**SVS Ansible Automation Handbook**

**Authors: Muralidharan Vasudevan**

**Shreeshrita Patnaik**

Introduction 2

**Why Use Ansible** 2

**Ansible Installation** 2

**Change Management** 2

**Provisioning** 3

**Automation** 3

**Orchestration** 3

**YAML** 4

Architecture and Process Flow 4

**System Requirements** 4

**Process Components** 4

**Execution Flow** 6

Ansible Inventory & Configuration 7

**Inventory Fundamentals** 7

**Configuration File Basics** 9

Modules 10

**Ansible Module Fundamentals** 10

**A Simple Demo of a Module** 11

**Host Group Patterns** 13

Plays & Playbooks 14

**Playbooks Basics** 14

**Play Breakdown** 15

**Tasks** 15

**Execution of Playbooks** 15

**Tasks Execution Failure** 16

**Sample Configuration File & Playbook** 16

**Playbook Advanced Feature** 17

Roles 20

**Role Basics:** 20

**Demonstration of a Role** 21

**Site.yml** 24

Reference Link: 25

# **Introduction**

Ansible is a simple automation language that can perfectly describe an IT application infrastructure. It’s easy-to-learn, self-documenting, and doesn’t require a grad-level computer science degree to read. Automation shouldn’t be more complex than the tasks it’s replacing. Ansible is easy to extend and implement.

Ansible is comprised of four different functions that do not work standalone but are tightly coupled with each other.

1. Change Management
2. Provisioning
3. Automation
4. Orchestration

## **Why Use Ansible**

* It is clean
* No agents required to be installed on remote system
* No database required
* No residual software after task completion
* No complex upgrades

## **Ansible Installation**

<http://docs.ansible.com/ansible/latest/intro_installation.html>

## **Change Management**

Change Management starts with defining the System State.

For e.g. A System State defined could be:

1. Apache web server Installed
2. Apache web server installed with version x.x.x
3. Apache web server up and running

Any deviations to the above defined System State will initiate a Change Event to put that system back to the state which it supposed to be. That is called a Change. A change event is seen as an expected or unexpected depends on the environment the change is happening. For e.g. in Production Environments Change event is not an expected function which reflects the instability of the production environment.

## **Provisioning**

Provisioning means preparing a system to make it ready. For e.g. To Make a Web Server. It involves installing and configuration of software that transitions one state of a server to another.

For e.g. Provisioning of a web server involves the following steps.

1. Install the Web server
2. Copy the Configuration files to the server
3. Copy the web files to the server
4. Install Security updates in the server
5. Start Web service in the server

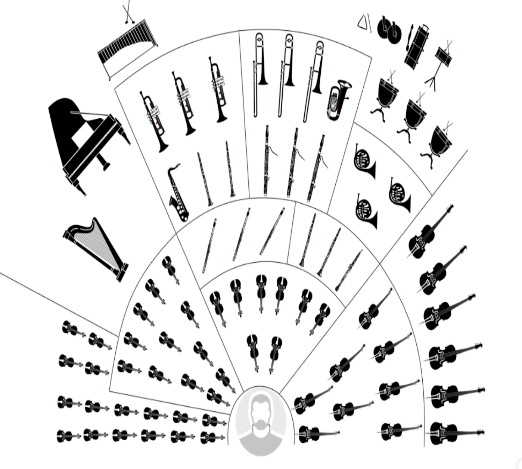
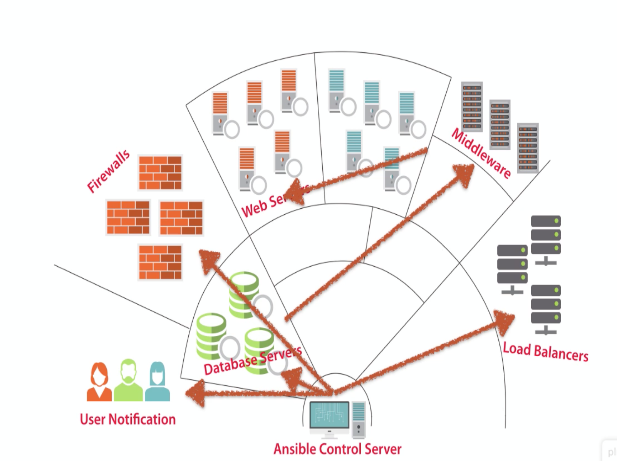
## **Automation**

Automation is the lower layer of the Ansible components. The Automation function of Ansible defines the tasks to be run automatically.

For e.g. A recent security threat expects you to update a patch in 100 systems in your lab and the expected time to perform this action is very less. Ansible in this case with the concept of Playbook helps you to define tasks with logical controls that makes decisions and run them automatically by enabling productivity.

## **Orchestration**

Orchestration takes Automation and coordination between multiple Systems. Each Server may have some tasks to run, but certain tasks may be dependent on a task in a different Server to run and complete first. Orchestration helps these interdependent tasks to coordinate themselves bidirectionally.

## **YAML**

YAML is a structured format language used by Ansible to run the tasks. It is very simple as it is a not a programming language and represented in text format which is easy to read and write.

The Playbooks which define the tasks in Ansible uses YAML format.

# **Architecture and Process Flow**

## **System Requirements**

Two major components involved here are

Ansible Control Server

Remote Server

**Ansible Control Server**

This is the server where Ansible will be installed and all the instructions are initiated and tasks are executed. Control server has OS limitations. Ansible control server is supported on only \*NIX platforms. i.e. Unix/Linux/Mac (Since Linux based).

