



ANSIBLE

# ANSIBLE BASIC LAB MANUAL

Student Lab Kit v1.1

## ABSTRACT

This lab manual is designed for students who are interested in Ansible Basic Automation

**Confidential Document**

Basic Playbooks

## Table of Contents

<b>Lab Overview and objectives</b>	<b>2</b>
<i>Guided Tasks</i>	2
Task 1: Create your first playbook	2
Task 2: Add another play to existing playbook	4
Commonly used modules	5
Task 2: User and group modules	5
Task 2.1: Create a new user account	5
Task 2.2: Set bash for "testuser"	6
Task 2.3: Create a new group	6
Task 2.4: Add user to groups	7
Task 3: File module	7
Task 3.1: Create .ssh folder for "testuser"	7
Task 3.2: Create authorized_keys file inside .ssh	8
Task 4: Copy module	8
Task 4.1: Copy ansible_key.pub to authorized_keys file	9
Task 4.2: Test authentication	9
Task 5: Lineinfile module	9
Task 5.1: Add "testgroup" to sudoers file	9
Raw vs Command vs Shell	10
Task 6: Command module	10
Task 7: Raw module	11
Task 7: Shell module	11
Task 8: Command line modules in Playbooks	12
Task results	13
Task 9: OK vs Changed vs Failed	13
Task 10: Setting password for a user	15

# Lab Overview and objectives

The purpose of this lab is learning how to write and use Ansible Playbooks, which are a completely different way to use Ansible than ad-hoc commands, being more powerful and useful in configuration management.

## Guided Tasks

### Task 1: Create your first playbook

Let's start by creating a simple playbook which contains just one play (and particularly this play will contain just one task) and performs the ping command to all hosts in the inventory:

```
student@ansible-00-01-hivemaster:~$ vi playbook.yml
---
- name: Ping all hosts
  hosts: all
  tasks:
    - name: Ping task
      ping:
```

It is recommended that all the plays and tasks should have a name, in order to be easier to watch the execution steps in case of debugging or errors.

Run the playbook using `ansible-playbook` command:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook playbook.yml

PLAY [Ping all hosts]
*****
*****

TASK [Gathering Facts]
*****
*****
ok: [ubuntu]
ok: [hivemaster]
ok: [centos]

TASK [Ping task]
*****
*****
ok: [ubuntu]
ok: [hivemaster]
ok: [centos]
```

## PLAY RECAP

```
*****
*****
centos           : ok=2    changed=0    unreachable=0
failed=0        skipped=0  rescued=0    ignored=0
hivemaster      : ok=2    changed=0    unreachable=0
failed=0        skipped=0  rescued=0    ignored=0
ubuntu         : ok=2    changed=0    unreachable=0
failed=0        skipped=0  rescued=0    ignored=0
```

As you can see, the ping command succeeded for all 3 hosts in the inventory. If you want to see more details about playbook execution, you can specify `-v` (you can also use double, triple v) argument to set verbose mode.

```
student@ansible-00-01-hivemaster:~$ ansible-playbook playbook.yml -vv
ansible-playbook 2.9.1
  config file = /etc/ansible/ansible.cfg
  configured module search path =
[u'/home/student/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-
packages/ansible
  executable location = /usr/bin/ansible-playbook
  python version = 2.7.15+ (default, Oct 7 2019, 17:39:04) [GCC 7.4.0]
Using /etc/ansible/ansible.cfg as config file
```

## PLAYBOOK: playbook.yml

```
*****
*****
1 plays in playbook.yml
```

## PLAY [Ping all hosts]

```
*****
*****
```

## TASK [Gathering Facts]

```
*****
*****
```

```
task path: /home/student/playbook.yml:2
```

```
ok: [ubuntu]
```

```
ok: [hivemaster]
```

```
ok: [centos]
```

```
META: ran handlers
```

## TASK [Ping task]

```
*****
*****
```

```
task path: /home/student/playbook.yml:5
```

```
ok: [ubuntu] => {"changed": false, "ping": "pong"}
```

```
ok: [hivemaster] => {"changed": false, "ping": "pong"}
ok: [centos] => {"changed": false, "ping": "pong"}
META: ran handlers
META: ran handlers

