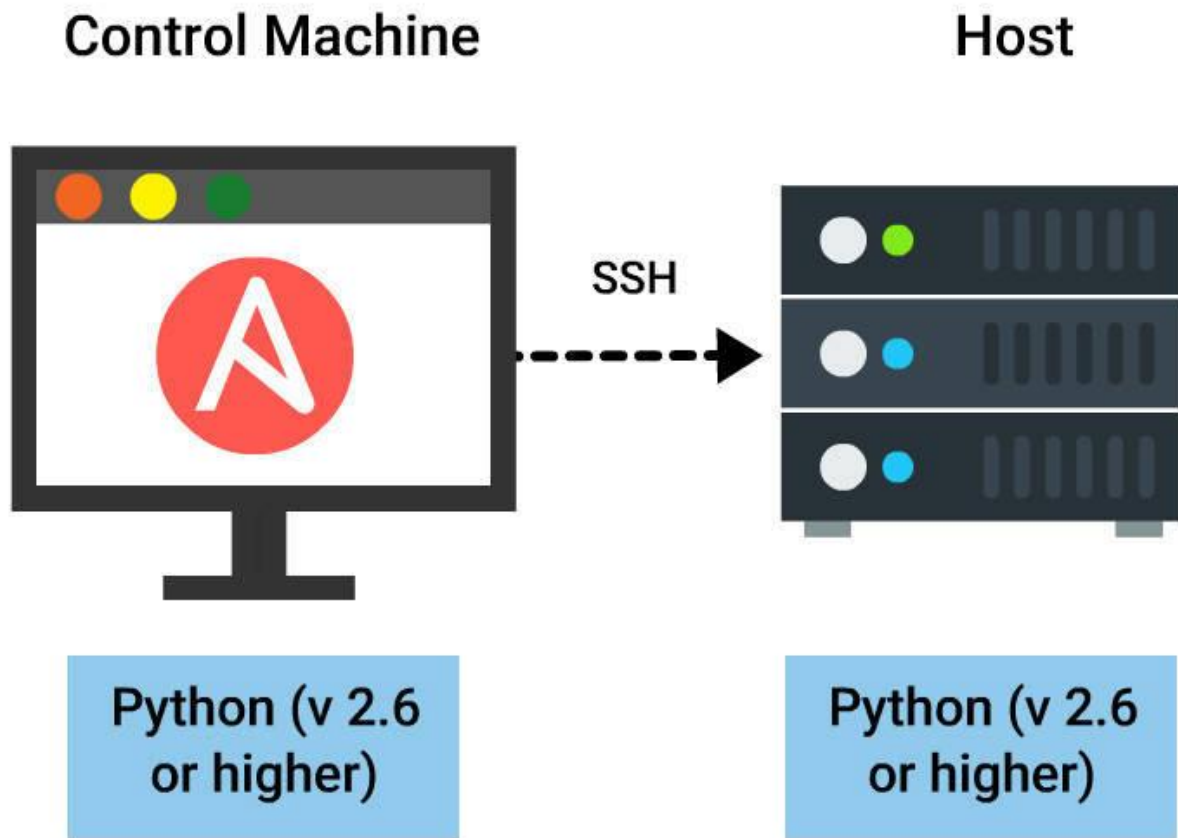


Environment at a Glance



Control Machine Requirements

- Linux/Mac OS (Windows not supported)
- Python (2.6 or later version) should be installed.
- Install Ansible.

Host Requirements

- Only Python needs to be installed.
- Uses SSH to connect to control machine.
- SFTP is used else scp can be configured.

