```
# Do not call puppet facter or ohai even if present.
# ansible all -m setup -a 'gather_subset=!facter,!ohai'


# Only collect the minimum amount of facts:
# ansible all -m setup -a 'gather_subset=!all'
 '
# Display facts from Windows hosts with custom facts stored in
C(C:\custom_facts).
# ansible windows -m setup -a "fact_path='c:\custom_facts'"
```

11. Playbooks

Playbooks are Ansible's configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.
If Ansible modules are the tools in your workshop, playbooks are your instruction manuals, and your inventory of hosts are your raw material.

At a basic level, playbooks can be used to manage configurations of and deployments to remote machines. At a more advanced level, they can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers along the way.

Playbooks are a completely different way to use ansible than in adhoc task execution mode, and are particularly powerful.

While you might run the main `/usr/bin/ansible` program for ad-hoc tasks, playbooks are more likely to be kept in source control and used to push out your configuration or assure the configurations of your remote systems are in spec.

12. Playbook Language Example

Playbooks are expressed in YAML format and have a minimum of syntax, which intentionally tries to not be a programming language or script, but rather a model of a configuration or a process.

Each playbook is composed of one or more 'plays' in a list.

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.

Playbook for starters

```
$ cat web_db.yml
```

```
---
- hosts: webservers
  become: yes
  tasks:
  - name: Ensure Apache installed
    yum: name=httpd state=present

  - name: Creates directoy
    file: path=/var/www/html/ansible state=directory

  - name: Ensure Apache is running
    service: name=httpd enabled=yes state=started

- hosts: dbservers
  become: yes

  tasks:
  - name: Ensure mysql server installed
    yum: name=mysql-server state=present

  - name: Ensure mysql running
    service: name=mysqld state=started
```

13. YAML Basics

For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a “hash” or a “dictionary”. So, we need to know how to write lists and dictionaries in YAML.

There’s another small quirk to YAML. All YAML files (regardless of their association with Ansible or not) can optionally begin with `---` and end with `...`. This is part of the YAML format and indicates the start and end of a document.

All members of a list are lines beginning at the same indentation level starting with a `"- "` (a dash and a space):

```
---
# A list of tasty fruits
fruits:
    - Apple
    - Orange
```

```
    - Strawberry
    - Mango
...
```

A dictionary is represented in a simple `key: value` form (the colon must be followed by a space):

```
# An employee record
martin:
    name: Martin D'vloper
    job: Developer
    skill: Elite
```

More complicated data structures are possible, such as lists of dictionaries, dictionaries whose values are lists or a mix of both:

```
# Employee records
- martin:
    name: Martin D'vloper
    job: Developer
    skills:
      - python
      - perl
      - pascal
- tabitha:
    name: Tabitha Bitumen
    job: Developer
    skills:
      - lisp
      - fortran
      - erlang
```

Dictionaries and lists can also be represented in an abbreviated form if you really want to:

```
---
martin: {name: Martin D'vloper, job: Developer, skill: Elite}
fruits: ['Apple', 'Orange', 'Strawberry', 'Mango']
```

14. First Playbook exercise.

We need two centos 6 servers for this exercise.
We will deploy a apache webserver to web1 with a sample website and MySQL db to db1 node.

Inventory file

```
$ cat inventory-dev

web1 ansible_ssh_host=192.168.1.13
```

```
db1 ansible_ssh_host=192.168.1.14

[webservers]
web1

[dbservers]
db1

[datacenter:children]
webservers
dbservers

[datacenter:vars]
ansible_ssh_user=vagrant
ansible_ssh_pass='vagrant'
```

Ansible configuration file

```
$ cat ansible.cfg
[defaults]
hostfile=inventory-dev
host_key_checking=False
```

Sample webpage

```
$ cat index.html
<html>
<head>
<title> favourites / bookmark title goes here </title>
</head>
<body bgcolor="white" text="blue">

<h1> My first page </h1>

This is my first web page and I can say anything I want in here - I do that by
putting text or images in the body section - where I'm typing right now :)
</body>
</html>
```