Note: Ansible is not supported for Windows platform from a Control server perspective.

**Remote Server**

This is the server that is managed by Ansible Control Server. Ansible extends supports for various platforms that includes Servers, Switches, Routers, Firewalls and so on from a management perspective.

However, from a Server perspective Ansible requires it to be a \*NIX platform with Python 2.6+ (but not Python 3.x) and SSH installed.

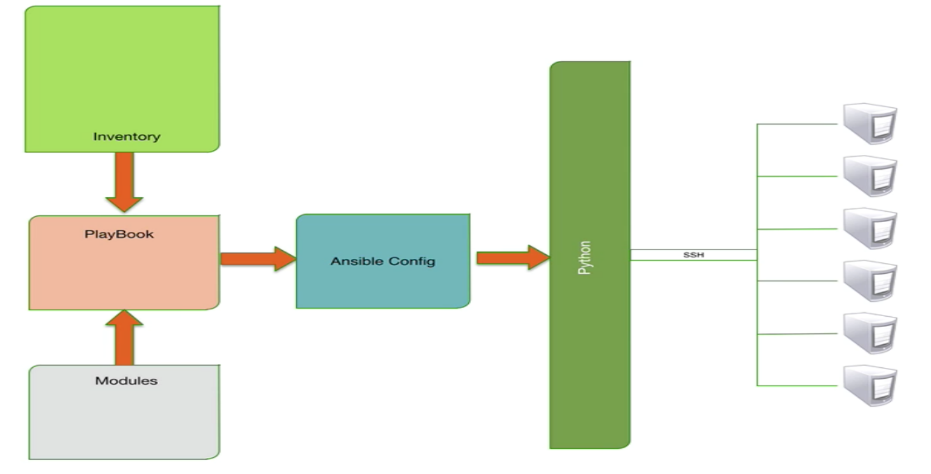
Note: Ansible has recently released support for Windows platform with Remote PowerShell enabled to be Managed as well.

## **Process Components**

There are 5 basic components of the Ansible processing.

1. Inventory
2. Modules
3. Playbooks
4. Configuration Files
5. Python

Process Map.

****

**Inventory**

Inventory is generally called a hosts file where we define all the server or system details to be managed like host name, ip address and authentication credentials. It’s a plain text file.

**Modules**

Modules are the programmable unit of work to be done. It’s a command center.

*For e.g. in Linux you have “yum” as a module to install software.*

There are core modules which is supported by Ansible and you can fine more such modules in Ansible Galaxy which is a community based catalogue.

**Playbooks**

Playbook is set of plays defined in a file that is used to accomplish a task or set of tasks. It uses YAML format. You can add logical controls in playbook to define the order of execution of the tasks.

E.g. Checking a drive size before copying a file

Play – A single or set of tasks executed on a defined set of hosts

Playbook - A set of plays built in specific sequence to execute on different set of hosts

**Configuration Files**

This is a text file where we can set the global configuration variables and can also change the default values of the variable.

For e.g. You can set how many parallel operations your system can handle or setup a notification if a host is skipped while execution.

**Python**

Python is used by Ansible to compile all the components discussed above and deliver as a package to the remote server. Python framework also helps in building Variables.

Three types of Variables ae used in Ansible

1. **Host variable** - These are variables defined in Inventory used per host
2. **Fact Variables** - Variables for data gathered from System like CPU speed, disk space, Memory etc.
3. **Dynamic Variables** - These are variables created during the course of playbook and will be destroyed later. The result of a playbook can be stored in a variable and used for another playbook as well.

## **Execution Flow**

**Remote Execution**

* + Ansible in Control Server first evaluates the playbook and inventory file to get the host details and the modules to be picked up for execution
  + All the details are compiled in a Python Package in Control Server
  + The Python package is transported (Copied) over SSH connection to the /temp directory in the remote hosts identified from Inventory file.
  + The Python framework in the remote server executes the package and return the results of the execution in a simple JSON format.
  + Once the task is done, the package copied in the remote server in /temp directory is completely deleted.
  + The SSH communication from Control Server is terminated to the Remote server.

**Local Execution**

Packages are executed in local Ansible Server because the Remote server may not be able to receive the Python Package.

For e.g. in Case of a Web Service request/ API calls

Here the Control server will execute the Python package locally and get a URL to send it to the Remote server as a HTTP request. The Remote server process the HTTP request and returns the HTTP response.

# **Ansible Inventory & Configuration**

## **Inventory Fundamentals**

**Features of Inventory**

1. Behavior Parameters - Configuration that applies to a System or group of Systems.

e.g. SSH Username/Password, Private Key files or Path to Python and so on.

1. Groups – Grouping of Systems together.

e.g. Test Database Servers or Production Web Servers

3) Groups of Groups – Parent Groups of multiple Sub Groups

e.g. Datacenter West, Datacenter East

4) Assign Variables – Create Variables and attach to a group

e.g. variables for NTP Server or Syslog Server for a particular group

5) Static/Dynamic - The Inventory file can be Static or Dynamic.

e.g. If Dynamic, Inventory file can be made as an executable Script and Ansible will execute it.

6) Scaling out using Multiple files.

