**ANSIBLE TUTORIAL**

1. CONFIGURATION MANAGEMENT

2. SOME TERMINOLOGIES

3. ANSIBLE INTRODUCTION

4. INSTALLING ANSIBLE

5. INVENTORY

6. INVENTORY FOR PRODUCTION SYSTEMS/REALTIME

7. ADHOC COMMAND

8. ABOUT MODULES

9. ANSIBLE CONFIGURATION

10. SETUP - GATHERS FACTS ABOUT REMOTE HOSTS

11. PLAYBOOKS

12. PLAYBOOK LANGUAGE EXAMPLE

13. YAML BASICS

14. FIRST PLAYBOOK EXERCISE

15. PLAYBOOK EXECUTION

16. VARIABLES

17. INCLUDING PLAYBOOKS

18. STORE OUTPUT OF A COMMAND

19. DEBUG MODULE

20. PROMPTING FOR INPUT

21. HANDLERS

22. CONDITIONAL EXECUTION

23. TEMPLATES

24. A SAMPLE PLAYBOOK WITH VARIABLES, TEMPLATES, CONDITIONS AND HANDLERS.

25. ROLES

26. ANSIBLE GALAXY

27. ANSIBLE VAULT MANAGING SECRETS WITH ANSIBLE VAULT

28. LOOPING IN ANSIBLE

29. CONFIGURE APACHE USING ANSIBLE

30. MULTI-TIER WEB APPLICATION STACK DEPLOYMENT USING ANSIBLE

31. WORDPRESS SETUP WITH ANSIBLE

**XII. Ansible Tutorial**

Ansible is a very powerful open source automation tool. It comes in the category of Configuration Management tools like Puppet, Chef, Saltstack etc. Its simplest among all the configuration management tool in terms of its easy to read & write approach and Masterless/serverless Architecture .

Being simple in use it can handle most complex of the task when it comes to Orchestration, where you run automation tasks in chain and order on several different servers and devices.

1. **Configuration Management**

In simple terms, tools that manages configuration of IT infrastructures like OS, network devices, Application, softwares and Cloud computing services are called as Configuration Management tools.

Configuration of IT Infrastructure like Softwares & OS changes happens every now and then. We add, remove and update them for good. Doing such changes to hundreds of servers and devices is very time consuming and error prone task if you do it manually.

These tools help you manage and automate all that changes with ease and from a centralised place.

**Some features**

**Centralised configuration**

Configuration of various servers and devices are managed from a central server. Like you would be managing webservers, dbservers or switches all of their configurations like softwares,patches,config files etc can be setup in the central server. When the node wants to update the latest changes,it can fetch from the server or pushed to the node from server.

**Enforcement.**

Configuration enforcement may be the single most important feature of a configuration management tool. By running regularly and ensuring the machine is configured to the desired state, configuration management tools prevent configuration to go out of sync. Configuration drift can happen in a variety of ways: Package updates, live debugging, "helpful" co-workers, etc. Whatever the cause, being able to say with confidence, "This is how this machine is configured," is a great way to shorten incident resolution time and reduce surprises.

**Abstraction.**

Few sysadmins maintain completely homogeneous environments. Even if you're an all-Linux shop, you probably have multiple distros that you support, or at least multiple versions of a distro. With configuration management tools, many of the operating-system-specific implementations of a configuration are abstracted away for you. The same configuration file can be used to manage, for example, the installation of Apache HTTPD on both Red Hat and Ubuntu systems.

**Version control friendly.**

Of course, the best way to enable cooperation is to have everything in a version control system. All of the tools listed below use some form of text for configuration. This means you can take advantage of the benefits of your favourite version control system.

**Replication.**

Configuration management makes it easy to replicate environments with the exact same software and configurations. This enables you to effectively build a multistage ecosystem, with production, development, and testing servers.

**Some Configuration Management Tools**

• Puppet

• Chef

• Ansible

• Saltstack

• CFEngine

**2. Some Terminologies**

**Change Management**

It is process when any specific configuration of the machine or software is changed. A typical CM

process in IT industry goes through a approval process from customer or higher management, of

course we are talking about production systems. A single change on 1, 10 or 100’s of servers has to

be done very precisely and effectively. For example, upgrading a software package which has a bug

on hundreds of servers or restarting a service like webservice to take effect of any new changes.

**Provisioning**

In general, provisioning means "providing" or making something available in the IT infrastructure.

For example, provisioning a web server means installing and setting up web service softwares and

its configuration on a OS. Provisioning a cloud instance means creating a virtual machine on the

cloud.

**Orchestration**

Automation is when we group list of tasks and execute them in top to down order through a script generally like installing Webserver through a script. Whereas Orchestration is the process where we group multiple automated tasks and execute them into an order.

For example, if we are orchestrating setup of a multi-tier web application, we have to setup all the services like Databases, Webservices, Loadbalancer, monitoring in an order so it gets validate whenwe have entire setup running.

First, we will setup Database service so when webservice is setup it gets connected to database and gets validated. Next, we may setup Loadbalancer and add webservices under it. Monitoring would be setup at the very end.

Now monitoring services will return right results from loadbalancer, web service and database service. This process is called as Orchestration but not limited to just this and is done for variety of other IT infrastructure process like cloud computing, network setup etc.

Orchestration word has come from the word Orchestra where different instruments are played in a proper order to generate the beautiful Melody. It’s all about order otherwise you don’t get music all you get is sound and probably not so good.

Similarly, IT orchestration is all about automating tasks in a proper order.

**Idempotent Behaviour**

Configuration management tools keep track of the state of resources in order to avoid repeating tasks that were executed before. If a package was already installed, the tool won't try to install it again. The objective is that after each provisioning run the system reaches (or keeps) the desired state, even if you run it multiple times. This is what characterizes these tools as having an *idempotent behaviour.* One more example would be if we are trying to push a file to multiple servers and some servers already have the same file with the same content then it’s not going to overwrite the file, will simply skip it and push it to their servers where there is a mismatch.

**3.** A**nsible Introduction**

Ansible can control large number of servers and eases administration and operations tasks. Ansible can do simple configuration management and complex orchestration, it has all the features thatconfig tools have plus it’s very easy to learn and implement.

It communicates over normal SSH channels in order to retrieve information from remote machines, issue commands, and copy files. For windows node it uses winrm.

Since it uses SSH for Linux and winrm for windows there no need to setup anything at the agent side as ssh comes by default in Linux servers and so as winrm in windows server.

**4. Installing Ansible**

Ansible by default manages machines over the SSH protocol.

Once Ansible is installed, it will not add a database, and there will be no daemons to start or keep running. You only need to install it on one machine (which could easily be a laptop) and it can manage an entire fleet of remote machines from that central point.

**Latest Release Via Yum**

RPMs are available from yum for **EPEL** 6, 7, and currently supported Fedora distributions. Ansible itself can manage earlier operating systems that contain Python 2.6 or higher (so also EL6). Fedora users can install Ansible directly, though if you are using RHEL or CentOS and have not already done so, **configure EPEL**

# *install the epel-release RPM if needed on CentOS, RHEL, or Scientific Linux*

*$ sudo yum install ansible*

**Latest Releases Via Apt (Ubuntu)**

Ubuntu builds are available in a PPA here.

(Windows isn't supported for the control machine). To configure the PPA on your machine

and install ansible run these commands:

*$ sudo apt-get install software-properties-common*

*$ sudo apt-add-repository ppa:ansible/ansible*

*$ sudo apt-get update*

*$ sudo apt-get install ansible*

**Some Quick Notes:**

> Machine, where ansible is installed is called as **Control Machine.**

> Ansible is written in Python Language.

> You should have python 2.6/2.7 to install ansible on control machine.

> Windows isn’t supported for the control machine.

> Ansible can automate tasks on Linux and windows Machines

**5. Inventory**

Inventory is a text file where you define the host information that you want to manage with ansible. The default inventory file location is /etc/Ansible/hosts. You can specify a different inventory file using the -i <pat h> option on the command line.

For this exercise we need two Linux servers, you can spin two centos vm or ec2 instance for practice.

**Hosts and Groups**

Create a file named inventory-dev(name can be anything) and add below mentioned entry.

*webl ansible\_ssh\_host=l92.168.1.13 ansible\_ssh\_user=vagrant ansible\_ssh\_pass='vagrant'*

*dbl ansible\_ssh\_host=l92.168.1.14 ansible\_ssh\_user=vagrant ansible\_ssh\_pass='vagrant'*

*[webservers]*

*Webl*

*[dbservers]*

*dbl*

**Explanation**

> webl and dbl are the names that we have given to the hosts.

> ansible ssh host is the variable and its value is the IP address of the server.

> ansible ssh user variable holds the username

> ansible\_ssh\_password holds the password

> [Webserver] & [dbservers] is the name of the group which can contain n number hosts.

Groupnames are enclosed in square brackets [] .

**Note:**  Mentioning password in the inventory file is not recommended, it’s just for initial learning later we will do ssh key exchange.

**6. Inventory for Production systems/Real Time.**

As we have seen above we put the password in clear text and IP address information also in the inventory. This is a real concern for security, you cannot share this inventory with anyone and also cannot track it in VCS like git. We have better ways to deal with this situation.

**1.** Since ansible uses SSH, its always recommended to do SSH key exchange and authorize ansible server login to the nodes its managing.

**Note:** Refer Bash Scripting chapter to learn SSH key exchange.

This way we don’t need to mention username and password in the inventory file.

**2.** Next thing is the IP address, we can manage that with the /etc/hosts file. Map IP to hostname in /etc/hosts file and you can then mention the hostname directly in the inventory.

**HOSTS File**

*$ cat /etc/hosts*

*192.168.1.13 webl*

*192.168.1.14 dbl*

**INVENTORY File**

*$hcat inventory-dev*

*[webservers]*

*Webl*

*[dbservers]*

*dbl*

So now our inventory is very simple and just contain the group and hostname, which is safe.

**7. Adhoc command**

Ansible gives a quick method to communicate and execute commands on remote/local machines

through Adhoc commands.

*imran@DevOps:*~/...*/exercisel$ ansible -i inventory-dev -m ping webl*

*webl SUCCESS => {*

*"changed": false,*

*"ping": "pong"*

*}*

*imran@DevOps:*~/...*/exercisel$ ansible -i inventory-dev -m ping dbl*

*dbl SUCCESS => {*

*"changed": false,*

*"ping": "pong"*

*}*

*imran@DevOps:*~/...*/exercisel$ ansible -i inventory-dev -m ping webservers*

*webl SUCCESS => {*

*"changed": false,*

*"ping": "pong"*

*}*

*imran@DevOps:*~/...*/exercisel$ ansible -i inventory-dev -m ping dbservers*

*dbl* | *SUCCESS => {*

*"changed": false,*

*"ping": "pong"*

***}***

*imran@DevOps:*~/...*/exercisel$ ansible -i inventory-dev -m ping all*

*webl SUCCESS => {*

*"changed": false,*

*"ping": "pong"*

*}*

*dbl* | *SUCCESS => {*

*"changed": false,*

*"ping": "pong"*

*}*

**Explanations**

> Adhoc commands are executed by specifying **“ansible”** command.

> -i inventory name> tells ansible to pick up host information from this file, if not specified ansible will look for the host information in /etc/ansible/hosts

> -m means the module name, **ping** is a module which will login to the host and check the connectivity.

> Webl, dbl, webservers, dbservers are the host and group name where we want to execute the task, “all” means all the hosts from the inventory file.