Step By Step

You need to follow these steps to setup environment on your system

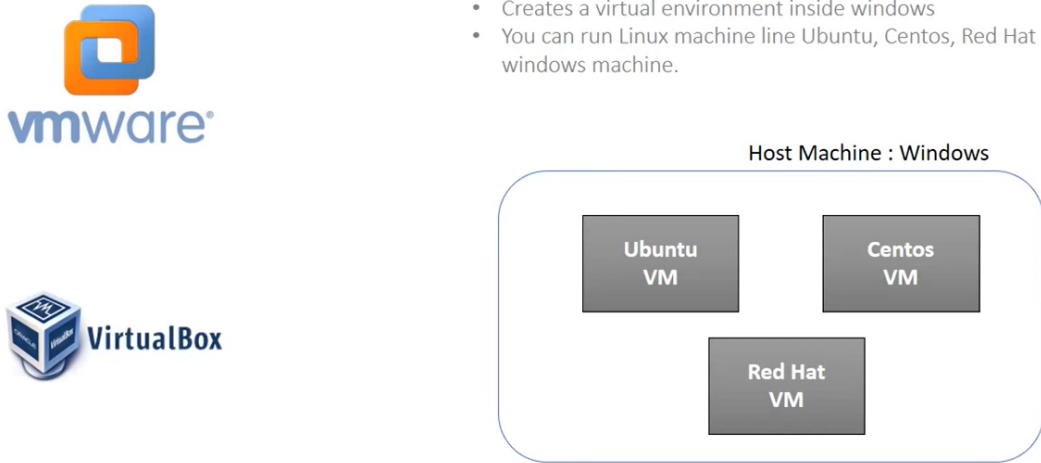
Install VirtualBox.

Install Vagrant. Vagrant enables virtual software development environment.

Install Ansible.

Creating development env in vagrant .

Virtual Machines



- Creates a virtual environment inside windows
- You can run Linux machine line Ubuntu, Centos, Red Hat inside windows machine.


Host Machine : Windows

is why there is a need for virtual machine so most of the people are having

Chapter 2 / 4

www.selftuts.com

We can use virtual box and run virtual machine of any os in it . We can run images of virtual machine in virtual box .



Provider

- <https://www.vagrantup.com/downloads.html>
- Provide a Command line tool called vagrant
- <https://www.virtualbox.org/wiki/Downloads>

Download vagrant > We will get command line tool to work with virtual box .

Install Vagrant

[Download and install Vagrant](#)

- **Identify the OS** you want to install in your virtual system. Accordingly, you need to find the [Vagrant Box](#) (let us consider Ubuntu on both our control and host machines).
- **Create a folder on your local machine**, to keep your Vagrant Boxes.

Now, **open your terminal** and move to the path where you created your folder for Vagrant boxes. Then run the following commands:

- `$ vagrant init ubuntu/trusty64`: Initialises vagrant file inside the folder
- `$ vagrant up --provider VirtualBox`: will download necessary files of your virtual machine
- `$ vagrant ssh`: will make secure connection with virtual machine

Vagrant boxes : Vm that we create are vagrant boxes . We can create multiple vms using vagrant box (eg : ubuntu / fedora) We can download vagrant box of specific os and vms can be created . We can download vagrant boxes from hashicorp site . Using vagrant command line tool we can add boxes .

- Vagrant follow the concept of boxes.
- Boxes are base images which is used to clone virtual machines