***Sample Inventory File explained***

[db] **-> Group Name**

db1.company.com **ansible\_ssh\_user=aaron ansible\_ssh\_pass=123** **-> Single Host specific Variable**

db2.company.com ansible\_python\_interpreter=/usr/bin/python

[datacenter-west:children] **-> Super Group with Children**

db **-> Subgroup**

[datacenter-west:vars] **-> Super Group Specific Variables. Applies to all the Host in Super Group**

**ansible\_ssh\_user=ansible\_user**

**ansible\_ssh\_pass=#45e!@Gh**

**ntp-server=5.6.7.8**

Notes:

**ansible\_ssh\_user & ansible\_ssh\_pass** are the default Ansible variable names Ansible uses when SSH is invoked.

When giving the host name as not a fully qualified name in the Inventory file , Ansible will always try to resolve DNS name by attaching the hostname of your system. So if given a host name as “db1” in the inventory file & “company.com” is your system hostname then Ansible will try to resolve the name as “db1.companyname.com”.

**Scaling of Inventory File**

In case of a large Environment, there comes a breaking point at which maintain a huge Inventory file would be tedious. There could be multiple users using the Inventory file and editing them in parallel may lead to loss of data or overwrites. Even there might be some expectation to have inventory files and variables based on geographical locations or type of environment.

Below is the directory Structure we can use to define the variables for our Inventory file in multiple Variable files that helps us in scaling the Inventory file.

├── group\_vars

│ ├── all **-> Variables defined in the All file applies to all the Groups**

│ └── db **-> Variables defined for a specific Group called “db” in the Inventory file**

├── host\_vars

│ └── web1 -**> Variable file for host web1 defined in the Inventory File**

├── inventory prod -**> Inventory File**

***Note: A Variable file uses YAML format and uses Key/Value pair to define variables***

**Sample Variable File & Call through Ansible**

**e.g. ALL File**

**---** -> YAML file always should start with “---“

**user: “all\_username”**

**pass: “all\_pass”**

**Now let’s look on how Ansible Calls the Variable used in ALL file**

***ansible webservers -i inventory\_prod –m user –a “name={{user}} password={{pass}}” –sudo***

where,

{{user}} & {{pass}} are the call to the variables used in the ALL file

webservers – Group name

inventory\_prod – Inventory File

Ansible module - user

--sudo - This is required to perform admin operation

***Note: If the same user name is defined in three files ALL, Group\_vars & Host\_vars with different values, then the below is the order of precedence.***

***Group\_vars will take more precedence than ALL***

***Host\_Vars will take more precedence than Group\_vars and All***

## **Configuration File Basics**

Configuration file as explained above is where you define the order of Operations and override default value settings.

To View the list of various Configuration settings in Ansible, visit the below url.

<http://docs.ansible.com/ansible/intro_configuration.html>

**Configuration setting Order of Operation**

Ansible will look for the Configuration file in the below order of precedence

1. **$ANSIBLE\_CONFIG**

Checks the value defined in the Environment Variable at first place

1. **./ansible.cfg**

Looks for the file ansible.cfg in the current execution directory. This can be transported along with the playbook

1. **~/.ansible.cfg**

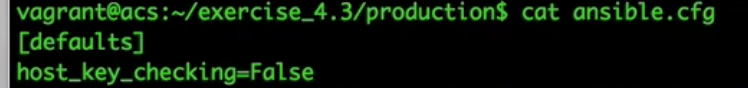
Looks at the hidden file .ansible.cfg Home directory of Logged in user account

1. **/etc/ansibe/ansible.cfg**

Global Configuration file. Looked at last resort. This file will be created only if Ansible installed with PIP command or using packet manager.

***Note: Ansible configuration files are not merged. The First one wins.***

Sample ansible.cfg file in present working directory

******

**Overriding Configuration Variables**

Specific Configuration settings can be overridden using Environmental Variables in two ways

1. **$ANSIBLE\_<configsetting>**

Overrides specific setting by prefixing ANSIBLE\_ to the name

1. **export ANSIBLE\_FORKS=10**

Using Export command to override a specific setting on the fly.

Here the default value of ANSIBLE\_FORK is 5 and we are overriding it to 10.

# **Modules**

## **Ansible Module Fundamentals**

Ansible makes use of Modules to perform different actions as required by the user on the remote system. For e.g. the yum module can be used to install packages on remote systems with RedHat based Linux. We can use parameters along with the modules to customize behavior of that task. Almost all modules that are needed exist in the Ansible core or in the Ansible galaxy.

Modules can be used in plays, playbooks and standalone adhoc commands.

There are 3 types of modules:

1. Core Modules: These are the modules supported by ansible and present in the ansible install packages

2. Extras: These are modules created by the Ansible community, but are included in the distribution, though they may or may not be supported by Ansible.

3. Deprecated: These are modules are not preferred and have been replaced by a new module

There are a total of 462 modules and documentation on these modules can be viewed in the Ansible Server locally.

We can view the modules Ansible supports by using the following commands:

1. To view all documents on the modules available: ansible-doc -l
2. To know more about a specific module: ansible-doc <name\_of\_module> 🡪 e.g. ansible-doc yum
3. To know about how to use a module in playbook with example: ansible-doc -s <name\_of\_module>

There are 15 broad categories of modules. Some common categories are:

1. Manage servers and deploy configs

2. Configure network equipment (Cisco, Juniper)

3. Maintain virtual servers

4. Manage database and tables

5. Deploy load-balancer configs

Some important and useful modules in Ansible:

1. Copy module: to copy files from local ansible box to remote system.

2. Fetch Module: Pulls a file from remote host to local system. It can use md5 checksums to validate the integrity of the file

3. Apt module: to manage installed applications on Debian systems. Can install, update or delete packages.

4. Yum module: to manage installed applications on RedHat systems. Can install, update or delete packages.

5. Service module: can start, stop or restart services. Can be used to enable services on boot.

## **A Simple Demo of a Module**

This is a simple example of using yum and service modules to install and start Apache webserver on a remote RedHat system.