**Host key checking Error**

*"msg":"ERROR! Using a SSH password instead of a key is not possible because*

*Host Key checking is enabled and sshpass does not support this. Please add this*

*host's fingerprint to your known\_hosts file to manage this host."*

*}*

**Solution:**

Open /etc/ansible/ansible.cfg file and uncomment “#host\_key\_checking = False” by removing #

sudo vi /etc/ansible/ansible.cfg

**Group of Groups and Group Variables**

Variables can be applied to an entire group

*[webservers]*

*webl*

*[dbservers]*

*dbl*

*[datacenter:children]*

*webservers*

*dbservers*

*[datacenter:vars]*

*ansible\_ssh\_user=vagrant*

*ansible\_ssh\_pass= ’vagrant ’*

**Explanation**

> [datacenter:children] tells datacenter is a master group and underneath we specify other

group names

> [datacenter:vars] is used to define variable at the group level, you can specify variable on

any group by specifying [groupname:vars].

**O**

General for all connections:

**Ansible\_host**

The name of the host to connect to, if different from the alias you wish to give to it.

**Ansible\_port**

The ssh port number, if not 22

**Ansible user**

The default ssh user name to use.

Specific to the SSH connection:

**ansible\_ssh\_pass**

The ssh password to use (never store this variable in plain text; always use a vault. See (Variables and Vaults')

**ansible\_ssh\_private\_key\_file**

Private key file used by ssh. Useful if using multiple keys and you don’t want to use SSH agent.

k

**8. About Modules**

Modules (also referred to as “task plugins” or “library plugins”) are the ones that do the actual work in ansible, they are what gets executed in each playbook task. But you can also run a single one using the ‘ansible’ command.

**List of all the modules**

[https://docs.ansible.com/ansible/list of all modules.html](https://docs.ansible.com/ansible/list%20of%20all%20modules.html)

*imran@DevOps:*~/...*/exercisel$ ansible-doc -1*

Let’s review how we execute three different modules from the command line:

*ansible webservers -m service -a "name=httpd state=started"*

*ansible webservers -m ping*

*ansible webservers -m command -a "/sbin/reboot -t now"*

**Installing package**

*imranQDevOps:*~/..*./exercisel$ ansible*

*state=installed" webl --sudo*

*i inventory-dev -m yum -a "name=httpd*

*webl* | *SUCCESS => {*

*"changed": false,*

*"msg": rr rt*

*"rc": 0,*

*"results": [*

*"httpd-2.2*.*15-59*.*el6.centos.x86\_64 providing httpd is already installed"*

*]*

*}*

**Explanation**

**> “yum”** is a ansible module that manages packages on red hat based systems, for Debian

based we use module named **“apt”.**

**> -a** is used to provide arguments for the module like name=httpd(key=value). Majority of the modules will have arguments, some arguments are mandatory like “name” argument for “yum”.

**> $ ansible-doc yum** will show you list of all the arguments for yum module.

**> -sudo** tells ansible to execute the module with root privileges, user should have the sudo privileges or else the module will fail.

**Starting service.**

*imrandDevOps:*~/...*/exercisel$ ansible -i inventory-dev -m service -a "name=httpd*

*state=started" webl --sudo*

*webl* | *SUCCESS => {*

*"changed": true*,

*"name": "httpd",*

*"state": "started"*

*}*

**Output of adhoc commands**

ansible command return output is json format

*webl SUCCESS => {*

*"changed":* true,

*"name": "httpd",*

*"state": "started"*

*}*

> webl is the name of the hosts on which module got executed.

> Status is SUCCESS that means it got executed successfully.

> changed: true means that the module execution made some changes in webl.

**>** changed: false means that the system is in the same desired state as shown below.

*imran@DevOps:*~/...*/exercisel$ ansible -i inventory-dev -m service -a "name=httpd*

*state=started" webl --sudo*

*webl SUCCESS => {*

*"changed": false,*

*"name": "httpd",*

*"state": "started"*

*}*

> httpd service was already running on webl so even executing the adhoc command again will not make any changes this is called the **IDEMPOTENT** behaviour.

**Few more sample modules with adhoc commands.**

To transfer a file directly to many servers:

***$ ansible -i inventory-dev -m copy -a "src=/etc/hosts dest=/tmp/hosts"datacenter***

The file module allows changing ownership and permissions on files. These same options can

be passed directly to the copy module as well:

*$ ansible webservers -m file -a ”dest=/opt/info.txt mode=600"*

*$ ansible webservers -m file -a "dest=/opt/info.txt mode=600 owner=devops*

*group=devops"*

Ensure a package is installed, but don’t update it:

*$ ansible webservers -m yum -a "name=acme state=present"*

Ensure a package is installed to a specific version:

*$ ansible webservers -m yum -a "name=acme-l.5 state=present"*

Ensure a package is at the latest version:

*$ ansible webservers -m yum -a "name=acme state=latest"*

Ensure a package is not installed:

*$ ansible webservers -m yum -a "name=acme state=absent"*

**9. Ansible Configuration**

Certain settings in Ansible are adjustable via a configuration file. The stock configuration should be sufficient for most users, but there may be reasons you would want to change them.

Changes can be made in global ansible.cfg file /etc/ansible/ansible.cfg or you can create your own ansible.cfg (current working directory) which will have higher precedence over the global file.

*$ cat ansible.cfg*

*[defaults]*

*hostfile inventory\_prod*

*host\_key\_checking=False*

*#ask\_sudo\_pass True*

This ansible.cfg file is located at the same place where you have the inventory file.

**Explanation**

> [defaults] is the main section of ansible.cfg

> hostfile will have the value where inventory file is located if it’s in the current working directory specify the name or else complete path of the file should be specified. After mentioning inventory path in ansible.cfg its not required to pass the inventory path with –I option.

> host\_key\_checking=False tells ansible to not check the host fingerprints before doing ssh to the host.

There long list of ansible config parameters that you can choose from. List of ansible configuration is specified in ansible documentation.

[**https://docs.ansible.com/ansible/intro configuration.html**](https://docs.ansible.com/ansible/intro%20configuration.html)

**10. Setup - Gathers facts about remote hosts**

It can be executed directly by/usr/bin/ansible to check what variables are available to a host. Ansible provides many *facts* about the system, automatically.

*$ ansible -m setup webl*

*webl* **|** *SUCCESS => {*

*"ansible facts": {*

*"ansible\_all\_ipv4\_addresses": {*

*"10.0.2.15* **",**

*"192.168.1.13"*

**},**

*],*

"ansible\_all\_ipv6\_addresses": {

"fe80::aOO:27ff:fel5:d519",

”fe80::aOO:27ff:febf:5936"

},

"ansible architecture": "x86 64",

"ansible bios date": "12/01/2006",

"ansible bios version": "VirtualBox",

"ansible cmdline": {

**output trimmed**

"ohai\_uptime": "1 hours 38 minutes 26 seconds",

"ohai\_uptime\_seconds": 5906,

"ohai virtualization": {

"role": "guest",

"system": "vbox"

}

},

“Changed”: False

# *Display facts from all hosts and store them indexed by I(hostname) at*

*C(/tmp/facts).*

# *ansible all -m setup --tree /tmp/facts*

# *Display only facts regarding memory found by ansible on all hosts and output*

*them*.

# *ansible all -m setup -a 'filter=ansible* \* *mb'*

# *Display only facts returned by facter.*

# *ansible all -m setup -a 'filter=facter \* i*

# *Display only facts about certain interfaces*.

# *ansible all -m setup -a filter=ansible eth[0-2]'*

# *Restrict additional gathered facts to network and virtual.*

*# ansible all -m setup -a 'gather\_subset=network,virtual'*

# *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 t n*

**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 " - 11 (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 webl with a sample website and MySQL db to dbl node.

**Inventory file**

*$ cat inventory-dev*

*webl ansible ssh host=l92.168.1.13*

*dbl ansible ssh host=l92.168*.*1.14*

*[Webservers]*

*Webl*

*[clbservers]*

*Dbl*

*[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">*

*<hl> My first page </hl>*

*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 runninq*

*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*

*TASK [Gathering Facts]*

*k -k 'k -k -k -k -k -k -k 'k 'k 'k i c i c -k -k -k T k -k -k -k i i r -k -k -k -k -k -k -k -k -k -k -k -k -k -k i c T k T k i c i c i c i c -k -k -k i c -k -k -k -k -k -k -k -k -k -k -k 'k*

*ok: [webl]*

*TQSK [Ensure Apache installed]*

*k 'k 'k i c i c i f i c i c 'k -k -k ^c -k -k -k -k -k -k i c -k -k -k -k -k -k 'k 'k 'k 'k i c ^ ^r ^r ^c ^c ^c ^ ^ ^c ^ ^ -k -k -k -k -k -k -k -k*

*changed: [webl]*

*TASK [Creates directory]*

*changed: [webl]*

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

*changed: [webl]*

*TASK [Ensure Apache is running]*

*'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k ’k i c 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k 'k i i r i i r i c i c i c i c i c i c*

*changed: [webl]*

*TASK [Flush all temporary rules]*

*changed: [webl]*

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

*changed: [webl]*

*PLAY [dbservers]*

*TASK [Gathering Facts]*

*ok: [dbl]*

*TASK [Ensure mysql server installed]*

*\*\*\*\*\*\*\*\*\*\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \**

*changed: [dbl]*

*TASK [Ensure mysql running]*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*changed: [dhl]*

*TQSK [Ensure MySQL-python is installed]*

*changed: [dhl]*

*TASK [Create Database]*

*changed: [dhl]*

*TASK [Create user named mint]*

*changed: [dhl]*

*PLAY RECAP*

*dhl : ok=6 changed=5 unreachable=0 failed=0*

*webl : 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: [webl]** is a status message for the task, it means that the task made changes to the target host.

**> ok: [webl]** 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**

i

production/

group\_vars

L all

L inventory

staging/

| group\_vars

I all

I 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/webl variables will be only accessible for the webl 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/webl*

# *Variables exposed for the host named webl from the inventory file*.

*user: edison*

*group: electric*

*pass: 12345*

If we try to access user variable for dbl its value will come from group\_vars/all file.

*$ ansible -m user -a "name={{user}} password^{{pass}}" --sudo* ***dbl***

*dbl* | *SUCCESS => {*

*"changed":*

*"comment":*

*"createhome":*

*"group": 1002,*

*"home": "/home/tesla",*

***"name”: "tesla",***

*"password": "NOT LOGGING PASSWORD",*

*"shell":*

*"state": "present",*

*"system": false,*

*"uid": 1002*

*}*

For webl host we created variables in host\_vars/webl. host\_vars variable will high higher precedence, so it will ignore user variable from group\_vars/all file and pickup value from host vars/webl file.