Playbook

```
$ cat web_db.yaml

---

- hosts: webservers
 'become: yes
  tasks:
  - name: Ensure Apache installed
    yum: name=httpd state=present


  - name: Creates directory
    file: path=/var/www/html/ansible state=directory


  - name: Deploy webpage to path=/var/www/html/ansible
    copy: src=index.html dest=/var/www/html/ansible/ mode=0644


  - name: Ensure Apache is running
    service: name=httpd enabled=yes state=started


  - name: Flush all temporary rules
    service: name=iptables state=restarted


  - name: Allow port 80/http access from anywhere
    iptables:
     action: insert
     chain: INPUT
     protocol: tcp
     destination_port: 80
     state: present
     source: 0.0.0.0/0
     jump: ACCEPT



- hosts: dbservers
  become: yes
```

```
  tasks:
  - name: Ensure mysql server installed
    yum: name=mysql-server state=present

  - name: Ensure mysql running
    service: name=mysqld state=started

  - name: Ensure MySQL-python is installed
    yum: name=MySQL-python state=present

  - name: Create Database
    mysql_db: name=devops state=present

  - name: Create user named mint
    mysql_user: name=mint  password=12345 priv='*.*:ALL' state=present
```

Explanation

- **---** is not mandatory but specifies a start of YAML file
- **- hosts: webservers** represent the host/group name where tasks will get executed. It is the start of a play.
- **become: yes** tell ansible to execute all the tasks with sudo privileges older version of ansible has sudo:yes
- **tasks:** specifies the list of task or modules that will be executed against webserver group.
- **- name:** is not a mandatory option but will always help us read the output of task when executed. If -name option is not specified then the module name should start with a " - " for example "– yum:"
- **yum:** is the name of the module that will get executed along with the arguments "name=httpd state=present" . If -name option is not provided then yum will begin with a "– " that represents the element in the YAML list as specified below.

```
tasks:
- yum: name=httpd state=present
```

- **Ansible documentation gives very nice description of every module with examples.**

  https://docs.ansible.com/ansible/list_of_all_modules.html

https://docs.ansible.com/ansible/yum_module.html

https://docs.ansible.com/ansible/file_module.html

https://docs.ansible.com/ansible/copy_module.html

https://docs.ansible.com/ansible/service_module.html

https://docs.ansible.com/ansible/iptables_module.html

- **New and Old argument style**

  Old Style

```
tasks:
- yum: name=httpd state=present
```

  New Style

```
tasks:
 - yum:
    name: httpd
    state: present
```

- **iptables** module modifies the systems firewall rule, knowledge of iptables is required to understand its options. We are allowing port 80/http access from everywhere.
- **- hosts:dbservers** is the start of a next play that will get executed on dbservers group. As mentioned earlier playbook is the list of Plays, os -hosts: dbservers is just the second play in the playbook. Likewise we can have multiple plays in the same playbook.
- **mysql_db** module is used to create/delete databases in mysql db service.

  https://docs.ansible.com/ansible/mysql_db_module.html
- **mysql_user** module is used to create user in mysql db service.

  https://docs.ansible.com/ansible/mysql_user_module.html

  NOTE: Its highly recommended to read ansible module documentation to understand more about modules used in the playbook.