1. To view information on the yum module and how to use it.

ansible-doc yum

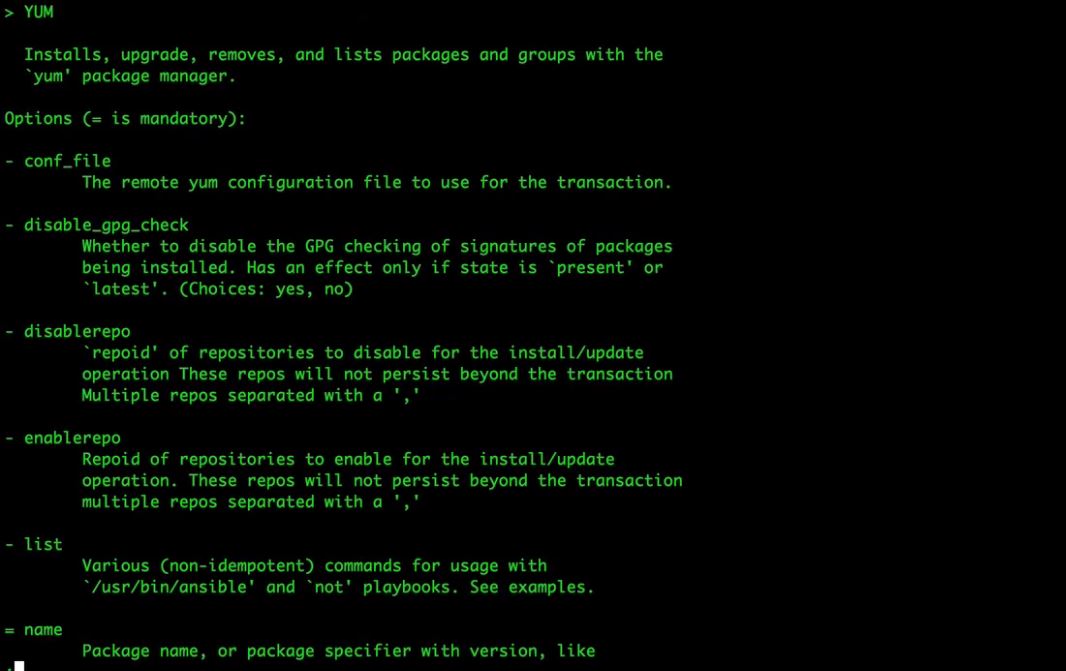


Figure *ansible\_doc yum:* The above figure is the output of the command, **ansible\_doc yum**.

Note: the parameters preceded by a *‘=’* are compulsory, during usage of the particular module

2. To use yum module to install Apache webserver:

**ansible webservers -i inventory -m yum -a "name=httpd state=present" --sudo**

webservers: host group containing all hosts where we want to install apache webserver.

inventory: inventory file containing addresses and credentials of the hosts

-m: indicates that succeeding parameter is module name

yum: module to install packages on RedHat system

-a: indicates that succeeding parameters are options to be used along with the command or module.

name: compulsory parameter which states the name of the package to be installed/removed/updated with yum. (in this case: installed)

state: can be present, absent or latest. In this case state = present indicates that the application httpd must be present and if not present, it must be installed. If the application was already present, in the results summary the field changed if 'False'. If not present, and the package was installed the changed becomes 'True'

sudo: to give root privileges

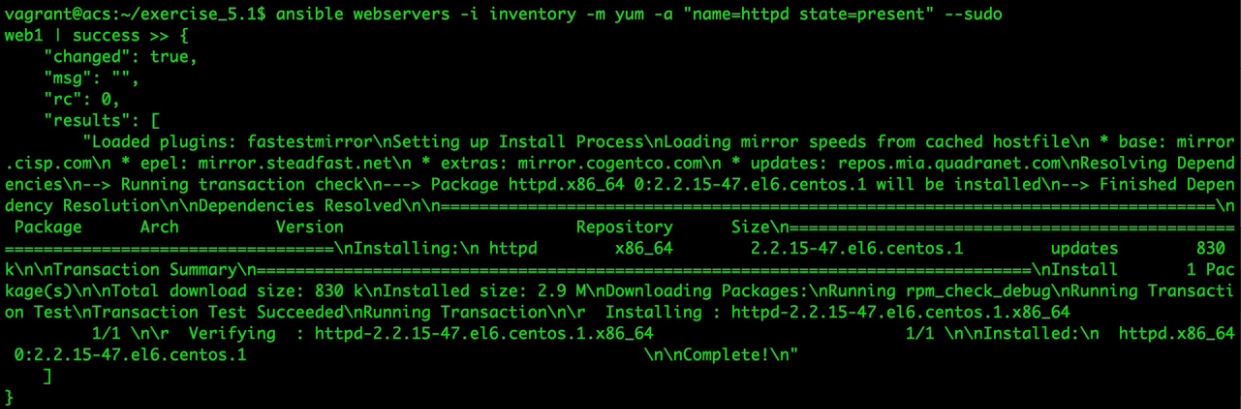


Figure *ansible with yum module:* the above picture demonstrates usage of yum module to install the apache web server application on a remote system

Note: The parameter *changed* indicates if the above action/command caused any change in the state of the system. Since Apache was not previously present, and got installed due to the command, the value of *changed* is *True.*