*$ ansible -m user -a "name={{user}} password={{pass}}" --sudo* ***webl***

*webl SUCCESS => {*

*"changed":*

*"comment":*

*"createhome":*

*"group": 1002,*

*"home": "/home/edison",*

***"name": "edison",***

*"password": "NOT LOGGING PASSWORD",*

*"shell": rt rt*

*"state": "present",*

*"system": false,*

*"uld": 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}}*

**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}}"*

*$ anslble-playbook deb.yml*

*PLAY [all]*

*TASK [Gathering Facts]*

*ok: [webl]*

*ok: [dbl]*

*TASK [debug]*

*k -k -k -k 'k -k ^c i c ^c ^c ^c ^c -k -k -k -k -k -k -k -k -^ -^ -k ^c i c -k 'k 'k i c -k 'k ^c -k 'k 'k 'k ^c ^c i c ^c ^c i c i c i c -k -k -k -k -k -k -k -k -k -k -k i c i c ^c ^c ^c ^c -k -k -k -k -k -k -k -k*

*ok: [webl] => {*

*"changed": false,*

*"msg": "Inventory hostnames are webl"*

*}*

*ok: [dbl] => {*

*"changed": false,*

*"msg": "Inventory hostnames are dbl"*

*}*

*TASK [debug]*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*&\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*&\*\**

*ok: [webl] => {*

*"changed": false,*

*"msg": "Port variable is 8087 & username is cassini"* 4

***}***

*ok: [dbl] => {*

*"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:*

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

*$ ansible-playbook prompt.yml*

*Enter password for database.:*

*PLAY [dbservers]*

*TASK [Gathering Facts]*

*ok: [dbl]*

*TASK [debug]*

*ok: [dbl] => {*

*"changed": falsef*

*"msg": "DB password is deltaql23"*

*}*

**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 webl*

*"ansible\_os\_family":*

*$ 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.

i

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>*

*<headxtitle>Hello from Ansile</title></head>*

*<body>*

*<hl>Congratulations!</hl>*

*<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.

*$'Is*

*ansible.cfg inventory\_prod templates Vagrantfile web.yaml*

*$ cat ansible.cfg*

*[defaults]*

*hostfile inventory\_prod*

*host\_key\_checking = False*

*$ cat inventory\_prod*

*webl ansible ssh host=192.168.1.11*

*dbl ansible ssh host=l92.168.1.8*

*[webservers]*

*webl*

*[dbservers]*

*dbl*

*[datacenter:children]*

*webservers*

*dbservers*

*[datacenter:vars]*

*ansible ssh user=vagrant*

*ansible\_ssh\_pass=vagrant*

*$ Is templates/*

*httpd*.*j2 index*.*j2*

**Template for index.html file**

*$ cat templates/index.j2*

*<html>*

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

*<body>*

*<hl>Congratulations!</hl>*

*<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*

*vars\_prompt:*

*name: username*

*prompt: What is your name?*

*tasks:*

*name: Ensure libselinux-python installed*

*yum: name=libselinux-python state=present*

*name: Ensure Apache installed*

*yum: name=httpd state=present*

*when: ansible\_os\_family*

*name: Creates directoy*

*file: path=/var/www/html/ansible state=directory*

*name: Ensure Apache installed*

*yum: name=httpd state=present*

*when: ansible\_os\_family*

*name: Ensure Apache is running*

*service: name=httpd enabled=yes state=started*

*name: Deploy configuration File*

*template: src=templates/httpd*.*j2 dest=/etc/httpd/conf/httpd.conf*

*notify:*

*name: Copy Site Files*

*template: src=templates/index.j2 dest={{doc\_root}}/index.html mode=0644*

*name: Stop IPTABLES Now!!*

*service: name=iptables state=stopped*

*handlers:*

*name: Restart Apache*

*service: name=httpd state=restarted*

**25. Roles**

While it’s possible to write a playbook in one very large file eventually you’ll want to reuse files and start to organize things.

In above example, we have seen in our playbook we have variables, tasks, handlers and templates.

This can slowly grow and will become eventually difficult to read and manage.

Roles is a directory structure where we distribute the content of our main playbook into proper directory structure.

**Roles directory structure.**

This is what they are all for:

**•files:** This directory contains regular files that need to be transferred to the hosts you are configuring for this role. This may also include script files to run.

**•handlers:** All handlers that were in your playbook previously can now be added into this directory.

**•meta:** This directory can contain files that establish role dependencies. You can list roles that must be applied before the current role can work correctly.

**•templates:** You can place all files that use variables to substitute information during creation in this directory.

**•tasks:** This directory contains all of the tasks that would normally be in a playbook. These can reference files and templates contained in their respective directories without using a path.

**•vars:** Variables for the roles can be specified in this directory and used in your configuration files. Instead of having all our code in one playbook mashed up, we can distribute it into different directory structure. For example, all the tasks from our playbook will got into roles/apache/tasks/main.yml file, likewise vars go into vars/main.yml.

We can create the roles directory structure with ansible-galaxy command.

*$ mkdir roles*

*$ cd roles/*

*imran@DevOps:...*/*roles$ ansible-galaxy init apache*

*- apache was created successfully*

*imran@DevOps:.../roles$ tree*

L *apache*

*defaults*

*main.yml*

| *handlers*

*main.yml*

| *meta*

*main.yml*

*README.md*

*tasks*

L *main.yml*

*tests*

*inventory*

L *test.yml*

L *vars*

L *main.yml*

*$ cd ..*

**Our main playbook will just call the role by its name and will not have any tasks.**

*$ cat Webservers.yml*

*hosts: webservers*

*sudo: yes*

*gather\_facts: no*

*roles:*

*- apache*

*name: Ensure Apache installed*

*yum: name=httpd state=present*

*name: Creates dlrectoy*

*file: path=/var/www/html/anslble state=dlrectory*

*name: Ensure llbsellnux-python Installed*

*yum: name=llbsellnux-python state=present*

*name: Ensure Apache is running*

*service: name=httpd enabled=yes state=started*

*name: Deploy configuration File*

*template: src=templates/httpd.j2 dest=/etc/httpd/conf/httpd.conf*

*notify:*

*- Restart Apache*

*name: Copy Site Files*

*template: src=templates/index.j2 dest={{doc\_root}}/index.html mode=0644*

*name: Stop IPTABLES Now!!*

*service: name=iptables state=stopped*

*$ cat roles/apache/handlers/main.yml*

*name: Restart Apache*

*' service: name=httpd state=restarted*

*$ cat roles/apache/vars/main.yml*

*http\_port: 80*

*doc dir: /ansible/*

*doc root: /var/www/html/ansible/*

*max clients: 5*

*ansible python interpreter: python*

*username: devops*

*$ Is roles/apache/templates/*

*httpd*.*j2 index.j2*

It would be a good idea to first write a mashed-up playbook then we can start copying the content of the main playbook to the roles directory structure.

As the complexity grow we can then manage with the roles directory structure.

**Executing Playbook.**

We execute the main playbook which in turn will read the roles directory structure and execute all

the tasks & handlers for us.

*$ ansible-playbook Webservers.yml*

**Overriding variables of roles.**

We have few variables defined in our apache roles in roles/apache/vars/main.yml file.

When we execute our playbook it uses all these defined variables but we can override these

variables without changing the content of roles/apache/vars/main.yml file.

*hosts: webservers*

*sudo: yes*

*gather\_facts: no*

*roles:*

***- {role:apache, http\_port:8090, max\_clients:250}***

As shown above we are passing a dictionary now {} instead of the role name. Dictionary has key=value pairs for role name and variables that we want to override.

**26. Ansible Galaxy**

So far, we have got the idea of what ansible roles are and how to create them. There are so many predefined roles in Ansible Galaxy website which we can download and use freely. For example, if we want to setup mysql service, we can create mysql role from scratch or we can use some existing role from ansible galaxy.

https://galaxv.ansible.com/list#/roles?page=l&page size=10

We can search and find a relevant role for our work. Role gets reviewed and gets star marks.

Click on the role to read about it in detail. Check the supported platforms and ansible version which

it supports.

README section talks about how to use this role with examples.

**Examples**

1) Install MySQL Server and set the root password, but don't create any database or users.

hosts: all

roles:

- {role: mysql, root\_db\_pass: foobar, mysql\_db: none, mysql\_users: none }

2) Install MySQL Server and create 2 databases and 2 users.

hosts: all

roles:

- {role: mysql, mysql\_db: [ {name: benz},

{name: benz2} ],

mysql\_users: [ {name: ben3, pass: foobar, priv : &quot;\* . \* :ALL&quot;},

{name: ben2, pass: foo} ] }

**Download the ansible galaxy role.**

Role by default gets downloaded into /etc/ansible/roles directory. Once its downloaded you can start

using this role from your playbook. Refer to the example given in the README section of the role.

*$ sudo ansible-galaxy install bennojoy.mysql*

*downloading role ’mysql ’, owned by bennojoy*

*- downloading role from https://github.com/bennojoy/mysql/archive/master.tar.gz*

*- extracting bennojoy.mysql to /etc/ansible/roles/bennojoy.mysql*

*- bennojoy.mysql (master) was installed successfully*

**27. Ansible Vault**

**Managing secrets with Ansible vault.**

The vault feature can encrypt any structured data file used by Ansible. This can include “group\_vars/” or “host\_vars/” inventory variables, variables loaded by “include\_vars” or “vars\_files”, or variable files passed on the ansible-playbook command line with “-e @file.yml” or “-e @file.json”. Role variables and defaults are also included!

We can store our passwords/secrets encrypted in the ansible vault.

We will create vault in group\_vars/all directory.

*mkdir -p group\_vars/all*

*cd group\_vars/all*

*export EDITOR=vim*

*ansible-vault create vault*

Give a vault password and put variables that you want to encrypt, below mentioned is the example of storing tomcat password.

*vault\_tomcat\_pass : <Enter password for tomcat here>*

Refer to vault\_\_tomcat\_pass in group\_vars/all/vars file

*cd group\_vars/all/*

*vi vars*

*tomcatuser: tomcat*

*tomcatpass: "{{vault tomcat pass}}"*

If you execute your playbook now where you are using tomcatpass, you should get some error like below. For example, I am using {{tomcatpass}} variable in my tomcat.yml playbook.

# *ansible-playbook tomcat.yml*

*ERROR! Decryption failed*

*ERROR! A vault password must be specified*

You can give --ask-vault-pass option while executing playbook which will ask you the vault

password.

**#** *ansible-playbook tomcat.yml ask-vault-pass*

You can also use a file where you specify vault password. Lets say “vaultpass” is our vault password.

**#k** *echo "vaultpass" >* **-/.***vault\_pass.txt*

**#•** *chmod 0600 -/.vault pass.txt*

**Sample Execution.**

*imranODevOps:.***..***/ans$ tree*

**|** *group\_vars*

*all*

**|** *vars*

*vault*

*test.yml*

*2 directories, 3 files*

*imranODevOps:.../ans$ cat group vars/all/vars*

*tomcatpassword: "{{tomcatpass}}"*

*imranODevOps:.../ans$ cat test.yml*

*- hosts: localhost*

*tasks:*

*- name: Print tomcat pass*

*debug: msg="{{tomcatpassword}}"*

*imranODevOps:.../ans$ ansible-playbook test.yml --ask-vault-pass*

*Vault password:*

*[WARNING]: provided hosts list is empty, only localhost is available*

*PLAY*

*TASK [setup]*

*ok: [localhost]*

*TASK [Print tomcat pass]*

*ok: [localhost] => {*

*"msg": "tomcat"*

*}*

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

*localhost : ok=2 changed=0 unreachable=0 failed=0*

**Editing Vault**

To edit an encrypted file in place, use the *ansible-vault edit* command. This command will decrypt the file to a temporary file and allow you to edit the file, saving it back when done and removing the temporary file:

*ansible-vault edit vault*

**28. Looping in ansible**

To save some typing, repeated tasks can be written in short-hand like so:

with\_items is a ansible “for loop” which runs on a list of items. with\_items while executing return variable named “item” which hold the value of element in the list

*name: add several users*

*user:*

*name: "{{ item }}"*

*state: present*

*groups: "wheel"*

*with items:*

*testuserl*

*testuser2*

If you have defined a YAML list in a variables file, or the ‘vars’ section, you can also do:

with\_items: "{{ somelist }}"

The above would be the equivalent of:

*name: add user testuserl*

*user:*

*name: "testuserl"*

*state: present*

*groups: "wheel"*

*name: add user testuser2*

*user:*

*name: "testuser2"*

*state: present*

*groups: "wheel"*

The yum and apt modules use with\_items to execute fewer package manager transactions.