15. Playbook Execution

ansible-playbook command is used to execute the playbook as shown below.

```
$ ansible-playbook web_db.yaml


PLAY [webservers]
*****************************************************************
```

```
TASK [Gathering Facts]
*********************************************************

ok: [web1]


TASK [Ensure Apache installed]
*****************************************************

changed: [web1]


TASK [Creates directory]
*********************************************************

changed: [web1]


TASK [Deploy webpage to path=/var/www/html/ansible]
*****************************

changed: [web1]


TASK [Ensure Apache is running]
************************************************

changed: [web1]


TASK [Flush all temporary rules]
*********************************************

changed: [web1]


TASK [Allow port 80/http access from anywhere]
*********************************

changed: [web1]


PLAY [dbservers]
*****************************************************************


TASK [Gathering Facts]
*********************************************************

ok: [db1]


TASK [Ensure mysql server installed]
******************************************

changed: [db1]
```

```
TASK [Ensure mysql running]
*****************************************************

changed: [db1]


TASK [Ensure MySQL-python is installed]
*****************************************

changed: [db1]


TASK [Create Database]
***********************************************************

changed: [db1]


TASK [Create user named mint]
************************************************

changed: [db1]


PLAY RECAP
*********************************************************************

db1                        : ok=6    changed=5    unreachable=0    failed=0

web1                       : ok=7    changed=6    unreachable=0    failed=0
```

Explanation

- Playbook gets executed in top to down order. First play is PLAY [webservers].
- TASK [Gathering Facts] is a default task that runs the setup module for every host in the play. We can disable gathering facts by specifying **gather_facts: False** in playbook as shown below.

```
---
- hosts: webservers
  become: yes
  gather_facts: False
  tasks:
```

- If the -name option is used in the task then it will display the content from it.
- **changed: [web1]** is a status message for the task, it means that the task made changes to the target host.
- **ok: [web1]** means that the task has not made any changes on the target host. It could be due to the nature of the module which could be for information gathering like setup module. It

could also mean that the system is in the same state for example httpd is already installed so it will return ok:

- **PLAY RECAP** Display the summary of all the tasks got executed on all the hosts. **unreachable** display the number of hosts that does not have proper connectivity with ansible server. **failed** displays number of failed tasks.

16. Variables

Variables Defined in a Playbook

Variables can be defined with some custom value in playbook as shown below. These variables can be used in the playbook by enclosing variable name in **{{**varname**}}**.

```
$ cat db.yml
- hosts: dbservers
  become: yes
  vars:
   dbname: devops
   dbuser: mint
   dbpass: 12345

  tasks:
  - name: Ensure mysql server installed
    yum: name=mysql-server state=present

  - name: Ensure mysql running
    service: name=mysqld state=started

  - name: Ensure MySQL-python is installed
    yum: name=MySQL-python state=present

  - name: Create Database
    mysql_db: name={{dbname}} state=present

  - name: Create user named mint
    mysql_user: name={{dbuser}}  password={{dbpass}} priv='*.*:ALL'
state=present
```

Variables in group_vars & host_vars

These variables are inventory specific and can only be accessed by host & groups from the inventory located in current directory. You can have multiple inventory in separate directory structure as shown below. Every inventory may have its own group_vars & host_vars directory where we store variables.

directory layout

```
production/
├── group_vars
│   └── all
└── inventory
staging/
├── group_vars
│   └── all
└── inventory
```

Variables can be defined into a directory structure. group_vars holds variables that can be used by all the groups. host_vars holds variable specific to the hostname.

- group_vars/all will contain variables that can be used by all the hosts from the inventory file.
- group_vars/dbservers variables will be only accessible for the dbservers group and not any other host or group from inventory.
- host_vars/web1 variables will be only accessible for the web1 host and not any other host from the inventory file.

```
$ mkdir group_vars
$ vi group_vars/all
# Common Variables for all hosts in the inventory
user: tesla
group: electric
pass: 12345
```

```
$ vi group_vars/dbservers
# Variables exposed for the group named dbservers from the inventory file.
dbname: devops
dbuser: mint
dbpass: 12345
```

```
$ mkdir host_vars

$ vi host_vars/web1

# Variables exposed for the host named web1 from the inventory file.

user: edison

group: electric

pass: 12345
```

If we try to access user variable for db1 its value will come from group_vars/all file.

```
$ ansible -m user -a "name={{user}} password={{pass}}" --sudo db1

db1 | SUCCESS => {

    "changed": true,

    "comment": "",

    "createhome": true,

    "group": 1002,

    "home": "/home/tesla",

    "name": "tesla",

    "password": "NOT_LOGGING_PASSWORD",

    "shell": "",

    "state": "present",

    "system": false,

    "uid": 1002

}
```

