



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

Loops. Conditionals.

Table of Contents

Lab Overview and objectives	2
<i>Guided Tasks</i>	2
Task 1: loop vs with_<lookup>	2
Task 1.1: with_items	2
Task 1.2: loop	3
Task 1.3: Registering variables with a loop	4
Task 1.4: Iterate over inventory file	5
Task 2: Conditionals	6
Task 2.1: "when: var == true"	6
Task 2.2: "when" file exists	7
Task 2.3: "when" multiple conditions	7
Task 2.4: "when" with Facts	8
Task 2.5: Loop and "when" on the same task using multiple conditions	10
Task 2.6: Setup webserver and dbserver	11

Lab Overview and objectives

The purpose of this lab is to get used with Ansible loops and conditionals. You are probably familiar with the concept of loop from computer programming, but Ansible loops include changing ownership on files or directories with the file module, creating multiple users with the user module and also taking decisions based on conditional clauses.

Guided Tasks

Task 1: loop vs with_<lookup>

Ansible provides two methods for creating loops: `loop` and `with_<lookup>`. Loop was added in Ansible 2.5 and it doesn't replaced the older `with_<lookup>` method (until now), but it is the recommended way to implement loops.

Task 1.1: with_items

Let's use `with_items` to install/uninstall a list of packages from our `hivemaster` host:

```
student@ansible-00-01-hivemaster:~$ vi with_items.yml
---
- name: Manage packages
  hosts: hivemaster
  become: true
  tasks:
    - name: Install packages
      apt:
        name: "{{ item.name }}"
        state: "{{ item.state }}"
      with_items:
        - { name: wget, state: present }
        - { name: nmap, state: present }
        - { name: apache2, state: absent }
```

Notice the apt module used right here, which is the package manager for Debian distribution.

Running the playbook should result in installing nmap (as wget is already installed and apache2 is absent):

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

PLAY [Manage packages]
*****
```

```
TASK [Gathering Facts]
*****
ok: [hivemaster]

TASK [Install packages]
*****
ok: [hivemaster] => (item={u'state': u'present', u'name': u'wget'})
changed: [hivemaster] => (item={u'state': u'present', u'name': u'nmap'})
ok: [hivemaster] => (item={u'state': u'absent', u'name': u'apache2'})

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

Task 1.2: loop

We are going to use `loop` in this example to iterate through a simple list and also through a dictionary:

```
student@ansible-00-01-hivemaster:~$ vi loop_1.yml
---
- name: Test loop over list and dict
  hosts: hivemaster
  gather_facts: no
  vars:
    my_grocery_list:
      - milk
      - bread
      - cereal
      - beer
    my_car_preferences:
      brand: mercedes
      model: G
      year: 2018
  tasks:
    - name: Loop over list
      debug:
        msg: "{{ item }}"
      loop: "{{ my_grocery_list }}"

    - name: Loop over dict
      debug:
        msg: "{{ item.key }} -> {{ item.value }}"
      loop: "{{ my_car_preferences | dict2items }}"
```

Run the playbook:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook loop_1.yml

PLAY [Test loop over list and dict]
*****
```

```
TASK [Loop over list]
*****
ok: [hivemaster] => (item=milk) => {
  "msg": "milk"
}
ok: [hivemaster] => (item=bread) => {
  "msg": "bread"
}
ok: [hivemaster] => (item=cereal) => {
  "msg": "cereal"
}
ok: [hivemaster] => (item=beer) => {
  "msg": "beer"
}

TASK [Loop over dict]
*****
ok: [hivemaster] => (item={'key': u'brand', 'value': u'mercedes'}) => {
  "msg": "brand -> mercedes"
}
ok: [hivemaster] => (item={'key': u'model', 'value': u'G'}) => {
  "msg": "model -> G"
}
ok: [hivemaster] => (item={'key': u'year', 'value': 2018}) => {
  "msg": "year -> 2018"
}
```

Task 1.3: Registering variables with a loop

When you use `register` with a `loop`, the variable registered will contain a list of all responses from the module. This differs from the data structure returned when using `register` without a loop:

```
student@ansible-00-01-hivemaster:~$ vi register_loop.yml
---
- name: Register var with a loop
  hosts: hivemaster
  gather_facts: no
  tasks:
    - name: Print some values
      shell: "echo {{ item }}"
      loop:
        - value1
        - value2
        - value3
      register: output

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

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

```
PLAY [Register var with a loop]
*****
TASK [Print some values]
*****
changed: [hivemaster] => (item=value1)
changed: [hivemaster] => (item=value2)
changed: [hivemaster] => (item=value3)

TASK [Print output]
*****
ok: [hivemaster] => {
  "output": {
    "changed": true,
    "msg": "All items completed",
    "results": [
      {
        "ansible_facts": {
          "discovered_interpreter_python": "/usr/bin/python3"
        },
        "ansible_loop_var": "item",
        "changed": true,
        "cmd": "echo value1",
        "delta": "0:00:00.002954",
        "end": "2019-11-26 20:57:40.161126",
        "failed": false,
        "invocation": {
          "module_args": {
            "_raw_params": "echo value1",
            "_uses_shell": true,
            "argv": null,
            "chdir": null,
            "creates": null,
            "executable": null,
            "removes": null,
            "stdin": null,
            "stdin_add_newline": true,
            "strip_empty_ends": true,
            "warn": true
          }
        },
        "item": "value1",
        "rc": 0,
        "start": "2019-11-26 20:57:40.158172",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "value1",
        "stdout_lines": [
          "value1"
        ]
      }
    ]
  }
}
```