# *tasks file for mysql*

*name: Install Mysql package*

*yum:*

*name: "{{ item }}"*

*state: installed*

*with items:*

*- mysql-server*

*- MySQL-python*

*libselinux-python*

*- libsemanage-python*

**29. Configure Apache Using Ansible**

***Introduction***

Apache is one of the most popular web servers currently used on the Internet. It is easy to set up and configure on Linux distributions like Ubuntu and Debian, as it comes in the package repositories and includes a default configuration that works out of the box.

***Prerequisites***

We will install Ansible on a Ubuntu server and use it to configure Apache on a second server.

For this tutorial, you will need:

Two Ubuntu servers: one master server with Ansible and one secondary which will run Apache configured through Ansible , Ansible installed on the master server. SSH key exchange to authorize master to login to secondary server. Refer Bash scripting chapter for SSH key exchange.

Manually set up a local hosts file on your local machine (using your secondary server's IP address), in order to set up and use the Virtual Hosts that will be configured.

*Configuring Ansible*

Create a new directory.

$ mkdir ansible-repo

Move into the new directory.

$ cd ~/ansible-repo/

Create a new file called ansible.cfg and open it for editing.

$ vi ansible.cfg

Within that file, we want to add in the hostfile configuration option with the value of hosts, within the[defaults] group. Copy the following into the ansible.cfg file, then save and close it.

[defaults]

inventory = hosts

Create a hosts file(inventory) and open it for editing.

$ vi hosts

Copy the following into the hosts file.

[apache]

apache\_server\_ip ansible\_ssh\_user=username

This specifies a host group called apache which contains one host. Replace apache\_server\_ip with the secondary server's hostname or IP address, and username with your SSH username. Now Ansible should be able to connect to your server.

Test the connectivity with ping module.

$ ansible apache -m ping

The output should look like this:

**1 9 2 . 1 6 8 . 1 . 5 1** success >> {

"changed": false,

"ping": "pong"

}

Another Ansible module that is useful for testing is the command module. It runs custom commands on the host and returns the results. To run the command using echo, a Unix command that echoes a string to the terminal, enter the following command.

$ ansible apache -m command -a "/bin/echo hello devops"

The output should look like this:

1 9 2 . 1 6 8 .1 . 5 1 success rc=0 »

hello devops

*Installing Apache*

We will write the task to install the Apache web server in the playbook.

To install apache via Ansible, we use Ansible's apt module. The apt module contains many options

for specialised apt-get functionality. The options we are interested in are:

• name: The name of the package to be installed, either a single package name or a list of

packages.

• state: Accepts either latest, absent, or present. Latest ensures the latest version is installed,

present simply checks it is installed, and absent removes it if it is installed.

• update\_cache: Updates the cache (via apt-get update) if enabled, to ensure it is up to date.

Now let's create our apache.yml playbook with the apt module. Open up the apache.yml file for

editing.

$ vi apache.yml

Copy the following text into it.

*hosts: apache*

*become: yes*

*tasks:*

*name: install apache2*

*apt: name=apache2 update\_cache=yes state=latest*

The apt line installs the apache2 package (name=apache2) and ensures we have updated the cache (update\_cache=yes).

Now run the playbook.

*$' ansible-playbook apache.yml --ask-sudo-pass*

The ask-sudo-pass flag will prompt you for the sudo password on the secondary server. This is necessary because the installation requires root privileges; the other commands we've run so far did not.

The output should look like this.

PLAY [apache] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GATHERING FACTS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [192.168.1.51]

TASK: [install apache2] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [192.168.1.51]

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

192.168.1.51 : ok=2 changed=l unreachable=0 failed=0

If you visit your secondary server's hostname or IP address in your browser, you should now get a Apache2 Ubuntu Default Page to greet you. This means you have a working Apache installation on your server, and you haven't manually connected to it to run a command yet.

*Configuring Apache Modules*

Now that Apache is installed, we need to enable a module to be used by Apache.

Let us make sure that the mod\_rewrite module is enabled for Apache. Via SSH, this can be done easily by using a2enmod and restarting Apache. However, we can also do it very easily with Ansible using the apache2\_module module and a task handler to restart apache2.

The apache2\_module module takes two options:

•name — The name of the module to enable, such as rewrite.

•state — Either present or absent, depending on if the module needs to be enabled or disabled.

Open apache.yml for editing.

$ vi apache.yml

Update the file to include this task. The file should now look like this:

- *hosts: apache*

*sudo: yes*

*tasks:*

*name: install apache2*

*apt: name=apache2 update cache=yes state=latest*

*name: enabled mod rewrite*

*apache2\_module: name=rewrite state=present*

We need to restart apache2 after the module is enabled. One option is to add in a task to restart apache2, but we don't want that to run every time we apply our playbook. To get around this, we need to use a task handler. The way handlers work is that a task can be told to notify a handler when it has changed, and the handler only runs when the task has been changed.

To do this we need to add the notify option into the apache2\_module task, and then we can use the service module to restart apache2 in a handler.

- *hosts: apache*

*sudo: yes*

*tasks:*

*name: install apache2*

*apt: name=apache2 update\_cache=yes state=latest*

*name: enabled mod rewrite*

*apache2\_module: name=rewrite state=present*

*notify:*

*handlers:*

*name: restart apache2*

*service: name=apache2 state=restarted*

Now, rerun the playbook.

*$ ansible-playbook apache.yml ask-sudo-pass*

The output should look like

*PLAY [apache]*

*GATHERING FACTS*

*ok: [192.168.1.51]*

*TASK: [install apache2]*

*ok: [192.168.1.51]*

*TASK: [enabled mod rewrite]*

*changed: [192.168.1.51]*

*NOTIFIED: [restart apache2] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*changed: [192.168.1.51]*

*PLAY RECAP*

*192.168.1.51*

It looks good so far. Now, run the command again and there should be no changes, and the restart task won’t be listed.

***Configuring Apache Options***

Now that we have a working Apache installation, with our required modules turned on, we need to configure Apache.

By default, Apache listens on port 80 for all HTTP traffic. For the sake of the tutorial, let us assume that we want Apache to listen on port 9090 instead. With the default Apache configuration on Ubuntu x64, there are two files that need to be updated:

/*etc/apache2/ports.conf*

*Listen 80*

*/etc/apache2/sites-available/000-default.conf*

*<VirtualHost \*:80>*

To do this, we can use the lineinfile module. It allows you to perform all sorts of changes to an

existing file on the host. For this example, we will use the following options:

dest — The file to be updated as part of the command.

regexp — Regular Expression to be used to match an existing line to be replaced,

line — The line to be inserted into the file, either replacing the regexp line or as a new line on the end.

state —Either present or absent.

What we need to do to update the port from 80 to 9090 is look for the existing lines which define

port 80, and change them to define port 9090.

Open the apache.yml file for editing.

$ vi apache.yml

- *hosts: apache*

*sudo: yes*

*tasks:*

*name: install apache2*

*apt: name=apache2 update\_cache=yes state=latest*

*name: enabled mod rewrite*

*apache2\_module: name=rewrite state=present*

*notify:*

*name: apache2 listen on port 9090*

*lineinfile: dest=/etc/apache2/ports.conf regexp=*

*9090" state=present*

*rr* A*Listen 80" line="Listen*

*notify:*

*name: apache2 virtualhost on port 9090*

*lineinfile: dest=/etc/apache2/sites-available/000-default.conf*

*regexp= n* **A***<VirtualHost \\*:80>" line="<VirtualHost \*:9090>" state=present*

*notify:*

*- restart apache2*

*handlers:*

*name: restart apache2*

*service: name=apache2 state=restarted*

It is important to notice that we also need to restart apache2 as part of this process, and that we can re-use the same handler but the handler will only be triggered once despite multiple changed tasks.

Now run the playbook.

$ ansible-playbook apache.yml --ask-sudo-pass

Once Ansible has finished, you should be able to visit your host in your browser and it will respond on port 9090, rather than port 80. In most web browsers, this can be easily achieved by adding: port onto the end of the URL: http://192.168.1.51:9090/.

***Configuring Virtual Hosts***

In this section, we will use the template module to configure a new virtual host on your server.

***Create Virtual Host Configuration***

The first step is to create a new virtual host configuration. We'll create the virtual host configuration

file on the master server and upload it to the secondary server using Ansible.

Here's an example of a basic virtual host configuration which we can use as a starting point for our

own configuration. Notice that both the port number and the domain name, highlighted below, are

hardcoded into the configuration.

*<VirtualHost \*:9090>*

*ServerAdmin webmaster@keplar.com*

*ServerName keplar.com*

*ServerAllas www.keplar.com*

*DocumentRoot /var/www/keplar.com*

*ErrorLog ${APACHE\_LOG\_DIR}/error.1og*

*CustomLog ${APACHE\_LOG\_DIR}/access.log combined*

*</VirtualHost>*

Create a new file called virtualhost.conf.

$ vi virtualhost.conf

Paste the following into virtualhost.conf. Because we are using templates, it is a good idea to change the hard-coded values above to variables, to make them easy to change in the future.

*<VirtualHost \*:{{ http\_port }}>*

*ServerAdmin webmaster®{{ domain ))*

*ServerName {{ domain }}*

*ServerAlias www.{{ domain }}*

*DocumentRoot /var/www/{{ domain }}*

*ErrorLog ${APACHE\_LOG\_DIR}/error*.*1og*

*CustomLog ${APACHE LOG DIR}/access.log combined*

*</VirtualHost>*

***Use Template Variables***

Next, we need to update our playbook to push out the template and use the variables. The first step is to add in a section into the playbook for variables. We need to put in both variables used in the template above, and we will change the port back to 80 in the process.

- *hosts: apache*

*sudo: yes*

*vars:*

*http\_port: 80*

*domain: keplar.com*

*tasks:*

*name: install Apache2*

Variables can be used in tasks and templates, so we can update our existing lineinfile modules to use the specified http\_port, rather than the hard coded 9090 we specified before. The variable needs to be added into the line, and the regexp option needs to be updated so it's not looking for a specific port. The changes will look like this:

lineinfile: dest=/etc/apache2/ports.conf regexp

http\_port }}" state=present

Listen " line="Listen {{ \_ I I A

lineinfile: dest=/etc/apache2/sites-available/000-default.conf

regexp=I I A<VirtualHost line="<VirtualHost \*:{{ http\_port }}>"

***Add Template Module***

The next step is to add in the template module to push the configuration file onto the host. We will use these options to make it happen:

32. dest -- The destination file path to save the updated template on the host(s),

i.e./etc/apache2/sites-available/{{ domain }}.conf.

k 33. src — The source template file, i.e. virtualhost.conf.

Applying these to your playbook will result in a task that looks like this:

name: create virtual host file

template:

}}.conf

***Enable the Virtual Host***

What we need to do now is enable the virtual host within Apache. This can be done in two ways: by running the sudo a2ensite keplar.com command or manually symlinking the config file into /etc/apache2/sites-enabled/. The former option is safer, as it allows Apache to control the process.

For this, the command module comes in use again.

The usage is quite simple, as we discovered above:

name: a2ensite {{ domain }}

command: a2ensite {{ domain }}

notify:

restart apache2

***Prevent Extra Work***

Finally, the command module needs to know when it should and shouldn't run, so the module is not run unnecessarily if the playbook is run multiple times. In our case, it only needs to be run if the .conf file hasn't been created on the host yet.

