsa (elliot@control)

**Ansible Modules Documentation**

[elliot@control plays]$ ansible-doc ping

**Command vs. Shell vs. Raw Modules**

There are three Ansible modules that people often confuse with one another; these are:

1. command
2. shell
3. raw

Those three modules achieve the same purpose; they run commands on the managed nodes. But there are key differences that separates the three modules.

You can’t use piping or redirection with the **command** module. For example, the following ad-hoc command will result in an error:

[elliot@control plays]$ ansible node2 -m command -a "lscpu | head -n 5"

node2 | FAILED | rc=1 >>

lscpu: invalid option -- 'n'

Try 'lscpu --help' for more information.non-zero return code

[elliot@control plays]$ ansible node2 -m shell -a "lscpu | head -n 5"

node2 | CHANGED | rc=0 >>

Architecture: x86\_64

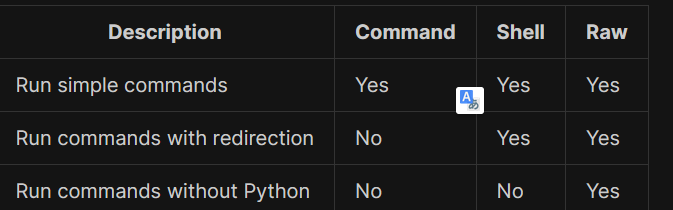
CPU op-mode(s): 32-bit, 64-bit

Byte Order: Little Endian

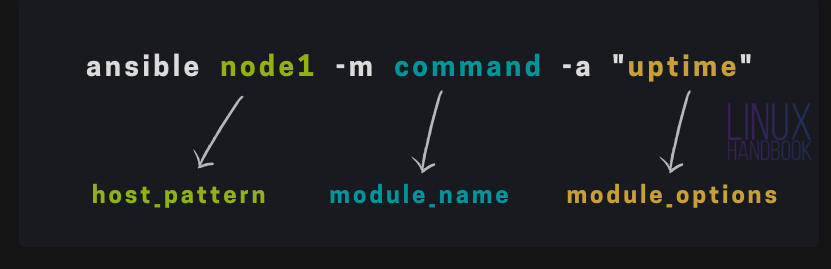
CPU(s): 1

On-line CPU(s) list: 0

Ansible uses SSH and Python scripts behind the scenes to do all the magic. Now, the **raw** module just uses SSH and bypasses the Ansible module subsystem. This way, that raw module would successfully work on the managed node even if python is not installed (on the managed node).



Run ad-hoc command in Ansible



# Chapter 3: Ansible Playbook

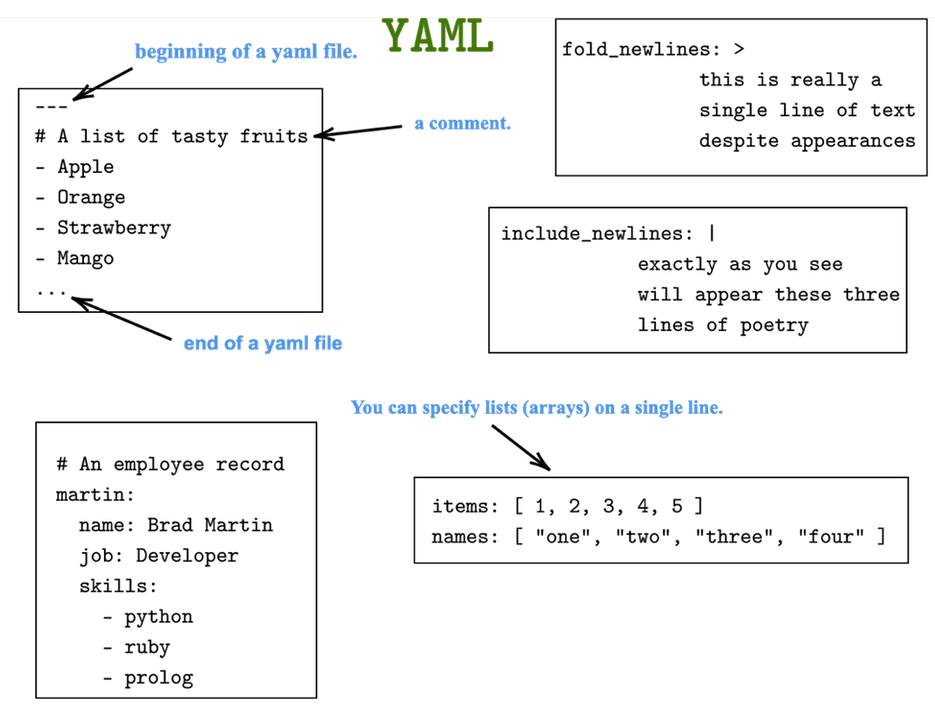
You can think ansible ad-hoc command is linux command and ansible playbook is bash script