PLAY RECAP
*****
*****
centos                : ok=2    changed=0    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
hivemaster        : ok=2    changed=0    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
ubuntu            : ok=2    changed=0    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
```

## Task 2: Add another play to existing playbook

You learned during lecture that a playbook contains at least one play, so we are going to add one more play to our playbook. Let's create a play which performs mongodb installation on our `dbservers`:

```
---
- name: Ping all hosts
  hosts: all
  tasks:
    - name: Ping task
      ping:

- name: Deploy mongodb for dbservers group
  hosts: dbservers
  become: true
  tasks:
    - name: Install mongo
      package:
        name: mongodb
        state: latest
      notify: restart mongodb

  handlers:
    - name: restart mongodb
      service:
        name: mongodb
        state: restarted
        enabled: yes
```

Notice that we introduced some new things in this play: we used `package` module to install mongodb, we targeted only `dbservers` group of hosts, the `become` parameter is set to "true", because we need a privileged user to perform package installation and we used a handler to restart and enable `mongodb` after installation.

## Commonly used modules

### Task 2: User and group modules

We are going to create a new playbook (called `modules.yml`) and we will start exploring different modules available in Ansible. Let's start with "user" module, which provides an option to manage user accounts and user attributes.

#### Task 2.1: Create a new user account

Let's add the first task of `modules.yml` playbook to create a new user account called "testuser" (set the playbook to affect all hosts in the inventory):

```
student@ansible-00-01-hivemaster:~$ vi modules.yml
---
- name: Modules Playbook
  hosts: all
  become: yes
  tasks:
    - name: Create "testuser"
      user:
        name: testuser
        state: present
```

Notice "become: yes" as we need to be root in order to add a new user.  
Run the playbook and watch the results:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook modules.yml

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

TASK [Gathering Facts]
*****
*****
ok: [ubuntu]
ok: [hivemaster]
ok: [centos]

TASK [Create "testuser"]
*****
*****
changed: [hivemaster]
changed: [ubuntu]
changed: [centos]
```

## PLAY RECAP

```
*****
*****
centos                : ok=2    changed=1    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
hivemaster         : ok=2    changed=1    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
ubuntu            : ok=2    changed=1    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
```

## Task 2.2: Set bash for “testuser”

Let’s edit the previous task and add also the `/bin/bash` as the default bash for our “testuser”:

```
student@ansible-00-01-hivemaster:~$ vi modules.yml
---
- name: Modules Playbook
  hosts: all
  become: yes
  tasks:
    - name: Create "testuser"
      user:
        name: testuser
        state: present
        shell: /bin/bash
```

Run the playbook again and see how many hosts are affected.

## Task 2.3: Create a new group

Using “user” module we can create new groups. So, we will create a new group called “testgroup”:

```
student@ansible-00-01-hivemaster:~$ vi modules.yml
---
- name: Modules Playbook
  hosts: all
  become: yes
  tasks:
    - name: Create "testuser"
      user:
        name: testuser
        state: present
        shell: /bin/bash

    - name: Create "testgroup"
      group:
        name: testgroup
        state: present
```

## Task 2.4: Add user to groups

In this task we will go back to the user module, in order to add “testuser” to the “testgroup”. The task will look like:

```
- name: Create "testuser"
  user:
    name: testuser
    state: present
    shell: /bin/bash
    groups: testgroup
```

Notice that we used “groups”, not just “group” – this sets the primary group.

We can also set the primary group for “testuser” to be “ansible” group:

```
- name: Create "testuser"
  user:
    name: testuser
    state: present
    shell: /bin/bash
    groups: testgroup
    group: ansible
```

Run the playbook and explore the tasks:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook modules.yml
[...]