This is done using the creates option, which allows you to tell the module what file is being created during the module execution. If the file exists, the module won't run. Because Apache creates a symlink when sites are enabled, checking for that solves the problem.

The changes will look like this:

name: a2ensite {{ domain }}

command: a2ensite {{ domain }}

args:

creates: /etc/apache2/sites-enabled/{{ domain }}.conf

notify:

- restart apache2

It is important to note the use of the args section in the task. This is an optional way of listing the module options, and in this case, removes any confusion between what is a module option and what is the command itself.

*Pinal apache.yml Playbook*

Now let's apply these changes. Open apache.yml.

$ vi apache.yml

With all of the changes above, change your apache.yml playbook to look like this.

- hosts: apache

sudo: yes

vars:

http\_port: 80

domain: keplar.com

tasks:

name: install apache2

apt: name=apache2 update cache=yes state=latest

name: enabled mod rewrite

apache2\_module: name=rewrite state=present

notify:

name: apache2 listen on port {{ http\_port }}

lineinfile: dest=/etc/apache2/ports.conf regexp II AListen " line=f,Listen {{

http\_port } }

notify:

name: apache2 virtualhost on port {{ http\_port }}

lineinfile: dest=/etc/apache2/sites-available/000-default.conf

regexp=II A<VirtualHost \\*:" line="<VirtualHost \*:{{ http\_port }}>"

notify:

restart apache2

name: create virtual host file

template: src=virtualhost.conf dest=/etc/apache2/sites-available/{{ domain

}}.conf

name: a2ensite {{ domain }}

command: a2ensite {{ domain }}

args:

creates: /etc/apache2/sites-enabled/{{ domain )(.conf

notify:

restart apache2

handlers:

name: restart apache2

service: name=apache2 state=restarted

Save and close the file, then run the playbook.

*$ ansible-playbook apache.yml --ask-sudo-pass*

If you now visit the hostname or IP address of your secondary server in your browser, you will see it responds on port 80 again, not port 9090. Next, visit the domain (i.e. keplar.com) we specified for the new virtual host. Because we haven't added any files in yet, it should show an Apache 404 error page rather than the Apache welcome page. If so, your virtual host is working correctly, and you still haven't SSH'ed into your secondary server to run a single command.

**30. Multi-Tier Web Application Stack Deployment using**

**Ansible.**

This project is about setting up & managing Loadbalancer, application server and database server

with Ansible.

In this project, we will use as many as features of ansible to understand Ansible more in depth.

***Prerequisites***

> Create 5 vm’s with vagrant of ubuntu/trusty64 box.

> Control [Ansible server]

***>*** lbOl

> appOl

> app02

> dbOl

> SSH key exchange should be done from control to all 4 vm’s.

> All vagrant vm’s should have private static IP.

k > Mention host to IP mapping in /etc/hosts file in all the vm’s. As shown below.

' 192.168.1.5 control

192.168.1.6 lbOl

192.168.1.7 appOl

192.168.1.8 app02

192.168.1.9 dbOl

Create directories on your local machines

# *mkdir ansible\_proj*

# *mkdir ansible\_proj/Vms*

# *mkdir ansible proj/control repo*

Place the vagrant file in Vms directory and Spin vagrant vm’s.

# cd learn\_proj/Vms

# *vagrant up*

**Creating Python web application for deployment.**

Please refer below the content of the python web application and create it.

*mkdir ansible\_proj/control\_repo/visualapp*

*mkdir ansible\_proj/control\_repo/visualapp/app*

**cat ansible proj/control repo/visualapp/visualapp.conf**

*<VirtualHost \*>*

*WSGIDaemonProcess visualapp threads=5*

*WSGIScriptAlias / /var/www/visualapp/visualapp.wsgi*

*<Directory /var/www/visualapp>*

*WSGIProcessGroup visualapp*

*WSGIApplicationGroup %{GLOBAL}*

*Order deny,allow*

*Allow from all*

*</Directory>*

*</VirtualHost>*

**cat ansible\_proj/control\_repo/visualapp/app/requirements.txt**

*Flask==0.10.1*

*Flask-SQLAlchemy==2.0*

**cat ansible\_proj/control\_repo/visualapp/app/visualapp.py**

*from flask import Flask*

*from flask.ext.sqlalchemy import SQLAlchemy*

*import os, socket*

*app = Flask( name )*

*app.config[ ’SQLALCHEMY DATABASE URI'] = os.environ['DATABASE URI']*

*db = SQLAlchemy(app)*

*hostname socket.gethostname()*

*@app.route( ’/')*

*def index():*

*return*

*@app.route('/db ’)*

*def dbtest():*

*try:*

*db.create all()*

*except Exception as e:*

*return e.message +*

*return 'Database Connected from %s!\n'*

*if name mam*

*app.run()*

**cat ansible proj/control repo/visualapp/app/visualapp.wsgi**

*activate this* '*/var/www/visualapp/.venv/bin/activate\_this.py'*

*execfHe(activate this, diet( file =activate this))*

*import os*

*os.environ['DATABASE URI ’] ’mysql://visualapp:visualappddbOl/visualapp'*

*import sys*

ss*.path.insert(0, '/var/www/visualapp ’)*

*from visualapp import app as application*

**Ansible installation**

Ansible package needs to be installed on control machine

Refer ansible doc

http://docs.ansible.eom/ansible/intro\_installation.html#latest-releases-via-apt-ubuntu

**Verify installation**

# ansible --help

# ansible-playbook -help

# ansible-galaxy -help

**Setting up ansible repo**

Login to control server

# vagrant ssh control

# mkdir -p /home/vagrant/repo/ansible

# cd /home/vagrant/repo/ansible

# vi dev (inventory)

+[loadbalancer]

+Ib01

+[webserver]

+app01

+app02

+[database]

+db01

+[control]

+control ansible connection=local

**Ansible.cfg**

Set inventory path in ansible.cfg

Benefit of tracking it in version control system

#>cd /home/vagrant/repo/ansible

#' vi ansible.cfg

+[defaults]

+inventory = ./dev

# ansible --list-hosts all

Validate the inventory and connections

# ansible -list-hosts all

Can use pattern syntax that allow you to select subset from the inventory.

# ansible -list-hosts all

# ansible -list-hosts

# ansible -list-hosts loadbalancer

# ansible -list-hosts Webserver

# ansible -list-hosts db01

# ansible -list-hosts database:control

# ansible -list-hosts webserver[0]

# ansible -list-hosts Vcontrol

https://docs.ansible.com/ansible/intro\_patterns.html

https://docs.ansible.eom/ansible/intro\_getting\_started.html#your-first-commands

https://docs.ansible.com/ansible/ping\_module.html

https://docs.ansible.com/ansible/command\_module.html

# ansible -m ping all

# ansible -m command -a “hostname” all

**Default module is command**

# ansible -a “hostname” all

All tasks are going to have return status

Return exit code non-zero

# ansible -a “/bin/false” all

http://docs.ansible.com/ansible/modules\_by\_category.html

Plays

# cd /home/vagrant/repo/ansible

# vi hostname.yml

+ —

+ - hosts: all

+ tasks:

**+** - command: hostname

Playbook Execution

# ansible-playbook hostname.yml

# vi hostname.yml

- hosts: all

tasks:

- name : get server info

- command: hostname

command: hostname

# ansible-playbook hostname.yml

**Four pillars of Linux application | Principles to setup app on Linux**

1. Packages’: From repositories (apt or yum) or any other resources

2. Services: init.d or system.d or your own start script

3. System configurations: Files, directories, users, permission, firewall rules etc

4. Config files for the app itself

Playbook intro

http://docs.ansible.com/ansible/modules\_by\_category.html

Packages

Creating playbook for installing nginx package on loadbalancer

# vi loadbalancer.yml

- hosts: loadbalancer

become: true

tasks:

- name: Install nginx

apt: name=nginx state=present update\_cache=yes

Creating playbook for installing mysql-server package on database server

# vi database.yml

- hosts: database

become: true

tasks:

- name: Install mysql-server

apt: name=mysql-server state=present update\_cache=yes

Execute loadbalancer playbook

#‘ansible-playbook loadbalancer.yml

Execute it again

# ansible-playbook loadbalancer.yml

Execute Database playbook

# ansible-playbook database.yml

Creating Webserver playbook, which covers loops and jinja2 templates

https://docs.ansible.eom/ansible/playbooks\_loops.html#standard-loops

http://jinja.pocoo.org/

# vi Webserver.yml

- hosts: Webserver

become: true

tasks:

- name: Install apache2

apt: name={{item}} state=present update\_cache=yes

with items:

- apache2

- libapache2-mod-wsgi

- python-pip

- python-virtualenv

Services modules

# vi loadbalancer.yml

**+**- name: Ensure nginx started

+ service: name=nginx state=started enabled=yes

Test the nginx service

# wget -qO- http://lb01 | less

Easier to do with curl, so install curl on the control server with playbook

# vi control.yml

- hosts: control

become: true

tasks:

- name: install tools

apt: name={{item}} state=present update\_cache=yes

withjtems:

k - curl

# ansible-playbook control.yml

# curl Ib01

Add service module to Webserver and database playbooks

# vi Webserver.yml

**+** - name: Ensure Apache started

+ service: name=apache2 state=started enabled=yes

# vi database.yml

+ - name: Ensure Mysql started

+ service: name=mysql state=started enabled=yes

# $ ansible-playbook webserver.yml

# ansible-playbook database.yml

**apache2\_module, handlers, notify**

We need to setup the apache receiving python application. Python app going to use mod-wsgi to

serve the request. Make sure mod-wsgi is enabled with apache\_module.

https://docs.ansible.com/ansible/apache2\_module\_module.html

https://docs.ansible.eom/ansible/playbooks\_intro.html#handlers-running-operations-on-change

# ansible-playbook webserver.yml

- name: Ensure mod\_wsgi enables

apache2\_module: state=present name=wsgi

notify: Restart apache2

+ handlers:

- name: Restart apache2

service: name=apache2 staterestarted

# ansible-playbook webserver.yml

**Files: Copy**

Copy the visualapp directory in the ansible directory which contains the python app written in flask.

Use copy module to ship application folder from control server to app01/02 servers.

Our visualapp site file also needs to be copied in /etc/apache2/sites-available directory.

We will enable out visualapp website in next section.

# vi Webserver.yml

- name: copy visualapp app source

copy: src=visualapp/app/ dest=/var/www/visualapp mode=0755

notify: Restart apache

- name: copy apache virtual host config

copy: src=visualapp/visualapp.conf dest=/etc/apache2/sites-available mode=0755

notify: Restart apache

# ansible-playbook webserver.yml

# curl app01

Still shows the default apache site for python app working we need to configure PIP and Virtualenv

**Application Modules**

To run the python flask application, we need to install flask & SQLalchemy python package. To install python packages, we will use ‘pip’ which is a python package manager. But we will not install the python packages directly on the system, we will create a virtual environment/container known as virtulenv which will hold all our python packages.

# vi webserver.yml

- name: setup python virtualenv

pip: requirements=/var/www/visualapp/requirements.txt virtualenv=/var/www/visualapp/.venv

notify: Restart apache2

# ansible-playbook webserver.yml

**Files: File | Activating python site and deactivating apache default site**

By default, apache will serve a website, also known as apache default page.

We will disable the default website and enable our visualapp website.

