1. Installation :
   1. Refer installation guide @ [http://docs.ansible.com/ansible/intro\_installation.html](https://urldefense.proofpoint.com/v2/url?u=http-3A__docs.ansible.com_ansible_intro-5Finstallation.html&d=CwMFAg&c=MOptNlVtIETeDALC_lULrw&r=uk36AHeoGz-xWiQRJobWSj_o_mvZphPAtmuXYd4uRVY&m=WzGsIA4N265Pj8-OKpkCJYR4wbox3H7hKHrIbHg4xyI&s=V4JkNJHJ5dqZUhQhE5EuDyxbmkRUfNaDQZIpvv9E2Bs&e=) for fresh installation on any box .
   2. Ansible can only be run from machine with Python 2.6 or 2.7
   3. ansible-playbook -i inventory/ansible\_hosts --private-key ./keys.pem playbooks/deploy\_app.playbook --limit  grouphosts -e "APP\_VER=1.0.1 deploy\_flag=True TLD\_SCHEMA=test"
2. Get started :
   1. Edit (or create) /etc/ansible/hosts
   2. Your public SSH key should be located in authorized\_keys
      * Copy users public SSH key from ansible master to deployment server’s authorized\_keys .
   3. ansible all --inventory-file=hosts -m ping
   4. ansible all --inventory-file=hosts -a "/bin/echo hello"
   5. ansible all --inventory-file=hosts -a "/bin/echo hello" -u username
   6. remote\_user :
      * This is the default username ansible will connect as for /usr/bin/ansible-playbook. Note that /usr/bin/ansible will always default to the current user if this is not defined
   7. Patterns :
      * 192.168.1.\*
      * webservers:dbservers
      * webservers:!phoenix
   8. Parallelism :
      * $ ansible atlanta -a "/sbin/reboot" -f 10
   9. transfer a file
      * $ ansible webservers -m file -a "dest=/srv/foo/b.txt mode=600 owner=mdehaan group=mdehaan"
   10. Managing Services
       * $ ansible webservers -m service -a "name=httpd state=restarted"
3. Playbooks :
   1. Playbooks are the basis for a really simple configuration management and multi-machine deployment system
   2. Examples :
      * <https://github.com/ansible/ansible-examples>
   3. Each playbook is composed of one or more ‘plays’ in a list.
   4. The goal of a play is to map a group of hosts to some well defined roles, represented by things ansible calls tasks. At a basic level, a task is nothing more than a call to an ansible module
   5. ansible-playbook playbook.yml -f 10
   6. To see what hosts would be affected by a playbook before you run it, you can do this:

ansible-playbook playbook.yml --list-hosts

* + - Example :

---

- hosts: sxxxxx

tasks:

- name: Copy

copy: src=/a/b/c/test.txt dest=/a/b/ansible\_deployment

---

- hosts: webservers

remote\_user: root

tasks:

- name: ensure apache is at the latest version

yum: name=httpd state=latest

- name: write the apache config file

template: src=/srv/httpd.j2 dest=/etc/httpd.conf

- hosts: databases

remote\_user: root

tasks:

- name: ensure postgresql is at the latest version

yum: name=postgresql state=latest

- name: ensure that postgresql is started

service: name=postgresql state=running

1. Tasks :
   1. Each play contains a list of tasks. Tasks are executed in order, one at a time, against all machines matched by the host pattern, before moving on to the next task.
   2. It is important to understand that, within a play, all hosts are going to get the same task directives. It is the purpose of a play to map a selection of hosts to tasks.
   3. The goal of each task is to execute a module, with very specific arguments. Variables, as mentioned above, can be used in arguments to modules.

tasks:

- name: run this command and ignore the result

shell: /usr/bin/somecommand

ignore\_errors: True

* 1. Now in a very basic playbook all the tasks will be listed directly in that play, though it will usually make more sense to break up tasks using the include: directive.

tasks:

- name: create a virtual host file for {{ vhost }}

template: src=somefile.j2 dest=/etc/httpd/conf.d/{{ vhost }}

1. Templates:
   * + template: src=templates/foo.j2 dest=/etc/foo.conf
2. Handlers :
   1. Handlers are lists of tasks, not really any different from regular tasks, that are referenced by a globally unique name. Handlers are what notifiers notify. If nothing notifies a handler, it will not run. Regardless of how many things notify a handler, it will run only once, after all of the tasks complete in a particular play.
   2. ‘notify’ actions are triggered at the end of each block of tasks in a playbook

- name: template configuration file

template: src=template.j2 dest=/etc/foo.conf

notify:

- restart memcached

- restart apache

handlers:

- name: restart memcached

service: name=memcached state=restarted

- name: restart apache

service: name=apache state=restarted

1. Roles :
   1. Roles in Ansible build on the idea of include files and combine them to form clean, reusable abstractions – they allow you to focus more on the big picture and only dive down into the details when needed.

tasks:

- include: wordpress.yml wp\_user=timmy

- include: wordpress.yml wp\_user=alice

- include: wordpress.yml wp\_user=bob

* 1. variables passed in can then be used in the included files. We’ll cover them in [Variables](http://docs.ansible.com/ansible/playbooks_variables.html). You can reference them like this:

{{ wp\_user }}

* 1. Includes can also be used to import one playbook file into another. This allows you to define a top-level playbook that is composed of other playbooks.