TASK [Create "testuser"]
*****
*****
changed: [ubuntu]
changed: [hivemaster]
changed: [centos]
```

## Task 3: File module

Using “file” module we can create, set attributes or remove files and directories.

### Task 3.1: Create .ssh folder for “testuser”

We are going to create `/home/testuser/.ssh` folder and set owner/group and proper permissions (700) for it:

```
- name: Create '.ssh' folder
  file:
    path: /home/testuser/.ssh
```



```
state: directory
owner: testuser
group: ansible
mode: '0700'
```

Run the playbook and explore the tasks:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook modules.yml
[...]

TASK [Create ".ssh" dir]
*****
*****
changed: [ubuntu]
changed: [hivemaster]
changed: [centos]
```

### Task 3.2: Create `authorized_keys` file inside `.ssh`

Next, we have to create the `authorized_keys` file for our “testuser” in `/home/testuser/.ssh` and set owner/group and proper permissions (644) for it:

```
- name: Create 'authorized_keys' file under '.ssh'
  file:
    path: /home/testuser/.ssh/authorized_keys
    state: touch
    owner: testuser
    group: ansible
    mode: '0640'
```

Run the playbook and explore the tasks:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook modules.yml
[...]

TASK [Create 'authorized_keys' file under '.ssh']
*****
*****
changed: [ubuntu]
changed: [hivemaster]
changed: [centos]
```

### Task 4: Copy module

This module copies a file from `local` or `remote` source to a `remote` destination. The source location can be specified using `remote_src` parameter which by default is set to no (so it copies from the local machine).

#### Task 4.1: Copy ansible\_key.pub to authorized\_keys file

This is the last step to provide to our “testuser” the option to be able to login using SSH:

```
- name: Copy key to 'authorized_keys'
  copy:
    src: /home/student/ansible_key.pub
    dest: /home/testuser/.ssh/authorized_keys
    owner: testuser
    group: ansible
```

#### Task 4.2: Test authentication

You can test SSH authentication for “testuser” from command line:

```
student@ansible-00-01-hivemaster:~$ ssh -i ansible_key testuser@ansible-00-02-ubuntu
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.0.0-1025-gcp x86_64)
[...]
Last login: Mon Jan 01 23:14:53 2020 from 10.128.0.48
testuser@ansible-00-02-ubuntu:~$
testuser@ansible-00-02-ubuntu:~$ exit
```

If you want to test using Ansible you have to edit in /etc/ansible/hosts the ansible\_user variable for our hosts (in this example I will add this variable for ubuntu host only):

```
student@ansible-00-01-hivemaster:~$ sudo vi /etc/ansible/hosts
ubuntu ansible_host="10.128.0.49" ansible_user=testuser
```

Now let's see if Ansible can login using “testuser”:

```
student@ansible-00-01-hivemaster:~$ ansible ubuntu -a "id"
ubuntu | CHANGED | rc=0 >>
uid=1005(testuser) gid=1005(ansible)
groups=1005(ansible),1007(testgroup)
```

Revert back to the original user (ansible) for in /etc/ansible/hosts.

#### Task 5: Lineinfile module

Using this module, we can add (or replace) a line to a file, using regular expressions to make sure that if the line is not going to be duplicated if it exists.

##### Task 5.1: Add “testgroup” to sudoers file

Because editing sudoers file can get us locked out of the system, we are going to use also a validation tool for this file, using `visudo`:

```
- name: Validate the sudoers file before saving
  lineinfile:
    path: /etc/sudoers
    state: present
    regexp: '^%testgroup\s'
    line: '%testgroup ALL=(ALL) NOPASSWD: ALL'
    validate: /usr/sbin/visudo -cf %s
```

After running the playbook, you can try to use “testuser” to “cat” the contents of “/etc/shadow”. Notice that you have to temporary modify `ansible_user` (from `/etc/ansible/hosts`) to be “testuser” for at least one host. Same steps like in [Task 4.2](#)

Otherwise you can use the following ad-hoc command:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m shell -a
"cat /etc/shadow" -u testuser --private-key /home/student/ansible_key --
become
```

## Raw vs Command vs Shell

These are the most common modules for executing commands on remote hosts. Each module has its own advantages and disadvantages. We also used them until now during this training, but right now we are going to see the main differences between them.