To disable default website, we need to unlink 000-default.conf file located in

sites-enabled directory. Next, we will create a link from /etc/apache2/sites-available/visualapp.conf

to /etc/apache2/sites-enabled/visualapp.conf.

# vi webserver.yml

**+** - name: de-activate default apache site

**+** file: path=/etc/apache2/sites-enabled/000-default.conf state=absent

**+** notify: Restart apache

+ - name: activate visualapp apache site

+ file: src=/etc/apache2/sites-available/visualapp.conf dest=/etc/apache2/sitesenabled/visualapp.conf state=link

+ notify: Restart apache

# ansible-playbook webserver.yml

# curl app01

# curl app02

# curl Ib01, will show the default page from nginx, we need to configure it to point to the App servers

**Templates | Configure Ib01 to point to app01 & app02**

By default, nginx will serve a website, also known as nginx default page. We will disable the default website and enable a redirect rule which will forward request from nginx to our app servers randomly.

To disable default website, we need to unlink default file located in /etc/nginx/sites-enabled/directory.

Next, we will create a template templates/nginx.conf.j2 as mentioned below and push that template

to /etc/nginx/sites-available/visualapp location on Ib01.

Once we push visualapp configuration file in site-available directory we can create a link from /etc/nginx/sites-enabled/visualapp to /etc/nginx/sites-available/visualapp which will enable the nginx site.

Nginx site is a redirect rule which will forward the requests to app01 & app02 randomly

# curl Ib01

# mkdir templates

# vi templates/nginx.conf.j2

+upstream visualapp {

+{% for server in groups.webserver %}

server {{ server }};

+{% endfor %}

**+**

+}

-rserver {

+ listen 80;

+ location / {

**+** proxy\_pass http://visualapp;

+ }

+}

# vi loadbalancer.yml

**+** - name: configure nginx site

**+** template: src=templates/nginx.conf.j2 dest=/etc/nginx/sites-available/visualapp mode=0644

**+** notify: restart nginx

**+** - name: de-activate default nginx site

file: path=/etc/nginx/sites-enabled/default state=absent

notify: restart nginx

- name: activate visualapp nginx site

file: src=/etc/nginx/sites-available/visualapp dest=/etc/nginx/sites-enabled/visualapp state=link

notify: restart nginx

+k handlers:

- name: restart nginx

service: name=nginx state=restarted

# ansible-playbook loadbalancer.yml

Test if it’s working.

# curl Ib01

**Lineinfile | Make db server to listen to on all interface O.O.O.O.**

https://docs.ansible.com/ansible/lineinfile\_module.html

Test the connection first

# curl app01/db

- Check visualapp.wsgi which has db connection info.

- Check visualapp.py which establishes connection from app to db server

- Login to db server and see mysql listening only on local interface

# netstat -an

- Open /etc/mysql/my.cnf

With ansible lineinfile module replace “bind-address 127.0.0.1” to bind-address to “bind-address

O.O.O.O”

Login to control server

# vi database.yml

+ - name: ensure mysql listening on all ports

+ lineinfile: dest=/etc/mysql/my.cnf regexp=Abind-address

+ line="bind-address = O.O.O.O"

**+** notify: restart mysql

+ handlers:

+- name: restart mysql

+ service: name=mysql state=restarted

# ansible-playbook database.yml

**Application Modules: mysql\_db, mysql\_user | Install python-mysqldb in app and db server,**

**create visualapp db and visualapp user**

Install mysql module in apache server

# vi Webserver.yml

- name: Install apache2

apt: name={{item}} state=present update\_cache=yes

withjtems:

- apache2

- libapache2-mod-wsgi

- python-pip

- python-virtualenv

+ - python-mysqldb

# ansible-playbook webserver.yml

https://docs.ansible.com/ansible/mysql\_db\_module.html

https://docs.ansible.com/ansible/mysql\_user\_module.html

# vi database.yml

- hosts: database

become: true

tasks:

- name: install tools

apt: name={{item}} state=present update\_cache=yes

withjtems:

- python-mysqldb

- name: install mysql-server

apt: name=mysql-server state=present update\_cache=yes

lineinfile: dest=/etc/mysql/my.cnf regexp=Abind-address line="bind-address = 0.0.0.0"

notify: restart mysql

- name: create visualapp database

mysql\_db: name=visualapp state=present

**+** - name: create visualapp user

mysql\_user: name=visualapp password=visualapp priv=visualapp.\*:ALL host='%' state=present

handlers:

- name: restart mysql

service: name=mysql state=restarted

Execute db playbook

# ansible-playbook database.yml

Test the connection

# curl app01/db

# curl app02/db

# curl Ib01/db

# curl Ib01/db

# curl Ib01/db

# curl Ib01/db

**Roles:**

We have three tiers in our app, for every tier we have a separate playbook.

If we have another team, they have their application which they want to manage with ansible.

How much of our existing code is reusable to our new team?

We can copy all our playbooks paste them and the go through each line and edit every place we

put visualapp and replace that with new application detail.

That is very tedious and especially we have more application coming down the line.

Also, if they have different config like they want to use different ports.

If we want to make changes to our database configuration, we got many places in all our

playbooks that require changes. Lot of code duplication there.

Ansible gives us Roles to solve such problems. Essentially, we need to create a folder for every

role which will have sub folders like tasks, files, handlers, vars etc, which holds all the information

separately.

For example, all the tasks that we have written in our playbooks goes into tasks folder. Now

instead of having all the configuration for our tiers mashed up in one playbook we can distribute it

into multiple folders or files.

Encapsulation

Code Reusability

Scalability

Ansible galaxy roles

# mkdir -p /home/vagrant/repo/ansible-roles

# cp -r /home/vagrant/repo/ansible/\* /home/vagrant/repo/ansible-roles/

# cd /home/vagrant/repo/ansible-roles

# mkdir roles

# cd roles

Create skeleton of roles with ansible-galaxy

# ansible-galaxy init control

# ansible-galaxy init mysql

# ansible-galaxy init nginx

# ansible-galaxy init apache2

# ansible-galaxy init visualapp\_app

Need to move content of all the playbooks to their respective roles

**1. control.yml:**

Move lists of tasks to control role’s tasks/main.yml

Replace tasks section from the control playbook with roles content.

# vi control.yml

- hosts: control

become: true

roles:

- control

Execute control.yml to test.

# ansible-playbook control.yml

2**. database.yml:**

Move lists of tasks to mysql role’s tasks/main.yml.

Move mysql start task after my.cnf file change section.

Move lists of handlers to mysql role’s handlers/main.yml.

Replace tasks & handlers section from the playbook with roles content.

- hosts: database

become: true

roles:

- mysql

3. **loadbalancer.yml**

Move lists of tasks to nginx role’s tasks/main.yml.

Move lists of handlers to nginx role’s handlers/main.yml.

Move nginx.conf.j2 template to nginx role’s templates directory

Open tasks/main.yml of the nginx role, change templates src path

from src= templates/nginx.conf.j2 to src=nginx.conf.j2

Replace tasks & handlers section from the playbook with roles content.

- hosts: loadbalancer

become: true

roles:

- nginx

**4. webserver.yml**

Move lists of apache tasks to apache role’s tasks/main.yml.

Move apache2 start task to the end of the.

Move lists of apache handlers to apache role’s handlers/main.yml.

‘ Move lists of app tasks to visualapp\_app role’s tasks/main.yml.

Move lists of app handlers to visualapp\_app role’s handlers/main.yml.

Copy visualapp app directory to visualapp\_app role’s files/

Replace tasks & handlers section from the playbook with roles content.

- hosts: Webserver

become: true

roles:

- apache2

- visualapp\_app

**Site.yml: include | include all the playbook into site.yml**

Create site.yml at the same place where all the playbooks are located

# vi site.yml

- include: control.yml

- include: database.yml

- include: webserver.yml

- include: loadbalancer.yml

# ansible-playboook site.yml

**Variables: facts | Will use fact variable to make mysql listen only on its own IP address and**

**not on 0.0.0.0**

Get the list of all the fact variable with setup module

# ansible -m setup database

We will use ansible\_eth0.ipv4.address fact variable in our lineinfile module for the bind-address

instead of 0.0.0.0

Edit database role’s tasks

# vi roles/mysql/tasks/main.yml

- name: ensure mysql listening on all ports

- lineinfile: dest=/etc/mysql/my.cnf regexp=Abind-address line="bind-address = 0.0.0.0"

+ lineinfile: dest=/etc/mysql/my.cnf regexp=Abind-address

line="bind-address = {{ ansible\_eth0.ipv4.address }}"

notify: restart mysql

**+** - name: ensure mysql started

Execute stack\_status.yml

# ansible-playbook stack\_status.yml

We also need to update the waitjor module in stack\_status playbook.

By default, it will wait for the loopback address 127.0.0.1. Since we updated mysql config to only

listen on its ipaddress we need to make change in stack\_status playbook as well.

Update stack\_status.yml

# vi playbooks/stack\_status.yml

- name: verify mysql is listening on 3306

wait\_for: port=3306 timeout=1

wait\_for: host={{ **+** ansible\_eth1,ipv4.address }} port=3306 timeout=1

Also update waitjor module in stack restart playbook.

Update stack\_restart.yml

# vi playbooks/stack\_restart.yml

become: true

tasks:

- service: name=mysql state=restarted

- waitjor: port=3306 state=started

**+** - waitjor: host={{ ansible\_eth1,ipv4.address }} port=3306 state=started

**Variables: defaults | Using variable in mysql\_db & mysql user module**

We have hard coded db config parameter like user, password, dbname etc in the playbook which

is not modular, if we want to change a parameter then we need to go over the entire playbook and

change every place where we mentioned that parameter like db name.

We will use Role’s defaults directory and put all our db config info into variables. We can then refer

to these variables in our playbook.

Lowest priorities are for default vars, refer the doc.

https://docs.ansible.com/ansible/playbooks\_variables.html#variable-precedence-where-should-iput-

a-variable

rdles/mysql/defaults/main.yml

-# defaults file for mysql

+db\_name: myapp

+db\_user\_name: dbuser

+db\_user\_pass: dbpass

+db user host: localhost

roles/mysql/tasks/main.yml

- name: ensure mysql started

service: name=mysql state=started enabled=yes

-- name: create visualapp database

- mysql\_db: name=visualapp state=present

+- name: create database

+ mysql\_db: name={{ db\_name }} state=present

-- name: create visualapp user

- mysql\_user: name=visualapp password=visualapp priv=visualapp.\*:ALL host='%' state=present

+- name: create user

+ mysql\_user: name={{ db\_user\_name }} password={{ db\_user\_pass }} priv={{ db\_name }}.\*:ALL

**+** host='{{ db\_user\_host }}' state=present

**Variables: Vars | Multiple ways/places to define vars, refer the doc.**

https://docs.ansible.com/ansible/playbooks\_variables.html#variable-precedence-where-should-iput-

a-variable

Already defined vars in mysql role’s defaults/main.yml which has the last precedence and would be

used if vars are nowhere defined. We can override these variables by defining them while referring

to the role in our database.yml playbook

We will define vars in database.yml playbook with a hash.

In this case variable defined in below playbook will have higher precedence than our default

variable defined in above section.

# vi database.yml

- hosts: database

become: true

roles:

- - mysql

- { role: mysql, db\_name: visualapp, db\_user\_name: visualapp, db\_user\_pass: visualapp,

db\_user\_host: '%' }

Variables: with dict | Use of with\_dict module in nginx.yml playbook.

We can also provide variables to playbooks in list format as mentioned below.