For web1 host we created variables in host_vars/web1. host_vars variable will high higher precedence, so it will ignore user variable from group_vars/all file and pickup value from host_vars/web1 file.

```
$ ansible -m user -a "name={{user}} password={{pass}}" --sudo web1

web1 | SUCCESS => {

    "changed": true,

    "comment": "",

    "createhome": true,

    "group": 1002,

    "home": "/home/edison",

    "name": "edison",

    "password": "NOT_LOGGING_PASSWORD",
```

```
    "shell": "",
    "state": "present",
    "system": false,
    "uid": 1002
}
```

17. Including Playbooks

In site.yml, we call other playbooks. Note this is SUPER short, because it's just including some other playbooks. Remember, playbooks are nothing more than lists of plays:

```
$ cat site.yml
---
# file: site.yml
- include: webservers.yml
- include: dbservers.yml
```

18. Store Output of a command.

Register module is used to store output of any module/command and store it into a variable.

```
---
- hosts: webservers
  become: yes
  tasks:
    - shell: /usr/bin/whoami
      register: username

    - file: path=/tmp/info.txt owner={{username}}
```

shell module is used to run Linux shell commands

register: username will store the output of shell command in username variable.

File module here is assigning ownership to file /tmp/info.txt

19. Debug module

Debug module is used to print messages or variable values while playbook execution. It helps finding the problem if the variable values are not properly assigned or accessed.

```
$ cat deb.yml

---
-' hosts: all
  vars:
   http_port: 8087
   username: cassini
  tasks:


    - debug: msg="Inventory hostnames are {{inventory_hostname}}"
    - debug: msg="Port variable is {{http_port}} & username is {{username}}"
```

```
$ ansible-playbook deb.yml


PLAY [all]
*********************************************************************


TASK [Gathering Facts]
*********************************************************
ok: [web1]
ok: [db1]


TASK [debug]
*********************************************************************
ok: [web1] => {
    "changed": false,
    "msg": "Inventory hostnames are web1"
}
ok: [db1] => {
    "changed": false,
    "msg": "Inventory hostnames are db1"
}
```

```
TASK [debug]
*********************************************************************
ok: [web1] => {
    "changed": false,
    "msg": "Port variable is 8087 & username is cassini"
}
ok: [db1] => {
    "changed": false,
    "msg": "Port variable is 8087 & username is cassini"
}
```

20. Prompting for Input

Take user input while executing playbook with vars_prompt and store into a variable as shown below.

```
$ cat prompt.yml
---
- hosts: dbservers
  vars:
   http_port: 8087
   username: cassini
  vars_prompt:
   - name: "dbpass"
     prompt: "Enter password for database."

  tasks:

- debug: msg="DB password is {{dbpass}}"
```

```
$ ansible-playbook prompt.yml
Enter password for database.:

PLAY [dbservers]
*********************************************************************
```

```
TASK [Gathering Facts]
*********************************************************

ok: [db1]


TASK [debug]
*********************************************************************
ok: [db1] => {
    "changed": false,
    "msg": "DB password is deltaq123"
}
```

21. Handlers

Handlers are special kind of tasks. In first glimpse, it will look exactly like any other task but the difference is in the execution. Tasks as we have seen so far gets executed as we run our playbook but handlers being in same playbook will only get executed when it gets notified. Notification would be sent from the task if the state of the task is changed: true.

For example, if we copy a file using copy module it gets copied if the destination file is different or not present. In this case the state of the task is changed: true but if the destination file is same as source then the file does not get overwritten and state would be changed: false.

So, if we are notifying a handler from such task the handler will get notified only if the file is copied or else it will not send a notification.

```
$ cat handler.yml
---
- hosts: webservers
  become: yes
  tasks:
    - name: Copy the website config file
      copy: src=httpd.conf dest=/etc/httpd/conf/httpd.conf
      notify:
        - Apache Restart


  handlers:
    - name: Apache Restart
      service: name=httpd state=restarted
```

In the above code the handler named “Apache Restart” will only get executed when the httpd.conf file gets copied or else it will not notify the handler.