### Task 6: Command module

This module can only execute binaries from the remote hosts and it is the default module in Ad-Hoc mode. Command module won't be impacted by local shell variables since it bypasses the shell. At the same time, it may not be able to run “shell” inbuilt functions (like `set`) and redirection (which is also shell's inbuilt functionality).

```
student@ansible-00-01-hivemaster:~$ ansible all -a "who"
hivemaster | CHANGED | rc=0 >>
student   pts/0          2019-11-25 15:01 (82.137.13.156)
student   pts/1          2019-11-25 15:44 (82.137.13.156)
ansible   pts/4          2019-11-26 14:35 (10.128.0.48)

ubuntu | CHANGED | rc=0 >>
ansible   pts/0          2019-11-26 14:35 (10.128.0.48)

centos | CHANGED | rc=0 >>
ansible   pts/0          Nov 26 14:35 (ansible-00-01-
hivemaster.training.sass.ro)
```

This command executed the `/bin/who` binary from the remote servers and returned the results. Notice that Python is required, otherwise command module will fail.

### Task 7: Raw module

There are several distributions which doesn't have Python preinstalled, and managing them with Ansible would be impossible without this module which doesn't require Python to be installed (in fact, raw module is usually used to install Python on remote hosts), because it executes a low-down SSH command over the network. Another possible usage is in case of managing network devices that don't have Python installed.

```
student@ansible-00-01-hivemaster:~$ ansible all -m raw -a "who"
hivemaster | CHANGED | rc=0 >>
student pts/0      2019-11-25 15:01 (82.137.13.156)
student pts/1      2019-11-25 15:44 (82.137.13.156)
ansible pts/5      2019-11-26 14:48 (10.128.0.48)
Shared connection to 10.128.0.48 closed.

centos | CHANGED | rc=0 >>
ansible pts/0      Nov 26 14:48 (ansible-00-01-
hivemaster.training.sass.ro)
Shared connection to 10.142.15.213 closed.

ubuntu | CHANGED | rc=0 >>
ansible pts/0      2019-11-26 14:48 (10.128.0.48)
Shared connection to 10.128.0.49 closed.
```

### Task 7: Shell module

Shell module is very useful when you want to use redirections and shell's inbuilt functionality, as the command module doesn't support these.

```
student@ansible-00-01-hivemaster:~$ ansible all -m shell -a "df -h |
grep sda"
hivemaster | CHANGED | rc=0 >>
/dev/sda1      20G  1.6G   18G    9% /
/dev/sda15     105M  3.6M  101M    4% /boot/efi

ubuntu | CHANGED | rc=0 >>
/dev/sda1      20G  1.4G   18G    8% /
/dev/sda15     105M  3.6M  101M    4% /boot/efi

centos | CHANGED | rc=0 >>
/dev/sda1      20G  2.0G   19G   10% /
```

Let's try to run the same command using command module:

```
student@ansible-00-01-hivemaster:~$ ansible all -a "df -h | grep sda1"
hivemaster | FAILED | rc=1 >>
df: '|': No such file or directory
[...]
```

Now let's try also to redirect the output of a command:

```
student@ansible-00-01-hivemaster:~$ ansible all -m shell -a "cat
/etc/hosts > /tmp/hosts.txt"
hivemaster | CHANGED | rc=0 >>

ubuntu | CHANGED | rc=0 >>

centos | CHANGED | rc=0 >>
```

You may also try this using command module.

#### Task 8: "Command-line" modules in Playbooks

We used these 3 modules in Ad-Hoc mode until now, but you have to know that it is also possible to use them in playbooks. Let's create a new playbook and add some tasks:

```
student@ansible-00-01-hivemaster:~$ vi command_modules.yml
---
- name: Command modules Playbook
  hosts: all
  become: yes
  tasks:
    - name: Raw module
      raw: cat /etc/hosts
      register: raw_output

    - name: Shell module
      shell: ls -l /var/log | grep log > /tmp/tmp.log

    - name: Command module
      command: cat /etc/shadow
      register: cmd_output

    - name: Print raw output
      debug:
        var: raw_output

    - name: Print cmd output
      debug:
        var: cmd_output
```

Run the playbook and explore the content of `/tmp/tmp.log` for shell module output. Notice that for `raw` and `command` we registered the output in 2 variables and then printed those variables. You are going to learn more about variables in the next lab.

## Task results

### Task 9: OK vs Changed vs Failed

As you already seen, every task of a play returns a status. This status is aiming to give the user a feedback about whether the task succeeded or not for a certain host (also if it performed a change or not).