On the other hand, Ansible playbooks are ideal to automate complex tasks like system patches, application deployments, firewall configurations, user management, etc.

Resource:

<https://github.com/kabary/rhce8?ref=linuxhandbook.com>

## Creating your first Ansible playbook



Playbooks are written in [YAML](https://yaml.org/?ref=linuxhandbook.com) (Yet Another Markup Language) format.

Also, YAML is indentation sensitive. A two-spaces indentation is the recommended indentation to use in YAML; however, YAML will follow whatever indentation system a file uses as long as it’s consistent.

Do a yourself favor and include the following line in ./vimrc file:

autocmd FileType yaml setlocal ai ts=2 sw=2 et

for example first playbook:

[elliot@control plays]$ cat first-playbook.yml

---

- name: first play

hosts: all

tasks:

- name: create a new file

file:

path: /tmp/foo.conf

mode: 0664

owner: elliot

state: touch

To run ansible playbook using the syntax below:

#ansible-playbook name\_of\_playbook.yml

Pay especial attention to “changed” and “Play recap” of result

Notice that you can also run an Ansible ad-hoc command that will do exactly the same thing as **first-playbook.yml** playbook:

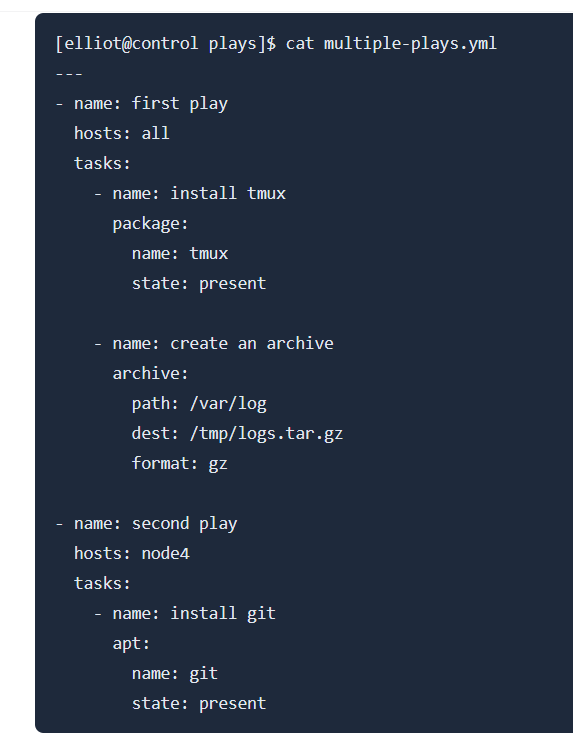
ansible all -m file -a "path=/tmp/foo.conf mode=0664 owner=elliot state=touch"

to helping with documentation just type:

#ansible-doc module\_name

## Running multiple plays with Ansible Playbook

A playbook can contains multiple plays and each play in turn contains many tasks



## Verifying your playbooks (before you run it)

#ansible-playbook –syntax-check playbook\_name.yml

You could also check playbook without change anything to managed nodes with option “ –check”

#ansible-playbook –check playbook\_name.yml

Or list all of node with each plays with “—list-hosts” options

#ansible-playbook –list-hosts playbook\_name.yml

Or list all tasks within each play with “ –list-tasks” options

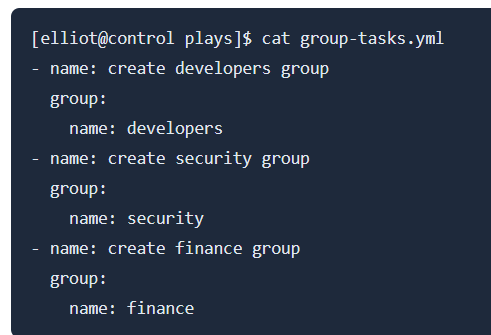
#ansible-playbook –list-tasks playbook\_name.yml