3. To start the webserver:

**ansible webservers -i inventory -m service -a "name=httpd enabled=yes state=started" --sudo**

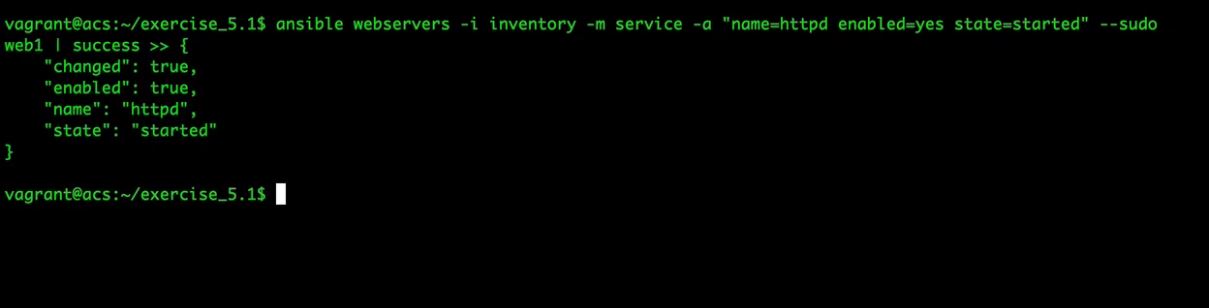


Figure *ansible with service module:* The above figure shows usage of service module to start the Apache web server.

service: to start, stop or restart services. Can be used to enable services on boot.

state: can be started, restarted, stopped. In the above case, the state=started, implies that the service module is used to start the httpd services, unless already started.

enabled=yes: Whether the service should start on boot. At least one of *state* and *enabled* are required.

## **Host Group Patterns**

We can use a combination of a group of hosts while using the ansible command. The combinations that can be used are:

1. OR (webservers:dbservers): The ansible command would apply to all hosts in both groups webservers and dbservers

2. NOT (!dbservers) : the ansible command would apply to all hosts in inventory file not in group dbservers

3. wildcard (web\*er.com): the ansible command would apply to all hosts beginning with web and wending with er.com.

4. Regex (~web[0-9]+) : the ansible command would apply to all hosts beginning with web and followed by one or more numbers. The tilde signifies that the expression is a Regex.

5. AND (dbservers:&webservers) : The ansible command would apply to only those hosts common to both webservers group and dbservers group

# **Plays & Playbooks**

## **Playbooks Basics**

Playbooks are the most common ways to execute your Automation Workflows. Playbooks are powerful and extensible that orchestrate your plays with Error handing and Logic controls.

* Plays maps the Hosts to Tasks
* Each Play can contain multiple tasks
* Each Playbook can contain multiple Plays

***Note: A Playbook uses YAML format and “White Spaces” and Indentations are very crucial as part of the Synta***x. ***Each Plays in a Playbook are separated by a blank line.***

A Sample Playbook with Two Plays:

---

- hosts: webservers

remote\_user: root

tasks:

- name: Install Apache -> Play 1

yum: name=httpd state=present

- name: Start Apache

service: name=httpd state=started

- hosts: dbservers

remote\_user: root

tasks:

- name: Install MySQL -> Play 2

yum: name=mysql-server state=present

- name: Start MySQL

service: name=mysqld state=started

## **Play Breakdown**

---

- hosts: webservers

remote\_user: root

tasks:

- name: Install Apache

yum: name=httpd state=present

- name: Start Apache

service: name=httpd state=started

Each Play contains a host defined and the set of tasks mapping to that host.

A Play can contain a section for declaring variables specific to that Play.

e.g.

---

- hosts: webservers

vars:

git\_repo: https://github.com/repo.git

http\_port: 8080

db\_name: wordpress

**sudo: yes** **-> Giving Sudo permission for user executing the play**

sudo\_user: wordpress\_user

**gather\_facts: no -> Setting the Play not to gather facts on the Host in order to save time during execution**

## **Tasks**

Tasks are executed in Top Down order and uses Modules for its actions. Tasks Maps to a host.

Syntax of a Task is given below with an example

**tasks:**

**- name: Name this task for readability**

**module: parameters=go\_here**

**e.g.**

***- name: Deploy Apache Configuration File***

***copy: src=/ansible/files/conf/httpd.conf***

***dest=/etc/httpd/conf/***

## **Execution of Playbooks**

If the Inventory file is already declared in the Ansible Config file, then below is the command to execute Ansible Playbook.

$ **ansible-playbook playbook.yaml**

If the Inventory file is not declared in the Ansible Config file, then below is the command to execute Ansible Playbook.

$ **ansible-playbook playbook.yaml –i inventory**

## **Tasks Execution Failure**

If a Host in a playbook fails during execution, Playbook will remove that Host from the rest of the execution sequence. The Failed Hosts are later added to the Retry file which can used to execute the tasks against only those failed hosts. This avoids necessity to execute the playbook again for all the hosts.

For e.g. If a Host is scheduled with 10 tasks and the host failed at Task 4, then the playbook will not go to Task 5 and the Host will be removed from the execution.

Retry File e.g.