Let's start with the playbook from previous task (`command_modules.yml`). Running the playbook again you will notice that it returns that all 3 command modules returned the `changed` status.

```
student@ansible-00-01-hivemaster:~$ ansible-playbook command_modules.yml

PLAY [Command modules Playbook]
*****
*****

TASK [Gathering Facts]
*****
*****
ok: [hivemaster]
ok: [ubuntu]
ok: [centos]

TASK [Raw module]
*****
*****
changed: [ubuntu]
changed: [hivemaster]
changed: [centos]

TASK [Shell module]
*****
*****
changed: [hivemaster]
changed: [ubuntu]
changed: [centos]

TASK [Command module]
*****
*****
changed: [ubuntu]
changed: [hivemaster]
changed: [centos]
```

## PLAY RECAP

```
*****
*****
centos                : ok=4    changed=3    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
hivemaster         : ok=4    changed=3    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
ubuntu            : ok=4    changed=3    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
```

This happens also if we run the playbook several times, which means that our tasks are not idempotent. In case of command and shell modules it is more difficult to make them idempotent, as they run a binary on the remote host.

In order to achieve idempotency an useful feature is “creates” (used in this case for shell module – it can also be used for command module). `Creates` (there is also `removes`) won’t create the file if it already exists (so if you run the playbook 2 times, you will notice that second time will just return `ok`:

```
---
- name: Command modules Playbook
  hosts: all
  become: yes
  tasks:
    - name: Raw module
      raw: cat /etc/hosts
      register: raw_output

    - name: Shell module
      shell: ls -l /var/log | grep log > /tmp/tmp.log
      args:
        creates: /tmp/tmp.log

    - name: Command module
      command: cat /etc/shadow
      register: cmd_output

# - name: Print raw output
#   debug:
#     var: raw_output

# - name: Print cmd output
#   debug:
#     var: cmd_output
```

Notice that I commented the variables printing tasks (in order to be easier to watch the playbook results).

Run again the playbook:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook command_modules.yml
```

```
PLAY [Command modules Playbook]
*****

TASK [Gathering Facts]
*****

ok: [ubuntu]
ok: [hivemaster]
ok: [centos]

TASK [Raw module]
*****

changed: [ubuntu]
changed: [hivemaster]
changed: [centos]

TASK [Shell module]
*****

ok: [hivemaster]
ok: [ubuntu]
ok: [centos]

TASK [Command module]
*****

changed: [ubuntu]
changed: [hivemaster]
changed: [centos]

PLAY RECAP
*****

centos                : ok=4    changed=2    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
hivemaster       : ok=4    changed=2    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
ubuntu           : ok=4    changed=2    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
```

Notice that usually Ansible modules are idempotent, which means that running a task several task will report changed just for the first run.

## Task 10: Setting password for a user

Create a new playbook and a task which performs the setting of a password for out “testuser”:



```
student@ansible-00-01-hivemaster:~$ vi password.yml
---
- name: User management
  hosts: all
  become: yes
  vars:
    user_pass: 123abc
  tasks:
    - name:
      user:
        name: testuser
        password: "{{ user_pass | password_hash('sha512') }}"
```

Run the playbook (several times) and explore tasks reports:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook password.yml

PLAY [User management]
*****
*****

TASK [Gathering Facts]
*****
*****
ok: [hivemaster]
ok: [ubuntu]
ok: [centos]

TASK [Setting password]
*****
*****
changed: [hivemaster]
changed: [ubuntu]
changed: [centos]

PLAY RECAP
*****
*****
centos                : ok=2    changed=1    unreachable=0
failed=0      skipped=0      rescued=0    ignored=0
hivemaster        : ok=2    changed=1    unreachable=0
failed=0      skipped=0      rescued=0    ignored=0
ubuntu            : ok=2    changed=1    unreachable=0
failed=0      skipped=0      rescued=0    ignored=0
```

You probably noticed that every time the task “Setting password” returns changed, even if the password is the same (and I said earlier that Ansible modules are idempotent, so it should return OK from the second run). This is due to the fact that in Linux passwords are salted with a random `salt`, so, the salted-password is in fact a new one.