## Re-using tasks and playbooks

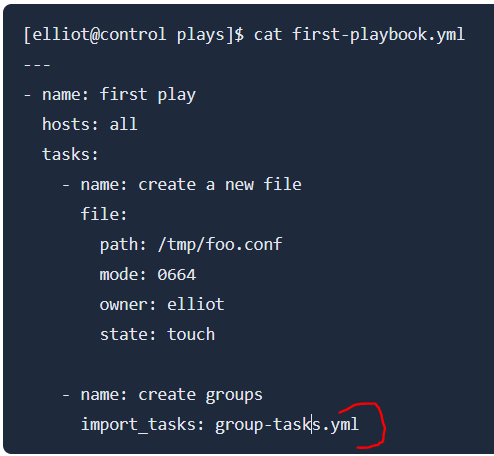
Re-use playbook with “import-playbook” module

Note: just only re-use on playbook within a on new play level, you can’t reuse a play within another play.

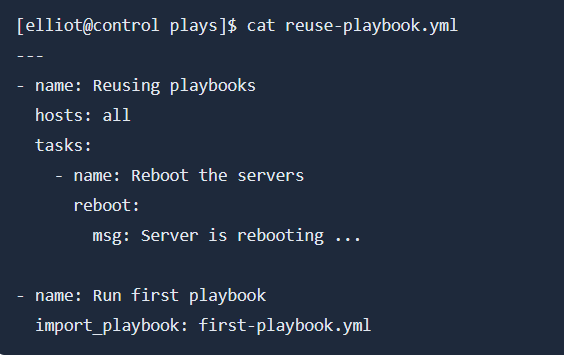
To demonstrate, let’s create a file named **group-tasks.yml** that contains the following tasks:



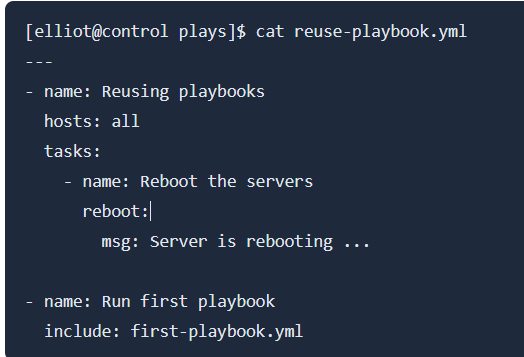
Now you can use the **import\_tasks** module to run all the tasks in **group-tasks.yml** in your first playbook as follows:



You can also the **import\_playbook** module to reuse an entire playbook.



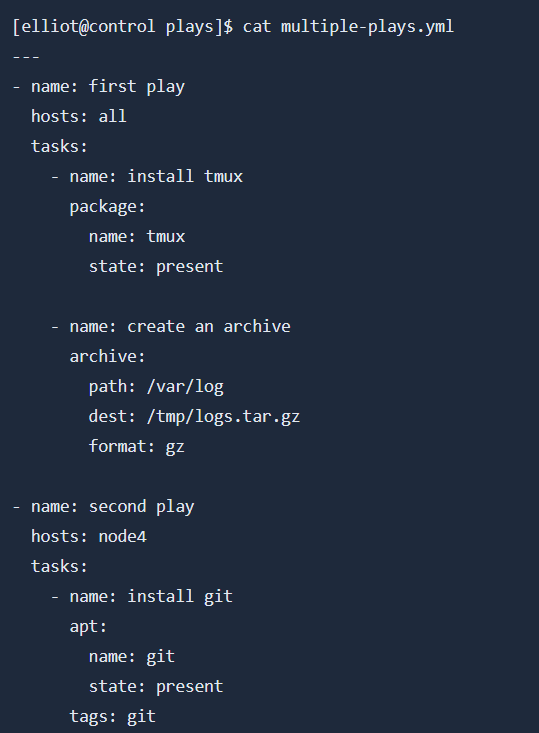
You can use “include” module instead of “import-playbook” and “import-tasks” module



The only difference is that the **import** statements are pre-processed at the time playbooks are parsed. On the other hand, **include** statements are processed as they are encountered during the execution of the playbook. So, in summary, **import** is static while **include** is dynamic.