22. Conditional Execution

Sometimes you want to run a particular task from the playbook only when it meets a certain criteria.

It would be similar to the if else condition which we have in programming language.

In ansible we have “**when”** module to check the condition, if it returns true the task gets executed or else skipped.

For example, you are writing a playbook that should run on Ubuntu and Centos systems. As we know if we want to install a package in ubuntu os, we use apt module, for centos we use yum module.

So in this case our task should check a condition of OS family and execute apt if its Ubuntu and yum when its Centos.

We can use the fact variable **ansible_os_family** which stores the value of OS type to check the condition.

```
$ ansible -m setup web1 | grep ansible_os_family
        "ansible_os_family": "Debian",
```

```
$ cat condition.yml
---
- hosts: webservers
  become: yes
  tasks:
    - name: Install Apache on Centos
      yum: name=httpd state=present
      when: ansible_os_family == "RedHat"

    - name: Install Apache on Ubuntu
      yum: name=apache2 state=present
      when: ansible_os_family == "Debian"
```

23. Templates

Templates are similar to copy module, it copies the source file to the target hosts destination.

But here we don't have plain static files, we have template file which contains pre-defined variables.

These variables could be defined in playbook or host_vars or group_vars etc.

While the template module gets executed it will read the template file and change all the variables to its value and copy the file to the target host.

Template file ends with .j2 extension which stands for Jinja2 templates.

```
$ mkdir templates
$ vi templates/index.j2
<html>
<head><title>Hello from Ansile</title></head>
<body>
<h1>Congratulations!</h1>
<p>Nice job {{ username }}!! </p>
</body>
</html>
```

We use template module to copy the template file and the variable used in template file can be defined in the playbook as shown below.

```
$ vi web.yaml
---
- hosts: webservers
  sudo: yes
  vars:
   username: GreenApple
   doc_root: /var/www/html/
  tasks:

  - name: Copy Site Files
    template: src=templates/index.j2 dest={{doc_root}}/index.html mode=0644
```

24. A sample playbook with variables, templates, conditions and handlers.

We need a centos 6 vm node to run this code.

```
$ ls
ansible.cfg  inventory_prod  templates  Vagrantfile  web.yaml
```

```
$ cat ansible.cfg
[defaults]
hostfile = inventory_prod
host_key_checking = False
```

```
$ cat inventory_prod


web1 ansible_ssh_host=192.168.1.11
db1 ansible_ssh_host=192.168.1.8



[webservers]
web1

[dbservers]
db1

[datacenter:children]
webservers
dbservers

[datacenter:vars]
ansible_ssh_user=vagrant
ansible_ssh_pass=vagrant
```

```
$ ls templates/
httpd.j2  index.j2
```

Template for index.html file

```
$ cat templates/index.j2

<html>

<head><title>Hello from Ansible</title></head>

<body>

<h1>Congratulations!</h1>

<p>Nice job {{ username }}!! You have successfully ran your smart playbook!
Yuppyyy! Now go and create some Roles out of this playbook

</p>

</body>

</html>
```

Templates variable details from httpd.j2 template.

You can get httpd.conf file after installing httpd on centos, its location is /etc/httpd/conf/httpd.conf.

Copy the content of httpd.conf file into templates/httpd.j2 and replace its values as shown below.

Below mentioned variables are defined in the playbook.

```
$ cat templates/httpd.j2 | grep '{{'

MaxClients       {{ max_clients }}

Listen {{ http_port }}

Alias {{ doc_dir }} {{ doc_root }}
```

The Playbook

```
$ cat web.yaml

---

- hosts: webservers

  sudo: yes

  vars:

    http_port: 80

    doc_dir: /ansible/

    doc_root: /var/www/html/ansible/

    max_clients: 5

    ansible_python_interpreter: python
```