Vagrant Box repository

- <https://atlas.hashicorp.com/boxes/search>



Adding a new box

- Vagrant box add user/boxname

We can also create our own box and add softwares to it .

```
{ ~ } » vagrant box add olbat/tiny-core-micro
==> box: Loading metadata for box 'olbat/tiny-core-micro'
box: URL: https://atlas.hashicorp.com/olbat/tiny-core-micro
==> box: Adding box 'olbat/tiny-core-micro' (v0.1.0) for provider: virtualbox
box: Downloading: https://atlas.hashicorp.com/olbat/boxes/tiny-core-micro/versions/0.1.0/providers/virtualbox.box
box:
==> box: Successfully added box 'olbat/tiny-core-micro' (v0.1.0) for 'virtualbox'!
{ ~ } » vagrant box list
boxcutter/ol73      (virtualbox, 3.0.9)
centos/7            (virtualbox, 1611.01)
olbat/tiny-core-micro (virtualbox, 0.1.0)
ubuntu/trusty64     (virtualbox, 20170313.0.1)
{ ~ } » vagrant box remove olbat/tiny-core-micro
Removing box 'olbat/tiny-core-micro' (v0.1.0) with provider 'virtualbox'...
{ ~ } »
{ ~ } »
{ ~ } »
{ ~ } »
{ ~ } » vagrant box list
boxcutter/ol73      (virtualbox, 3.0.9)
centos/7            (virtualbox, 1611.01)
ubuntu/trusty64     (virtualbox, 20170313.0.1)
{ ~ } »
```

Install Ansible

Once you are done with Virtual Box and Vagrant setup, you only need to run one last command in your terminal to install Ansible on the control machine.

On Ubuntu/Debian/Linux Mint

```
$ sudo apt-get install ansible
```

On RHEL/CentOS/Fedora

```
$ sudo yum install epel-release
```

```
$ sudo yum install ansible
```

To verify if Ansible is successfully installed or not: `ansible --version`

Writing Playbook

Let us now write a Playbook to print **Hello World**. You can follow the steps parallelly.

Step By Step

- **Open** any environment
- Enter **commands given in Step 2 of 6**. This makes an SSH connection to your host.
- Open Playbook using the **vi editor** (eg: `vi demo.yml`).
- **Write** your playbook (given in next card).
- **Press Escape** key on your keyboard.
- **Save and close editor** using `:wq`(w: write, q: quit).
- **Run your playbook:** `$ ansible-playbook -i myhosts demo.yml`.

Ansible has a shell module too. This module is used to run shell commands on target systems. The Ansible shell module allows the user to run complex commands with redirection, pipes, etc. It takes a command name, its arguments with white space delimiters and runs it on remote hosts.

It may sound like the exact same thing as the Ansible command module, but the difference is that it runs the commands on the host using a shell. The shell module also has access to environment variables and special operators such as `| < > & ;` etc. Even better, you can run entire scripts using the shell module. Nonetheless, it is common knowledge among Ansible users that the command module is a safer and more predictable option than the shell module.

Finally, it is important to keep in mind that this module only works with Linux systems. Windows users can use `ansible.windows.win_shell` in its place.

Sample Playbook 1 - "demo.yml"

- name: this play displays "hello world"

hosts: all

tasks:

- name: displaying "hello world"

shell: echo "hello world"

- name: displaying wishes for the day

shell: echo "have a good day"

Playbook Step 1 - Name and Hosts

- name: this play displays "hello world"

hosts: all

- A Playbook always starts with three dashes ---.
- name tells the name of the play.
- hosts tell the list of hosts on which this play will be played.

Playbook Step 2 - Tasks

tasks:

- name: displaying "hello world"

shell: echo "hello world"

- name: displaying wishes for the day

shell: echo "have a good day"

name is optional but is always recommended as it improves readability

shell: echo "hello world" is a single task. This executes shell module and calls its echo argument to display the message written

Playbook Step 3 - Run Your Playbook

Type `ansible-playbook -i myhosts demo.yml` from the terminal to run your Playbook.

- `ansible-playbook` is the command to execute your Playbook
- `-i myhosts` tell the inventory name is *myhosts*

- demo.yml is the playbook that needs to be executed

Output Of "demo.yml"

This is how your Playbook would run :

```
PLAY [this play displays "hello world"] *****
GATHERING FACTS *****
ok: [host01]
TASK: [displaying "hello world"] *****
changed: [host01]
TASK: [displaying wishes for the day] *****
changed: [host01]
PLAY RECAP *****
host01                : ok=3    changed=2    unreachable=0    failed=0
```

```
$ ansible --version
bash: ansible: command not found
$ sudo apt-get install ansible
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be insta
  ieee-data python-babel-localedata python3-arg
python3-chardet python3-distutils python3-dns
python3-jmespath python3-kerberos python3-lib
python3-markupsafe python3-netaddr python3-nt
python3-requests python3-requests-kerberos py
python3-selinux python3-simplejson python3-tz
python3-yaml tzdata
Suggested packages:
```

```
$ ansible --version
ansible 2.10.8
  config file = None
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/an
odules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0]
```

Sample Playbook 2 - Install Apache

Let us now take another example to install apache in your hosts.