## Running selective tasks and plays with Ansible playbook

You can choose not to run all playbook instead of run specific task(s) or play(s) in a playbook using “tags” options.



Now you can use –tags option follow by tag\_name to only run the specific task.

#Ansible-playbook playboo\_name.yml –tags git

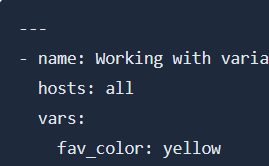
# Chapter #4: Ansible Variables, Facts and Registers

This chapter will show you how to use ansible variable and fact to get information on managed nodes.

Furthermore, you will also learn how to use registers to capture task output.

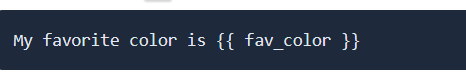
Define variable:

Use “var” keyword to define variable



Ansible uses the [Jinja2 templating system](https://palletsprojects.com/p/jinja/?ref=linuxhandbook.com) to work with variables.

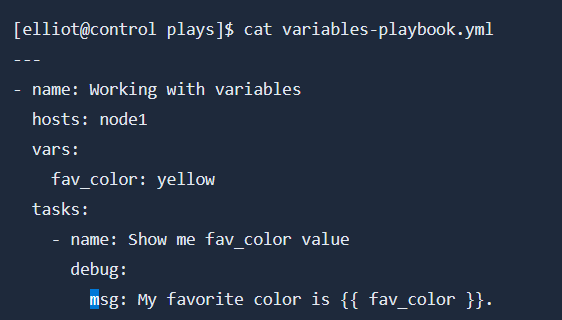
To get the value of the **fav\_color** variable; you need to surround it by a pair of curly brackets as follows:



Notice that if your variable is the first element (or only element) in the line, then using quotes is mandatory as follows:



Ex:



Note:

I have used the **debug** module along with the **msg** module option to print the value of the **fav\_color** variable.

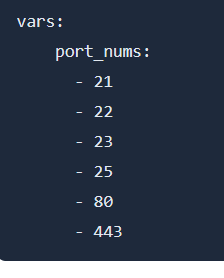
### Creating lists and dictionaries

you may define a list named **port\_nums** and set its value as follows:

vars:

port\_nums: [1,2,3,4,5]

above is list defined as one line, in case you would like to define as multiple line:



print all the values in **port\_nums** as follows:

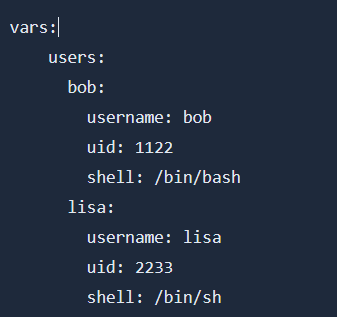


access a specific list element:



Note: Notice that you use an index (position) to access list elements.

Dictionary

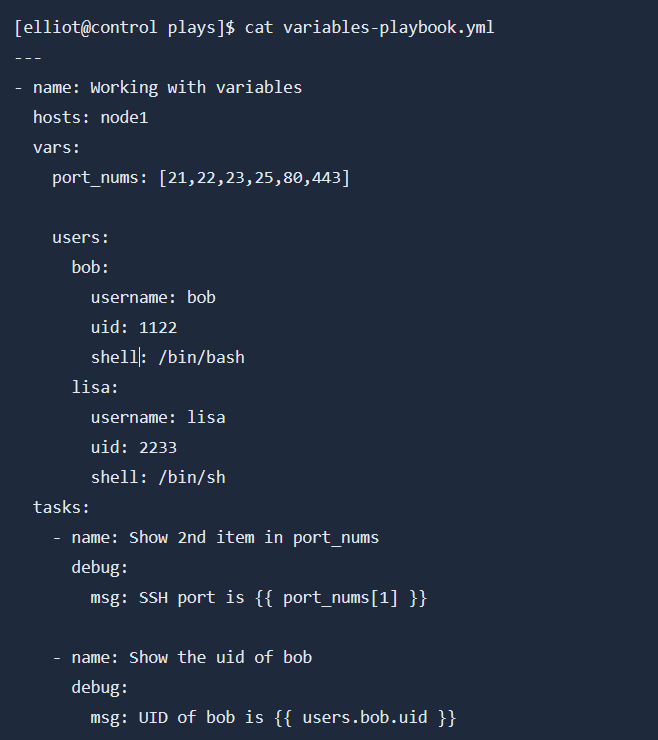


There are two different ways you can use to access dictionary elements:

* dict\_name['key'] -> **users['bob']['shell']**
* dict\_name.key -> **users.bob.shell**

note: Notice that you use a key to access dictionary elements.