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To retry, use: --limit @/home/vagrant/ping.retry

db1 : ok=0 changed=0 unreachable=1 failed=0

web1 : ok=2 changed=0 unreachable=0 failed=0

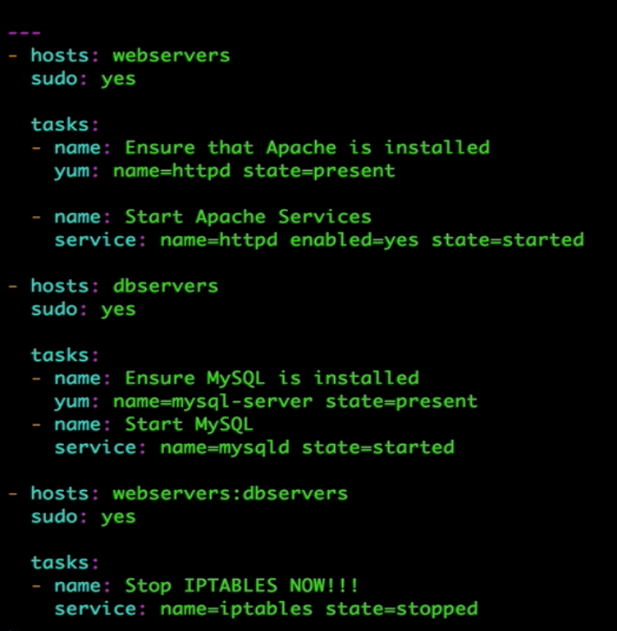
## **Sample Configuration File & Playbook**

**Configuration File**

[defaults]

hostfile = inventory

**Playbook**



## **Playbook Advanced Feature**

**Common Include Files**

Include files are used to break long Playbooks in to small manageable containers. It can also be used to reuse other Playbooks inside a playbook and to add externa variables.

tasks:

- include: wordpress.yml **-> Reusing other Playbooks**

vars:

sitename: My Awesome Site

- include: loadbalancer.yml

- include\_vars: variab**les.yml -> e.g. for External variables. Declare Variables in another file and call that file.**

**Register Task Output**

Useful to use tasks to feed data into other tasks. Useful to create custom error trapping i.e. to get the success or failure of the response.

tasks:

- shell: /usr/bin/whoami

register: username **-> Output of “whoami” is stored in variable username**

- file: path=/home/myfile.txt

owner={{ username }} **-> Variable username is used here.**

**Debug Modules**

Useful to send output to screen during execution. Helps find problems

tasks:

- debug: msg=“This host is {{ inventory\_hostname }} during execution”

- shell: /usr/bin/whoami

register: username

- debug: var=username **-> Print the output in the Screen while execution**.

**Prompting for Input**

Used to prompt for user input while playbook execution. This creates a dynamic playbook.

- hosts: web1

**vars\_prompt:**

**- name: "sitename"**

**prompt: "What is new site name?" -> User will be asked to Key-in value in keyboard here.**

tasks:

- debug: msg=“The name is {{ sitename }}"

**Playbook Handlers**

Handlers are used in Playbook to deal with tasks that needs to deal with an action only certain circumstances.

* Tasks with asynchronous execution
* Only runs tasks when notified
* Tasks only notify when state=changed
* Does not run until all playbook tasks have executed
* Most common for restarting services to load changes (if changes are made)

For e.g. The below Handler to restart Apache Service will run only in case if a Copy action returns the state=Changed.

tasks:

- copy: src=files/httpd.conf

dest=/etc/httpd/conf/

notify:

- Apache Restart **-> Handler “Apache Restart” Called.**

**handlers:**

**- name: Apache Restart**

**service: name=httpd state=restarted**

**Conditional Execution**

Uses a “When” clause to choose logically when a task should run.

For e.g. below task uses the module YUM when the OS type is REDHAT and uses APT module when the IS type is Debian.

tasks:

- yum: name=httpd state=present

**when: ansible\_os\_family == “RedHat”**

- apt: name=apache2 state=present

**when: ansible\_os\_family == “Debian”**

***Note: In the above e.g. ansible\_os\_family is a value retrieved by gathering facts from the host by setting gather\_facts yes***

**Conditional Clause based on Output**

tasks:

- command: ls /path/doesnt/exist

register: result

ignore\_errors: yes

- debug: msg=“Failure!”

**when: result|failed**

The above task prints a debug message in the console if the previous task Failed.

**Templates**

Templates are easy to use. It uses the Jinja 2 Engines to insert variables in to Static files and making it a dynamic file. Templates are mostly used for deploying custom configurations.

In the below e.g. The template takes a file with predefined variable names, inserts variable values in file and copies the file to a destination.

Template Module:

tasks:

- template:

src=templates/httpd.j2

dest=/etc/httpd/conf/httpd.conf

owner=httpd

httpd.j2 Jinja File:

……

<VirtualHost \*:80>

ServerAdmin {{ server\_admin }}

DocumentRoot {{ site\_root }}

ServerName {{ inventory\_hostname }}

</VirtualHost>

……

# **Roles**

## **Role Basics:**

**What are Roles and why are they important?**

After we learn the basics of any automation framework, the next step would be to optimize it. One way of optimization is reusability. Every module we write should be able to be reused. In Ansible there are various ways to do this. In the previous section, an example of reusability would be to use the include statement, which inserts other plays and tasks into a playbook, also helping to break up the playbook into more readable sections, in the process.

Roles in Ansible, build up on a similar idea. They are used to include files and combine them to form clean, reusable abstractions. Each Role can be defined to cater to different requirements.

Each Role can be associated with its own tasks, variables, handlers, templates etc.

Roles are defined via a directory structure. A particular role is then a sub directory under the Parent Roles Directory. This sub-directory is further divided into more directories namely defaults, files, handlers, meta, tasks, templates, and vars, wherein each folder may or may not be mandatory. Some are required only if we need to include that particular information in our playbook.



Figure *Role Tree*: The tree shows two roles: server-common and webserver and the various functions  
associated with them.