Will provide lists of variables in nginx role’s defaults/main.yml. As specified below for reference.

Name of the list is “sites”.

# defaults file for nginx

sites:

myapp:

frontend: 80

backend: 80

Will refer to those variables with the module with dict in nginx.yml playbook. As specified below for

reference,

with diet: sites

with\_dict will supply variable name in hash/dictionary format, variable names are {{ item.key }} & {{

item.value.frontend }} & {{ item.value.backend }}

(item={'value': {u'frontend': 80, u'backend': 80}, ’key': u'myapp'})

Update nginx role’s defaults/main.yml

+sites:

+ myapp:

+ frontend: 80

+ backend: 80

Update nginx role’s tasks/main.yml

- name: install nginx

apt: name=nginx state=present update\_cache=yes

-- name: configure nginx site

- template: src=nginx.conf.j2 dest=/etc/nginx/sites-available/visualapp mode=0644

+- name: configure nginx sites

+ template: src=nginx.conf.j2 dest=/etc/nginx/sites-available/{{ item.key }} mode=0644

+ with dict: sites

notify: restart nginx

- name: de-activate default nginx site

file: path=/etc/nginx/sites-enabled/default state=absent

notify: restart nginx

-- name: activate visualapp nginx site

- file: src=/etc/nginx/sites-available/visualapp dest=/etc/nginx/sites-enabled/visualapp state=link

+- name: activate nginx sites

+ file: src=/etc/nginx/sites-available/{{ item.key }} dest=/etc/nginx/sites-enabled/{{ item.key }}

state=link

+ with\_dict: sites

notify: restart nginx

Update nginx role’s templates/nginx.conf.j2

-upstream visualapp {

-i-upstream {{ item.key }} {

{% for server in groups.webserver %}

- server {{ server }};

server {{ server }}:{{ item.value.backend }};

{% endfor %}

**+**

}

server {

- listen 80;

+ listen {{ item.value.frontend }};

location / {

proxy\_pass http://visualapp;

**+** proxy\_pass http://{{ item.key }};

}

}

**Variables continued | will push visualapp.wsgi as a template from visualapp\_app role**

We have replaced hardcoded DB values in our playbook as variables.

We also have db config parameters hardcoded in visualapp.wsgi file from our python flask app.

We will push visualapp.wsgi as a template and put all our db variables into that template.

Move visualapp.wsgi to visualapp app role’s templates directory

# mv roles/visualapp\_app/files/visualapp/app/visualapp.wsgi

roles/visualapp\_app/templates/visualapp.wsgi.j2

Update the visualapp.wsgi.j2 with db variables names.

activatejhis = 7var/www/visualapp/.venv/bin/activate\_this.py'

execfile(activate\_this, dict( file =activate\_this))

import os

+os.environ['DATABASE\_UFH'] = 'mysql://{{ db\_user }}:{{ db\_pass }}@{{ groups.database[0] }}/{{

db\_name }}'

import sys

sys.path.insert(0, '/var/www/visualapp')

from visualapp import app as application

Refer the template visualapp.wsgi.j2 in visualapp\_app role’s tasks/main.yml

# vi roles/visualapp\_app/tasks/main.yml

copy: src=visualapp/app/ dest=/var/www/visualapp mode=0755

‘notify: restart apache2

+- name: copy visualapp.wsgi

+ template: src=visualapp.wsgi.j2 dest=/var/www/visualapp/visualapp.wsgi mode=0755

+ notify: restart apache2

**+** - name: copy apache virtual host config

copy: src=visualapp/visualapp.conf dest=/etc/apache2/sites-available mode=0755

notify: restart apache2

Put the variables name in webserver.yml playbook where we refer the nginx role

# vi webserver.yml

become: true

roles:

- apache2

- - visualapp\_app

- { role: visualapp\_**+** app, db\_user: visualapp, db\_pass: visualapp, db\_name: visualapp }

# ansible-playbook webserver.yml

**Variables: vars\_files, group\_vars | Use group\_vars/all file to specify variables for database**

Database variables are referred in two places

visualapp.wsgi.j2 template from visualapp\_app role &

tasks/main.yml from mysql role.

We will use group\_vars/all file to specify below variables

db\_name: visualapp

db\_user: visualapp

db\_pass: visualapp

# mkdir group\_vars

# vi group\_vars/all

+—

+db\_name: visualapp

+db\_user: visualapp

+db\_pass: visualapp

Remove db variables from websever.yml playbook

# vi Webserver.yml

become: true

roles:

- apache2

- { role: visualapp\_app, db\_user: visualapp, db\_pass: visualapp, db\_name: visualapp }

**+** - visualapp\_app

Refer db variables from group\_vars/all file inside database.yml playbook also called as variable

routing.

Variable routing can be defined as one variable pointing to another variable.

E:g db\_user\_name: “{{ db\_user }}”

# vi database.yml

- hosts: database

become: true

roles:

- - { role: mysql, db\_name: visualapp, db\_user\_name: visualapp, db\_user\_pass: visualapp,

db\_user\_host: '%' }

- role: mysql

**+** - role: mysql

**+** db\_user\_name: "{{ db\_user }}"

+ db\_user\_pass: "{{ db\_pass }}"

db user host: '%'

**Variables: vautl | encrypting db\_pass variables value with vault**

https://docs.ansible.com/ansible/Dlavbooks vault.html

Instead of having all file for variables, we will create all directory and move our variables in all

directory.

# cd group\_vars

# mv all vars

# mkdir all

# mv vars all

# export EDITOR=vim

Change directory to all directory and create vault. Vault file should be present in all directory for our

scenario.

# cd all

# ansible-vault create vault

Give a vault password and put below mentioned content.

+ —

+ vault\_db\_pass : admin123

Refer to vault\_db\_pass in all/vars file

# vi vars

+—

+db\_name: visualapp

+db\_user: visualapp

+db\_pass: "{{ vault\_db\_pass }}"

Execute database.yml playbook, you should get some error like below

# ansible-playbook database.yml

ERROR! Decryption failed

ERROR! A vault password must be specified

Either you can give -ask-vault-pass option and execute the playbook which will ask you the

vaultpassword or you can use a file where you specify vault password.

“vaultpass” is our vault password.

# echo "vaultpass" > ~Avault\_pass.txt

# chmod 0600 ~7.vault\_pass.txt

# vi ansible.cfg

[defaults]

inventory = ./dev

+vault\_password\_file = ~Avault\_pass.txt

Execute database playbook to test it

# ansible-playbook database.yml

Login to db01 instance and login to mysql database to verify.

# ssh db01

# mysql -h localhost -u visualapp -p

**Advanced Execution**

**Advanced Execution: gatherjacts | By disabling gatherjacts we can save the execution time.**

Time the execution with time command for site.yml, stack\_status and stack\_restart.yml

# time ansible-plabook site.yml

# time ansible-plabook stack\_status.yml

# time ansible-plabook stack\_restart.yml

Disable gatherjacts for all the plays except dbserver because we use fact varible for dbservers play.

ansible\_eth1,ipv4.address

control.yml

- hosts: control

become: true

+ gatherjacts: false

roles:

- control

loadbalancer.yml

- hosts: loadbalancer

become: true

+ gatherjacts: false

roles:

- nginx

hostname.yml

- hosts: all

gatherjacts: false

tasks:

- name: get server hostname

command: hostname

**+**

stack\_restart.yml

# Bring stack down

- hosts: loadbalancer

become: true

+ gatherjacts: false

tasks:

- service: name=nginx state=stopped

- wait\_for: port=80 state=drained

- hosts: Webserver

become: true

+ gather\_facts: false

‘tasks: i

- service: name=apache2 state=stopped

- wait\_for: port=80 state=stopped

# Bring stack up

- hosts: Webserver

become: true

+ gather\_facts: false

tasks:

- service: name=apache2 state=started

- wait\_for: port=80

- hosts: loadbalancer

become: true

+ gather\_facts: false

tasks:

- service: name=nginx state=started

- wait\_for: port=80

stack\_status.yml

- hosts: loadbalancer

become: true

+ gather\_facts: false

tasks:

- name: verify nginx service

command: service nginx status

- hosts: Webserver

become: true

+ gather\_facts: false

tasks:

- name: verify apache2 service

command: service apache2 status

wait\_for: host={{ ansible\_eth0.ipv4.address }} port=3306 timeout=1

- hosts: control

+ gather\_facts: false

tasks:

- name: verify end-to-end index response

uri: url=http://{{item}} return\_content=yes

withjtems: "{{lb\_db.results}}"

- hosts: loadbalancer

+ gather\_facts: false

tasks:

k - name: verify backend index response

uri: url=http://{{item}} return\_content=yes

Webserver.yml

- hosts: Webserver

become: true

+ gather\_facts: false

roles:

- apache2

Time the execution again with time command to verify.

**Extracting Repetitive Tasks: cache\_valid\_time**

Open site.yml and add below mentioned play

+- hosts: all

+ become: true

+ gather\_facts: false

+ tasks:

- name: update apt cache

apt: update\_cache=yes cache\_valid\_time=86400

**+** - include: control.yml

+ - include: database.yml

+ - include: webserver.yml

Remove update\_cache=yes parameter from all the tasks/main.yml of all the roles.

# vi roles/mysql/tasks/main.yml

E:g

- apt: name=mysql-server state=present update\_cache=yes

+ apt: name=mysql-server state=present

**Limiting Execution by Hosts : limit**

# ansible-playbook site.yml -limit app01

**Limiting Execution by Tasks : tags**

We can also select particular task or tasks by tagging them and then using tag name while

executing playbooks or site.yml.

# vi roles/control/tasks/main.yml

withjtems:

- apache2

- libapache2-mod-wsgi

+ tags: [ 'packages' ]

List the available tags.

# $ ansible-playbook site.yml -list-tags

Execute it to verify

# ansible-playbook site.yml -tags “packages”

We can also skip the specific tags and execute the rest.