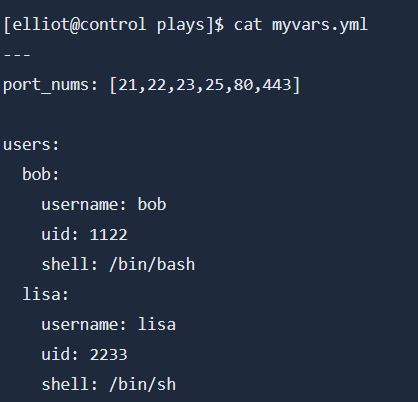
Ex:



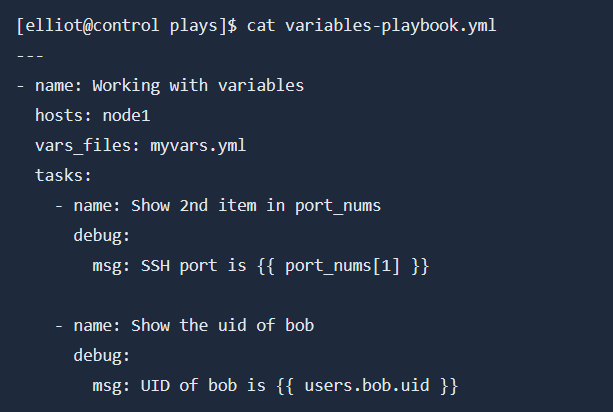
### Including external variables

Just like you can import (or include) tasks in a playbook. You can do the same thing with variables as well. That is, in a playbook, you can include variables defined in an external file.

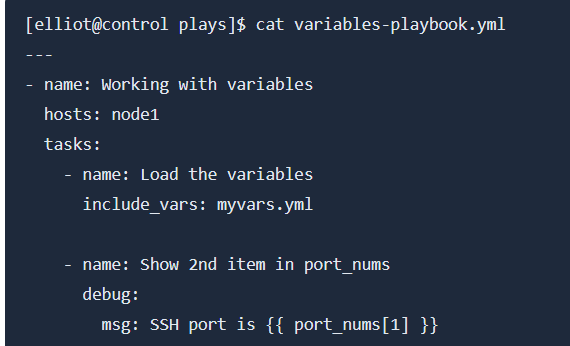
To demonstrate, let’s create a file named **myvars.yml** that contains our **port\_nums** list and **users** dictionary:



Use the **vars\_files** keyword in your **variables-playbook.yml** to include the variables in **myvars.yml** as follows:



Keep in mind that **vars\_files** preprocesses and load the variables right at the start of the playbook. You can also use the **include\_vars** module to dynamically load your variables in your playbook:

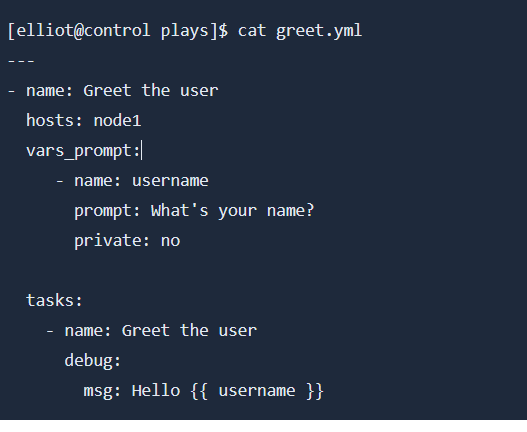


Note: include\_vars is a module and vars\_files is not a module

### Getting user input

You can use the **vars\_prompt** keyword to prompt the user to set a variable’s value at runtime.

For example, the following **greet.yml** playbook asks the running user to enter his name and then displays a personalized greeting message:



Notice I used **private: no** so that you can see your input on the screen as you type it; by default, it’s hidden.

### Setting host and group variables

# Reading

<https://linuxhandbook.com/ansible-variables-facts-registers/>