For example:

- name: this is a play at the top level of a file

hosts: all

remote\_user: root

tasks:

- name: say hi

tags: foo

shell: echo "hi..."

- include: load\_balancers.yml

- include: webservers.yml

- include: dbservers.yml

* 1. what is the best way to organize your playbooks? The short answer is to use roles! Roles are ways of automatically loading certain vars\_files, tasks, and handlers based on a known file structure. Grouping content by roles also allows easy sharing of roles with other users.
  2. In a playbook, it would look like this:

---

- hosts: webservers

roles:

- common

- webservers

1. Variables :
   1. In a playbook, it’s possible to define variables directly inline like so:

- hosts: webservers

vars:

http\_port: 80

* 1. For instance, in a simple template, you can do something like:

My amp goes to {{ max\_amp\_value }}

* 1. This is also valid directly in playbooks, and you’ll occasionally want to do things like:

template: src=foo.cfg.j2 dest={{ remote\_install\_path }}/foo.cfg

* 1. Register Variables :

Another major use of variables is running a command and using the result of that command to save the result into a variable.

The value of a task being executed in ansible can be saved in a variable and used later.

Example:

- hosts: web\_servers

tasks:

- shell: /usr/bin/foo

register: foo\_result

ignore\_errors: True

- shell: /usr/bin/bar

when: foo\_result.rc == 5

* 1. Variable File Separation

You can do this by using an external variables file, or files, just like this:

---

- hosts: all

remote\_user: root

vars:

favcolor: blue

vars\_files:

- /vars/external\_vars.yml

tasks:

- name: this is just a placeholder

command: /bin/echo foo

This removes the risk of sharing sensitive data with others when sharing your playbook source with them.

The contents of each variables file is a simple YAML dictionary, like this:

---

# in the above example, this would be vars/external\_vars.yml

somevar: somevalue

password: magic

* 1. Passing Variables On The Command Line

Example:

---

- hosts: '{{ hosts }}'

remote\_user: '{{ user }}'

tasks:

- ...

ansible-playbook release.yml --extra-vars "hosts=vipers user=starbuck"

1. Facts :
   1. Facts are information derived from speaking with your remote systems.
   2. An example of this might be the ip address of the remote host, or what the operating system is.

To see what information is available, try the following:

ansible hostname -m setup

1. Conditionals :
   1. Often the result of a play may depend on the value of a variable
   2. the values of variables may depend on other variables
   3. The When Statement

* Sometimes you will want to skip a particular step on a particular host.

tasks:

- name: "shutdown Debian flavored systems"

command: /sbin/shutdown -t now

when: ansible\_os\_family == "Debian"

* Suppose we want to ignore the error of one statement and then decide to do something conditionally based on success or failure:

tasks:

- command: /bin/false

register: result

ignore\_errors: True

- command: /bin/something

when: result|failed

- command: /bin/something\_else

when: result|succeeded

- command: /bin/still/something\_else

when: result|skipped

* 1. Variables defined in the playbooks or inventory can also be used. An example may be the execution of a task based on a variable’s boolean value:

vars:

epic: true

Then a conditional execution might look like:

tasks:

- shell: echo "This certainly is epic!"

when: epic

* 1. with\_items:

tasks:

- command: echo {{ item }}

with\_items: [ 0, 2, 4, 6, 8, 10 ]

when: item > 5

1. Loops :
   1. Standard Loops
      * with\_items:
   2. Nested Loops
   3. Looping over Hashes
   4. Looping over Files
   5. Looping over Fileglobs
   6. Looping over Parallel Sets of Data
   7. Looping over Subelements
   8. Looping over Integer Sequences
   9. Random Choices
   10. Do-Until Loops
   11. Finding First Matched Files
   12. Iterating Over The Results of a Program Execution
   13. Looping Over A List With An Index
   14. Using ini file with a loop
   15. Flattening A List
   16. Using register with a loop
   17. Looping over the inventory
   18. Loops and Includes
   19. Writing Your Own Iterators
2. Tags

* Example:

tasks:

- yum: name={{ item }} state=installed

with\_items:

- httpd

- memcached

tags:

- packages

- template: src=templates/src.j2 dest=/etc/foo.conf

tags:

- configuration

* If you wanted to just run the “configuration” and “packages” part of a very long playbook, you could do this:

ansible-playbook example.yml --tags "configuration,packages"

* On the other hand, if you want to run a playbook without certain tasks, you could do this:

ansible-playbook example.yml --skip-tags "notification"

1. Best Practices :
   1. <http://docs.ansible.com/ansible/playbooks_best_practices.html>
2. References :
   1. <http://docs.ansible.com>
   2. <https://serversforhackers.com/an-ansible-tutorial>
3. shell vs command:

A typical example are the Ansible modules Shell and Command. In the most use cases both modules lead to the same goal. Here are the main differences between these modules.

* 1. With the Command module the command will be executed without being proceeded through a shell. As a consequence some variables like $HOME are not available. And also stream operations like <, >, | and & will not work.
  2. The Shell module runs a command through a shell, by default /bin/sh. This can be changed with the option executable. Piping and redirection are here therefor available.
  3. The command module is more secure, because it will not be affected by the user’s environment.

1. Template module with different set of variables?
   * + <http://stackoverflow.com/questions/31142369/how-to-use-template-module-with-different-set-of-variables>
     + <https://nsrc.org/workshops/2015/rwnog/raw-attachment/wiki/Track2Agenda/variables-templates.htm>