# $ ansible-playbook site.yml -skip-tags “packages”

Tag all the tasks as per our 4 principles of app deployment on Linux systems

[‘packages']

[‘service’]

[‘system’]

[‘configure’]

# vi roles/apache2/tasks/main.yml

withjtems:

- apache2

- libapache2-mod-wsgi

+ tags: [ 'packages' ]

- name: ensure mod\_wsgi enabled

apache2\_module: state=present name=wsgi

notify: restart apache2

+ tags: [ 'system' ]

- name: de-activate default apache site

file: path=/etc/apache2/sites-enabled/000-default.conf state=absent

notify: restart apache2

+ tags: [ 'system' ]

- name: ensure apache2 started

service: name=apache2 state=started enabled=yes

+ tags: [ 'service' ]

# vi roles/control/tasks/main.yml

withjtems:

- curl

\* - python-httplib2

+ tags: [ 'packages' ]

# vi roles/visualapp\_app/tasks/main.yml

- python-pip

- python-virtualenv

- python-mysqldb

+ tags: [ 'packages' ]

- name: copy visualapp app source

copy: src=visualapp/app/ dest=/var/www/visualapp mode=0755

notify: restart apache2

+ tags: [ 'configure' ]

- name: copy visualapp.wsgi

template: src=visualapp.wsgi.j2 dest=/var/www/visualapp/visualapp.wsgi mode=0755

notify: restart apache2

+ tags: [ 'configure' ]

- name: copy apache virtual host config

copy: src=visualapp/visualapp.conf dest=/etc/apache2/sites-available mode=0755

notify: restart apache2

+ tags: [ 'configure' ]

- name: setup python virtualenv

pip: requirements=/var/www/visualapp/requirements.txt virtualenv=/var/www/visualapp/.venv

notify: restart apache2

+ tags: [ 'system' ]

- name: activate visualapp apache site

file: src=/etc/apache2/sites-available/visualapp.conf dest=/etc/apache2/sitesenabled/

visualapp.conf state=link

notify: restart apache2

+ tags: [ 'configure' ]

# vi roles/mysql/tasks/main.yml

apt: name={{item}} state=present

withjtems:

- python-mysqldb

+ tags: [ 'packages' ]

- name: install mysql-server

apt: name=mysql-server state=present

+ tags: [ 'packages' ]

- name: ensure mysql listening on all ports

lineinfile: dest=/etc/mysql/my.cnf regexp=Abind-address

line="bind-address = {{ ansible\_eth0.ipv4.address }}"

‘notify: restart mysql

+ tags: [ 'configure' ]

- name: ensure mysql started

service: name=mysql state=started enabled=yes

+ tags: [ 'service' ]

- name: create database

mysql\_db: name={{ db\_name }} state=present

+ tags: [ 'configure' ]

- name: create user

mysql\_user: name={{ db\_user\_name }} password={{ db\_user\_pass }} priv={{ db\_name }}.\*:ALL

host='{{ db\_user\_host }}' state=present

+ tags: [ 'configure' ]

# vi roles/nginx/tasks/main.yml

apt: name={{item}} state=present

withjtems:

- python-httplib2

+ tags: [ 'packages' ]

- name: install nginx

apt: name=nginx state=present

+ tags: [ 'packages' ]

- name: configure nginx sites

template: src=nginx.conf.j2 dest=/etc/nginx/sites-available/{{ item.key }} mode=0644

with dict: sites

notify: restart nginx

+ tags: [ 'configure' ]

- name: get active sites

shell: Is -1 /etc/nginx/sites-enabled

register: active

+ tags: [ 'configure' ]

- name: de-activate sites

file: path=/etc/nginx/sites-enabled/{{ item }} state=absent

withjtems: active.stdoutjines

when: item not in sites

notify: restart nginx

+ tags: [ 'configure' ]

- name: activate nginx sites

file: src=/etc/nginx/sites-available/{{ item.key }} dest=/etc/nginx/sites-enabled/{{ item.key }}

state=link

with diet: sites

notify: restart nginx

+k tags: [ 'configure' ]

- name: ensure nginx started

service: name=nginx state=started enabled=yes

+ tags: [ 'service' ]

# vi site.yml

tasks:

- name: update apt cache

apt: update\_cache=yes cache\_valid\_time=86400

**+** tags: [ 'packages' ]

- include: control.yml

- include: database.yml.

Save execution time by skipping packages tags

# time $ ansible-playbook site.yml --skip-tags “packages”

**Troubleshooting, Testing & Validation ordering problem**

Make sure to have mysql service start task after you changed the mysql configuration. If there is some typo or bad config in my.cnf, notify mysql restart is going to stop mysql and while starting it will fail. If you run the playbook again first it will hit mysql start task which will fail to start the mysql service and your play is not going to reach to the task where it pushes new my.cnf file to fix the issue. Ordering is very important in such scenario.

- name: Install mysql-server

apt: name=mysql-server state=present

- name: ensure mysql listening on all ports

lineinfile: dest=/etc/mysql/my.cnf regexp=Abind-address

line="bind-address = {{ansible\_eth1,ipv4.address}}"

notify: restart mysql

- name: Ensure Mysql started

service: name=mysql state=started enabled=yes

**Jumping to Specific tasks: list-tasks, step, start-at-task**

https://docs.ansible.com/ansible/playbooks\_startnstep.html

# ansible-playbook site.yml -step

Select y for yes, n for no and c to continue with the execution without asking the prompt.

List all the tasks from the playbook and select the task from where you want to start the execution.

# ansible-playbook site.yml -list-tasks

# ansible-playbook site.yml -start-at-task “copy visualapp app source”

**Retrying failed hosts**

# ansible-playbook site.yml -limit @/home/ansible/site.retry

**Syntax-Check & Dry-Run: syntax-check, check**

***# ansible-playbook -syntax-check site.yml***

# ansible-playbook —check site.yml

**31. WordPress setup with Ansible**

We are going to learn how to automate the deployment of a LAMP stack and install WordPress. LAMP stands for Linux, Apache (a Web server), MySQL (a database) and PHP (server-side scripting). It is a technology stack on which you can deploy different Web applications. We are also going to explore the installation of WordPress, which is free and open source software for creating websites and blogs.

**Assumptions:**

• Ansible installed on the server and target node is a ubuntu server.

• Inventory created with target server hostname/IP and credentials setup.

• ansible.cfg with inventory path host\_key\_checking taken care.

**Apache setup Playbook**

Installs apache after apt update, starts apache service and wait for port 80 to come up.

- name: Install Apache web server

hosts: ubuntu

become: yes

become\_method: sudo

gather\_facts: true

tags: [web]

tasks:

- name: Update the software package repository

apt:

update\_cache: yes

- name: Install Apache

package:

name: "{{ item }}"

state: latest

with\_items:

- apache2

- wait\_for:

port: 80

**Execute playbook**

$ ansible-playbook apache.yml -K

SUDO password:

PLAY [Install Apache web server] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [setup]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [ubuntu]

TASK [Update the software package repository] \*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [ubuntu]

TASK [Install Apache]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [ubuntu] => (item=[u'apache2'])

TASK [wait\_for] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [ubuntu]

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

ubuntu : ok=4 changed=2 unreachable=0 failed=0

**Installing MySQL**

Install the MySQL database server.

- name: Install MySQL database server

hosts: ubuntu

become: yes

become\_method: sudo

gather\_facts: true

tags: [database]

tasks:

name: Update the software package repository

apt:

update\_cache: yes

- name: Install MySQL

package:

name: "{{ item }}"

state: latest

with\_items:

- mysql-server

- mysql-client

- python-mysqldb

- name: Start the server

service:

name: mysql

state: started

- wait\_for:

port: 3306

- mysql\_user:

name: guest

password: '

encrypted: yes

priv:

state: present

» \*.\*:ALL,GRANT

**Execute Mysql Playbook.**

$ ansible-playbook mysql.yml -K

SUDO password:

PLAY [Install MySQL database server]

TASK [setup]

ok: [ubuntu]

TASK [Update the software package repository]

changed: [ubuntu]

TASK [Install MySQL]

changed: [ubuntu] => (item=[u'mysql-server', u 1 mysql-client', u'pythonmysqldb'])

TASK [Start the server]

ok: [ubuntu]

TASK [wait\_for]

ok: [ubuntu]

TASK [mysql\_user]

ok: [ubuntu]

PLAY RECAP

ubuntu : ok=6 changed=2 unreachable=0 failed=0

**Installing PHP**

PHP is a server-side programming language

Installing PHP

name: Install PHP

hosts: ubuntu

become: yes

become\_method: sudo

gather\_facts: true

tags: [web]

tasks:

- name: Update the software package repository

apt:

update\_cache: yes

- name: Install PHP

package:

name: "{{ item }}"

state: latest

with\_items:

- php5

- php5-mysql

$ ansible-playbook php.yml -K

SUDO password:

PLAY [Install PHP]

TASK [setup]

ok: [ubuntu]

TASK [Update the software package repository]

changed: [ubuntu]

TASK [Install PHP]

changed: [ubuntu] => (item=[u'php5', u'php5-mysql'])

PLAY RECAP

ubuntu : ok=3 changed=2 unreachable=0 failed=0

**Installing WordPress**

Install and set up WordPress.

- name: Setup WordPress

hosts: ubuntu

become: yes

become\_method: sudo

gather facts: true

tags: [database]

vars:

wordpress\_file: "/home/({ ansible\_user }}/Downloads/wordpress-latest.zip"

wordpress\_dest: "/var/www/html"

tasks:

- name: Update the software package repository

apt:

update cache: yes

- name: Create a database for wordpress

mysql\_db:

name: wordpress

state: present

- name: Create downloads directory

file:

path: "/home/{{ ansible\_user }}/Downloads"

state: directory

- name: Create target directory

file:

path: "{{ wordpress\_dest }}/wordpress"

state: directory

- name: Download latest wordpress

get\_url:

url: https://wordpress.org/latest.zip

dest: "{{ wordpress\_file }}"

- name: Extract to /var/www/html

unarchive:

src: "{{ wordpress\_file }}"

dest: "{{ wordpress\_dest}}"

remote\_src: True

- name: Copy wp-config-sample.php to wp-config.php

command: cp "{{ wordpress\_dest }}/wordpress/wp-config-sample.php" "{{

wordpress\_dest }}/wordpress/wp-config.php"

- name: Update database credentials in the file

replace:

dest: "{{ wordpress\_dest }}/wordpress/wp-config.php"

regexp: "{{ item.regexp }}"

replace: "{{ item.replace }}"

with\_items:

- { regexp: 'database\_name\_here', replace: 'wordpress' }

- { regexp: 'username\_here', replace: 'guest' }

- { regexp: 'password\_here', replace: 'osfy'}

- name: Restart apache2 server

service:

name: apache2

state: restarted

$ ansible-playbook wordpress.yml -K

SUDO password:

PLAY [Setup WordPress]

TASK [setup]

ok: [ubuntu]

TASK [Update the software package repository]

changed: [ubuntu]

TASK [Create a database for wordpress]

changed: [ubuntu]

TASK [Create downloads directory]

ok: [ubuntu]

TASK [Create target directory]

changed: [ubuntu]

TASK [Download latest wordpress]

ok: [ubuntu]

TASK [Extract to /var/www/html]

changed: [ubuntu]

TASK [Copy wp-config-sample.php to wp-config.php]

ch\* anged: [ubuntu]

TASK [Update database credentials in the file]

changed: [ubuntu] => (item={u'regexp': u'database\_name\_here', u'replace': u'wordpress'})

changed: [ubuntu] => (item={u'regexp': u'username\_here', u'replace': u'guest'})

changed: [ubuntu] => (item={u'regexp': u'password\_here', u'replace': u'osfy'})

TASK [Restart apache2 server]

changed: [ubuntu]

PLAY RECAP

ubuntu : ok=10 changed=7 unreachable=0 failed=0

If you open the URL *http://targetserverip/wordpress* in a browser on the host system, you will see

wordpress setup page.

**Summary:**

*S* Configuration Management tools have majorly replaced the scripting languages in DevOps

for automation.

*S* It’s easy to manage and automate infrastructure from a centralised place compared to scripts.

*•S* Ansible is the current hot favourite automation tool and is very widely adopted in DevOps

life cycles.

*S* It’s easy to setup. Easy to read and write, does not need any programming knowledge.

*S* Ansible gives it little but very powerful AdHoc commands which is used to execute single

tasks on multiple servers.

*S* Playbooks are automation scripts of Ansible and includes series of tasks that should be

executed on selected hosts.

*•S* Ansible Roles gives us a very modular structure to our code and make the code sharable

with other team or projects. It focuses on reusability of the code.

*S* There are so many Sample Playbook in this chapter which you can use in your day to day

DevOps Automation.

*•S* Ansible is very innovative and releases new modules every now and then. Always refer

Ansible documentation, because that’s the single source of truth for everything in Ansible.

https://docs.ansible.com/

**Conclusion:**

Practice as much as possible when it comes to Ansible, always read Ansible documentation . Its currently most widely used automation tool and you should be an expert of it if you are using it in DevOps lifecycle or any automation.