Task 1.4: Iterate over inventory file

We can also use loop to iterate over our hosts from the inventory file (all of them or just some groups):

```
student@ansible-00-01-hivemaster:~$ vi loop_inventory.yml
---
- name: Loop over inventory file
  hosts: hivemaster
  gather_facts: no
  tasks:
    - name: Print inventory hosts
      debug:
        msg: "{{ item }}"
      loop: "{{ groups['webservers'] }}"
```

We looped over hosts from ‘webservers’ group:

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

PLAY [Loop over inventory file]
*****
TASK [Print inventory hosts]
*****
ok: [hivemaster] => (item=ubuntu) => {
    "msg": "ubuntu"
}
ok: [hivemaster] => (item=centos) => {
    "msg": "centos"
}
[...]
```

Task 2: Conditionals

Ansible provides a method for implementing conditionals using the “when” clause. The syntax is simple “when: condition” and we can use this with any module to control when they are executed.

Task 2.1: “when: var == true”

Let’s suppose that we want to backup SSH configuration for a host when some variable (called “backup” is set to “true”). Also make sure that you customize the backup including the name of the host on the backup file so that the file doesn’t get overwritten if we run the playbook against several hosts on the same time.

```
student@ansible-00-01-hivemaster:~$ vi when_clause_1.yml
---
- name: Using conditionals
  hosts: centos
  gather_facts: yes
  become: true
  vars:
    backup: true
```

```
tasks:
- name: Backup ssh configuration
  fetch:
    src: /etc/ssh/sshd_config
    dest: ./sshd_config-{{ ansible_hostname }}
    flat: yes
    when: backup == true
```

Task 2.2: “when” file exists

We can check if a file (or directory) exists using “stat” module. Then we can customize the execution of the playbook using the result from the stat module. So, let’s start from the previous example and create the backup of SSH configuration only if the sshd_config file exists (make sure that you remove the previous backup from your hivemaster host, otherwise Ansible will just return ok):

```
student@ansible-00-01-hivemaster:~$ rm -rf sshd_config-ansible-00-03-centos
student@ansible-00-01-hivemaster:~$ vi when_clause_2.yml
---
- name: Using conditionals
  hosts: centos
  gather_facts: yes
  become: true
  vars:
    backup: true
  tasks:
  - stat:
    path: /etc/ssh/sshd_config
    register: result

  - name: Backup ssh configuration
    fetch:
      src: /etc/ssh/sshd_config
      dest: ./sshd_config-{{ ansible_hostname }}
      flat: yes
      when: result.stat.exists
```

We used in this example parameters “exists”, which returns if the destination path exists or not. There are also other parameters like “isdir”, “isreg” which returns if the path is a directory or a regular file. You can read more about stat module at https://docs.ansible.com/ansible/latest/modules/stat_module.html

Task 2.3: “when” multiple conditions

Starting from the same example, let’s condition the backup on more conditions at once:

```
---
- name: Using conditionals
  hosts: centos
```



```
gather_facts: yes
become: true
vars:
  backup: true
tasks:
- stat:
    path: /etc/ssh/sshd_config
    register: result

- name: Backup ssh configuration
  fetch:
    src: /etc/ssh/sshd_config
    dest: ./sshd_config-{{ ansible_hostname }}
    flat: yes
    when: result.stat.exists and result.stat.isreg and backup == true
```

In this example we used “**and**” operator, but it also possible (as you expect) to use “**or**” operator with **when** clause. So, let’s suppose that we want to make the backup on every “27” day of the month even if the **backup** variable is not set to true:

```
---
- name: Using conditionals
  hosts: centos
  gather_facts: yes
  become: true
  vars:
    backup: false
  tasks:
- stat:
    path: /etc/ssh/sshd_config
    register: result

- name: Backup ssh configuration
  fetch:
    src: /etc/ssh/sshd_config
    dest: ./sshd_config-{{ ansible_hostname }}
    flat: yes
    when: (result.stat.exists and result.stat.isreg and backup == true)
or (ansible_date_time.day == "27")
```

Task 2.4: “when” with Facts

We can also use **Ansible facts** as conditions for “when” clause, so let’s reboot all hosts from inventory that are running CentOS version 7:

```
student@ansible-00-01-hivemaster:~$ vi when_clause_3.yml
---
- name: Reboot all hosts running CentOS 7
  hosts: all
```

```
gather_facts: yes
become: true
tasks:
- name: Rebooting...
  reboot:
    msg: "System is going to reboot now!"
    reboot_timeout: 40
    test_command: uptime
    when: ansible_facts['distribution'] == "CentOS" and
ansible_facts['distribution_major_version'] == "7"
    register: result