In the above figure we can see the example of a role called webserver. The webserver role has its own tasks, handlers, vars and templates. In the succeeding section, we will see this break up and how these roles can be deployed

## **Demonstration of a Role**

A typical playbook, without using the abstraction of Roles, would contain all tasks, handlers, vars within itself.

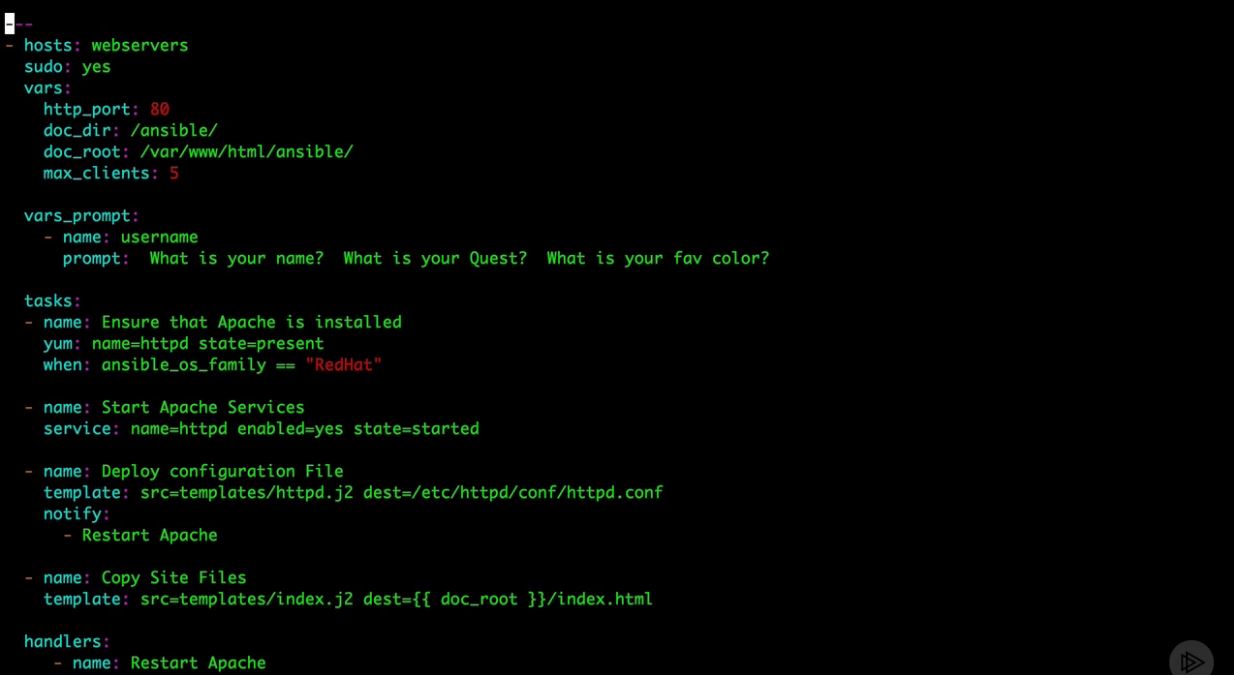


Figure: *A typical Playbook.*

To Breakdown this playbook and associate into roles, we can use the following structure:

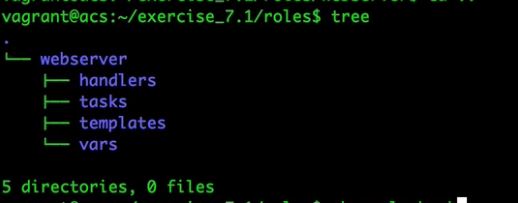


Figure: *Tree structure of Role webserver.*

We have defined a Role webserver, under the Roles directory, which is linked to a set of tasks, vars, templates and handlers.

**Tasks:**

The tasks directory will contain a main.yml file, which will include a set of all tasks we want to associate with the webserver role. If the number of tasks are very large, we can use include statements to break this down as well.



Figure *Sample Tasks main.yml file*.

**Handlers:**

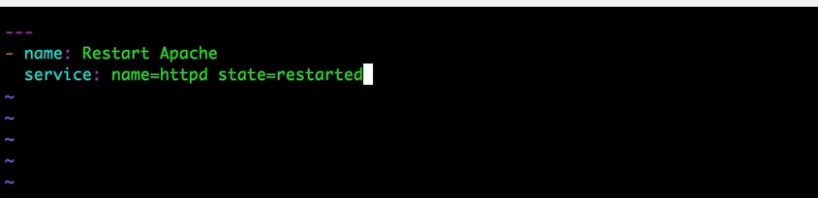
****

Figure *Sample Handlers main.yml file*.

**Vars:**

The vars main.yml file contains key value pairs.

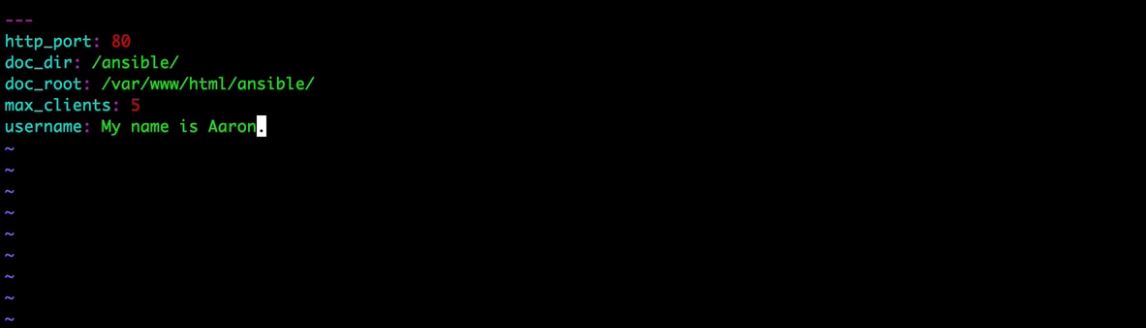


Figure *Sample Vars main.yml file*.