This is how your Playbook would look:

- name: install apache

hosts: all

sudo: yes

tasks:

- name: install apache2

apt: name=apache2 update_cache=yes state=latest

Save and run this playbook as apache.yml in Katacoda and check the output.

Please note that for installing in *RHEL/CentOS/Fedora*,yum module is used instead of apt.

Restructuring Playbook

You can define the same playbook as:

- name: install apache

hosts: all

sudo: yes

tasks:

- name: install apache2

apt:

name: apache2

update_cache: yes

state: latest

- observe **colon : is used while structuring arguments vertically**, whereas equal to sign = is used while structuring arguments horizontally

Both ways of structuring your playbook is fine.

- **This vertical structuring of arguments is not a list, as list starts with dash sign -.**

Here **name, update_cache** and **state** are arguments of module **apt**, hence they do not start with -.

Handlers

Handlers are special tasks that run only on certain triggers like notify keyword.

- Handlers run mostly at the **end of the play**.
- Handlers **run only once** even if you run the playbook multiple times.

This is how a handler would look:

- name: install apache

.....

tasks:

- name: install apache2

.....

notify:

- start Apache

.....

handlers:

- name: start Apache

Question Type: Single-Select

What is the output statement of the following snippet?

```
tasks:
- name: test
  shell: echo "hello"
  register: a
  debug: msg="the message is {{ a.stdout }}"
```

The message 'hello' will get printed without any errors

Syntax error because of conflicting action statements

A Playbook starts with three dots

False

True

Which command do you use to do a syntax check on your playbook?

ansible-playbook <playbook_name> --syntax-check

ansible-playbook <playbook_name> -syntax--check

ansible-playbook <playbook_name> -syntax-check

ansible-playbook <playbook_name> --syntax--check

*Recollect while reading the section on setting up your environment or while installing an application , **you need different modules (apt and yum) to execute task.***

What will you do, if you have to configure 50 such servers with different OS and requirements?

This is where conditionals can be used to decide and control the execution flow in Ansible.

Conditionals - When Clause

When clause in Ansible is a raw jinja2 expression that defines the condition which will be evaluated for TRUE or FALSE.

tasks:

- name: "shutdown Debian flavored systems"
- command: /sbin/shutdown -t now
- when: ansible_os_family == "Debian"

Conditionals - When Clause

- You can **group conditions using parenthesis ()**

tasks:

- name: "shutdown CentOS 6 and Debian 7 systems"
- command: /sbin/shutdown -t now
- when: (ansible_distribution == "CentOS" and ansible_distribution_major_version == "6") or
(ansible_distribution == "Debian" and ansible_distribution_major_version == "7")+

- You can **define multiple conditions**, where all of them should be true to execute the tasks

tasks:

- name: "shut down CentOS 6 systems"
- command: /sbin/shutdown -t now
- when:
 - ansible_distribution == "CentOS"
 - ansible_distribution_major_version == "6"

If you need to repeat the same task with different items (loop through), you can use with_items clause

For example, consider you want to add several users

- name: add several users

user:

name: "{{ item }}"

state: present

groups: "developer"

with_items:

- raj

- david

- john

- lauren

Looping Through The Inventory

You can loop through the Inventory file to *list all hosts* or *hosts from a play*, using with_items with the play_hosts or groups variables,

show all the hosts in the inventory

- debug:

msg: "{{ item }}"

with_items:

- "{{ groups['all'] }}"

show all the hosts in the current play

- debug:

msg: "{{ item }}"

with_items:

- "{{ play_hosts }}"

Sample Playbook 3 - Conditions And Loop

This Playbook will add Java Packages to different systems (handling Ubuntu/Debian OS)

- name: debian | ubuntu | add java ppa repo

apt_repository:

repo=ppa:webupd8team/java

state=present

become: yes

when: ansible_distribution == 'Ubuntu'

- name: debian | ensure the webupd8 launchpad apt repository is present

apt_repository:

repo="{{ item }}" http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main"

update_cache=yes

state=present

with_items:

- deb

- deb-src

become: yes

when: ansible_distribution == 'Debian'

You cannot define multiple conditions.

True

False

Question Type: Single-Select

```
vars:
  score: 3
tasks:
  - shell: echo "Bang on!! You won."
    when: ??
```

How do you use a variable to apply condition?

when: score == 3

when: {{ score }} == 3

when: 'score' == 3

when: "score" == 3

ansible.cfg

You told how your hosts should behave (via Playbook). But how do you tell Ansible (your control machine) should behave?

through ansible.cfg

ansible.cfg is a configuration file that defines how Ansible should behave.

It tells

- how to establish an ssh connection
- duration of ssh connection with the host
- how to run the playbook
- where to log errors that might occur while playing playbook on hosts etc.

Ansible Configuration Settings

Your ansible.cfg file will have the following settings:

[defaults]

[privilege_escalation]

[paramiko_connection]

[ssh_connection]

[accelerate]

You will now learn the frequently used settings in upcoming cards. For detailed understanding, you may refer to the Ansible Official Docs.

[defaults]

poll_interval

For **asynchronous tasks in Ansible**, this tells the frequency of checking the status of task completion. The default value is 15 seconds; which implies, for every 15 sec, it will check if the task is completed.

poll_interval = 15

sudo_user

This is the default user to sudo. If --sudo-user is not specified in an Ansible playbook, the default is the most logical: 'root':

sudo_user = root

ask_pass

This tells if **Ansible Playbook should ask for a password** by default. If SSH key is used for authentication, the **default behavior is no**.

ask_pass = True

ask_sudo_pass

Just like ask_pass, this asks for **sudo password** by default while sudoing.

ask_sudo_pass = True

remote_port

If your systems did not define an alternate value in Inventory for SSH port, this sets the default value to 22.

remote_port = 22

[privilege_escalation]

*For some tasks to execute in Ansible, you require administrative access to the system.

The settings under [privilege_escalation] will escalate your privileges.*

become

If set to true or yes, you **activate privilege escalation**.

Default behavior is no.

become=True

become_method

You can define the **method for privilege escalation**.

Default is sudo, other options are su, pbrun, pfexec, doas, ksu.

become_method=sudo

become_user

This allows you to **become the user over privilege escalation**.

Default is root.

become_user=root

become_ask_pass

Asks the **password for privilege escalation**, the *default is False*.

become_ask_pass=False

accelerate_timeout

This setting will **close the socket connection** if no data is received from the client for the defined time duration.

default: accelerate_timeout = 30

record_host_keys

If the setting is True(default value), this will **record new hosts on the user's host file**, provided **host key checking is enabled**. Setting it to False, the performance will improve and which is recommended when **host key checking is disabled**.

record_host_keys=False

pipelining

If enabled, the **number of SSH connections** required for execution of a module on the remote server is **reduced**, improving the performance significantly.

By default this is disabled: pipelining = False.

You can activate your privilege escalations utilizing _____ settings.

become=false

become=true

ansible.cfg must be present in _____.

/etc/ansible

/etc/configurations

/etc/hosts

/ansible/

How to define the number of parallel processes while communicating to remote hosts?

pipelining

become

become_method

forks