- name: Print registered result
  debug:
    var: result
```

In this example we used the reboot module, with a few parameters: `msg` is the message to be displayed for users, `reboot_timeout` is the period of time (in seconds) to wait for the machine to reboot and respond to a test command, and `test_command` which is set by default to `"whoami"`. Notice that we also registered the output of the task as a variable called `result` and displayed in the next task its content:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook when_clause_3.yml

PLAY [Reboot all hosts running CentOS 7]
*****

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

TASK [Rebooting...]
*****
skipping: [ubuntu]
skipping: [hivemaster]
changed: [centos]

TASK [Print registered result]
*****
ok: [ubuntu] => {
  "result": {
    "changed": false,
    "skip_reason": "Conditional result was False",
    "skipped": true
  }
}
ok: [centos] => {
  "result": {
    "changed": true,
    "elapsed": 27,
```

```

        "failed": false,
        "rebooted": true
    }
}
ok: [hivemaster] => {
    "result": {
        "changed": false,
        "skip_reason": "Conditional result was False",
        "skipped": true
    }
}

```

PLAY RECAP

```

*****
centos           : ok=3    changed=1    unreachable=0    failed=0
skipped=0       rescued=0  ignored=0
hivemaster       : ok=2    changed=0    unreachable=0    failed=0
skipped=1       rescued=0  ignored=0
ubuntu          : ok=2    changed=0    unreachable=0    failed=0
skipped=1       rescued=0  ignored=0

```

Running this playbook you will notice another output status “**skipped**” for `ubuntu` and `hivemaster` (we already know “**OK**”, “**changed**” and “**failed**” from a previous lab) which is returned because “**Conditional result was False**”, but this is exactly what we intended to do (apply the reboot task only for the host(s) running CentOS 7).

Task 2.5: Loop and “when” on the same task using multiple conditions

```

---
- name: Install several packages on web servers
  hosts: web servers
  gather_facts: yes
  vars:
    my_var: nginx
    my_list:
      - curl
      - wget
      - htop
      - nginx

  tasks:
    - name: Loop with conditional clause
      debug:
        msg: "{{ item }}"
      loop: "{{ my_list }}"
      when: item == my_var

```

Notice that this playbook does not install these packages, it just iterates through `my_list` and prints the `item` name when condition is met.

Task 2.6: Setup webserver and dbserver

Our hosts have 2 different distributions (Debian and RedHat). As you already seen, they use different package managers (`apt` and `yum`), so we have to apply some conditions if we want to run the same installation playbook against both of them. There is also a module called `package` which can be used for both distributions, but just in case that the package has the same name.

Let's say that we want to install **Apache** on our `webserver`s (Apache package is called `apache2` on Ubuntu and `httpd` on CentOS). For our `dbserver`s we are going to install `MySQL`.

```
student@ansible-00-01-hivemaster:~$ vi installer.yml
---
- name: Install packages on webserver and dbserver
  hosts: all
  become: yes
  gather_facts: yes
  tasks:
    - name: Set proper name for apache package
      set_fact:
        apache_package: "apache2"
      when: ansible_facts['os_family'] == "Debian"

    - name: Set proper name for apache package
      set_fact:
        apache_package: "httpd"
      when: ansible_facts['os_family'] == "RedHat"

    - name: Install packages for webserver
      package:
        name:
          - "{{ apache_package }}"
          - wget
          - curl
          - htop
        state: latest
        notify: start apache
        when: "'webserver' in group_names"

    - name: Install packages for dbserver
      package:
        name:
          - python-mysqldb
          - mysql-server
        update_cache: yes
        state: latest
        when: "'dbserver' in group_names"

  handlers:
    - name: start apache
      service:
        name: "{{ apache_package }}"
```

```
enabled: yes
state: started
```

This is a more complex playbook, because it performs the installation for all packages necessary for our groups. We set `apache_package` fact for every host according to the distribution and installed the packages using `general_package` module.

Notice that we didn't use a loop explicitly, but we passed a list of packages to the `package` module, as this is the recommended way to install several several packages at one time (loop is considerably slower in this case).

We also used conditional clauses to install packages just for the hosts that belong to certain groups.

Running the playbook should perform the proper installation of all packages:

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

PLAY [Install packages on webservers and dbservers]
*****

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

TASK [Set proper name for apache package]
*****
ok: [ubuntu]
skipping: [centos]
ok: [hivemaster]

TASK [Set proper name for apache package]
*****
skipping: [ubuntu]
ok: [centos]
skipping: [hivemaster]

TASK [Install packages for webservers]
*****
skipping: [hivemaster]
changed: [ubuntu]
changed: [centos]

TASK [Install packages for dbservers]
*****
skipping: [centos]
changed: [hivemaster]
changed: [ubuntu]
[...]
```

Notice that we also added a `handler` to start and enable `apache` after installation. This handler is called only when the task is returning changed!

If you encounter any errors with `ubuntu` host during installation of packages please login using ssh and perform a “sudo apt update”, because this is not available using package module.