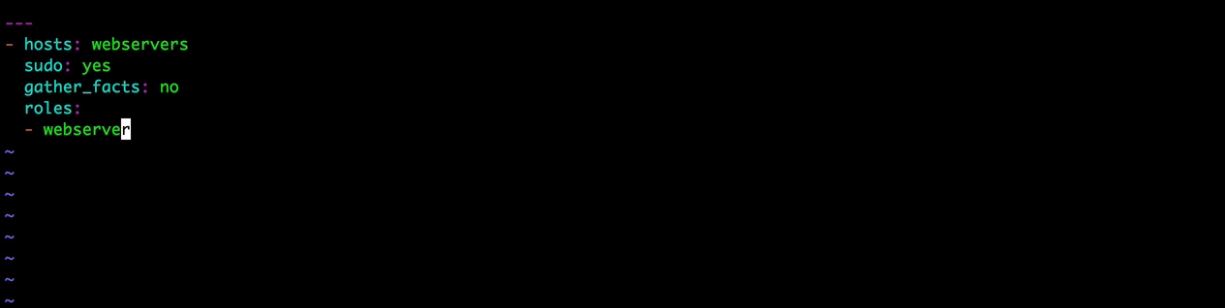
**Templates:**

****

Figure *Sample Templates Folder.*

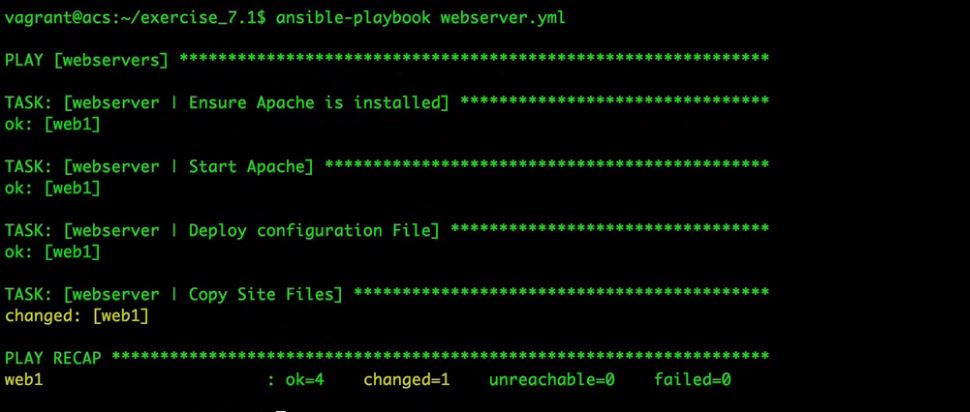
Note: Details on templates is out of the scope of the document. Please refer to links provided, for further information.

**Final Playbook with Role abstraction:**

****

*Figure Sample Playbook with roles.*

**Running the playbook:**



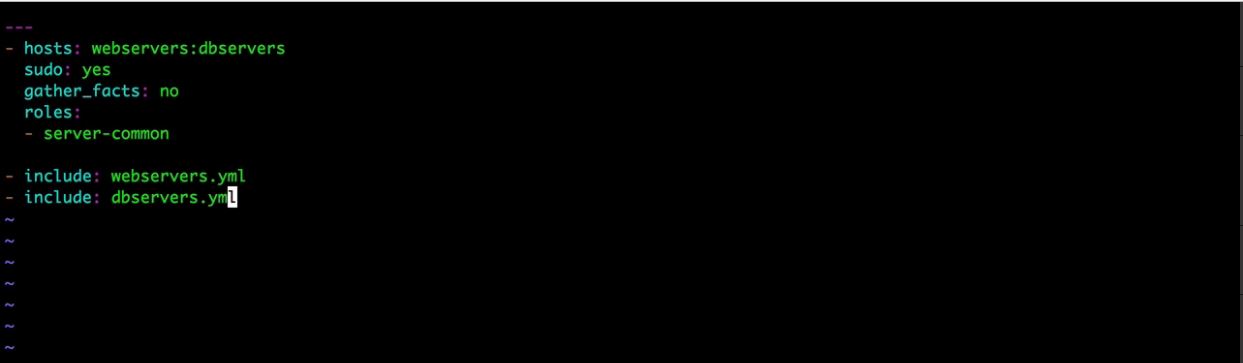
*Figure: Running the playbook with Roles.*

On running the playbook, the data associated with the Role webservers are automatically

referenced, and the playbook runs successfully.

## **Site.yml**

If we need to run multiple playbooks at the same time, this also can be done by creating a file called Site.yml and placing the playbooks to be run inside this.



*Figure Site.yml*

Here running site.yml runs both webservers.yml and dbservers.yml.

We can run Site.yml similar to how we run a playbook:

**ansible-playbook site.yml**

# **Reference Link:**

<http://docs.ansible.com/>

<http://docs.ansible.com/ansible/latest/modules.html>

<http://docs.ansible.com/ansible/latest/playbooks.html>

<http://docs.ansible.com/ansible/latest/galaxy.html>

<http://docs.ansible.com/ansible/latest/faq.html>