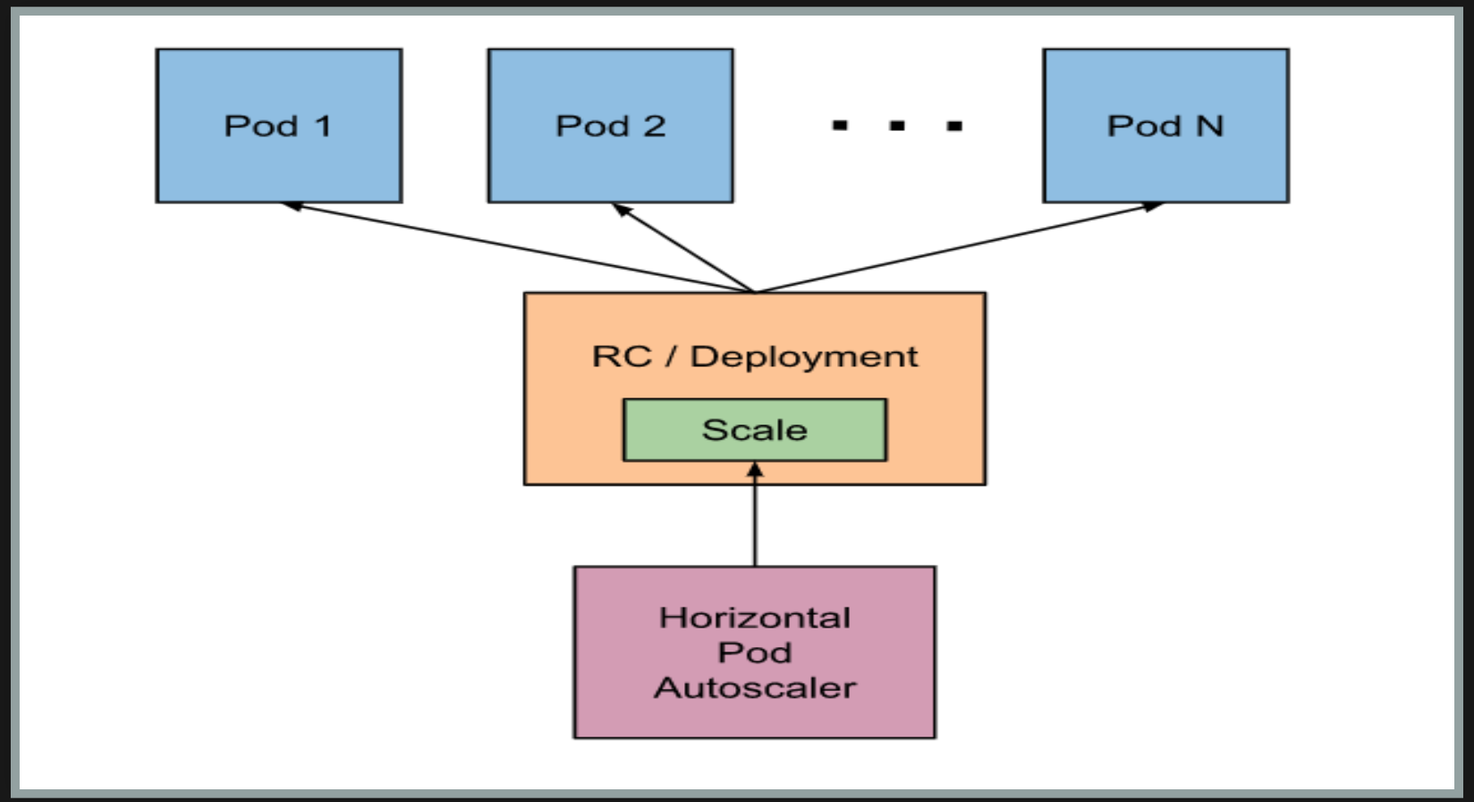
KUBERNETES

HORIZONTAL POD AUTOSCALER



DevOps course by Ali Kahoot - Dice Analytics

HPA

HPA needs metrics to evaluate whether to autoscale or not, for it we need metrics server

```
minikube addons list
```

```
minikube addons enable heapster
minikube addons enable metrics-server
```

HPA

Check Grafana

```
minikube addons open heapster
```

HPA

Deploy a sample application

```
<?php
  $x = 0.0001;
  for ($i = 0; $i <= 1000000; $i++) {
    $x += sqrt($x);
  }
  echo "OK!";
?>
```

```
kubectl run php-apache --image=k8s.gcr.io/hpa-example --requests=cpu=200m --limits=cpu=500m --expose --
```

HPA

Create Autoscaler

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```

HPA

Generate load

```
kubectl run --generator=run-pod/v1 -it --rm load-generator --image=busybox /bin/sh
```

After starting, run `while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done`

Within a minute or so, we should see the higher CPU load by executing:

```
kubectl get hpa
```

ANSIBLE

DevOps course by Ali Kahoot - Dice Analytics

CONFIGURATION MANAGEMENT

- Everything can be managed in code: servers, configurations, documentation, automated tests, deployment processes, and more.
- Configuration management is an automated method for maintaining computer systems and software in a known, consistent state.
- Configuration item is anything that can be configured e.g source code, proprietary files, binaries, servers.
- Concerned with managing evolving systems
- Configuration management tools enable
  - changes and deployments to be faster
  - repeatable
  - scalable
  - predictable
  - maintain the desired state

CONFIGURATION MANAGEMENT TOOLS

ANSIBLE

- Open source configuration management and provisioning tool.
- Automation engine that allows for agentless system configuration and deployment.
- Uses ssh to connect to the servers to execute tasks.
- Developed in python, hence you can use python packages in it.
- Enables you to control multiple nodes from a single machine.
- Project was founded in 2013.
- Acquired by RedHat it in 2015

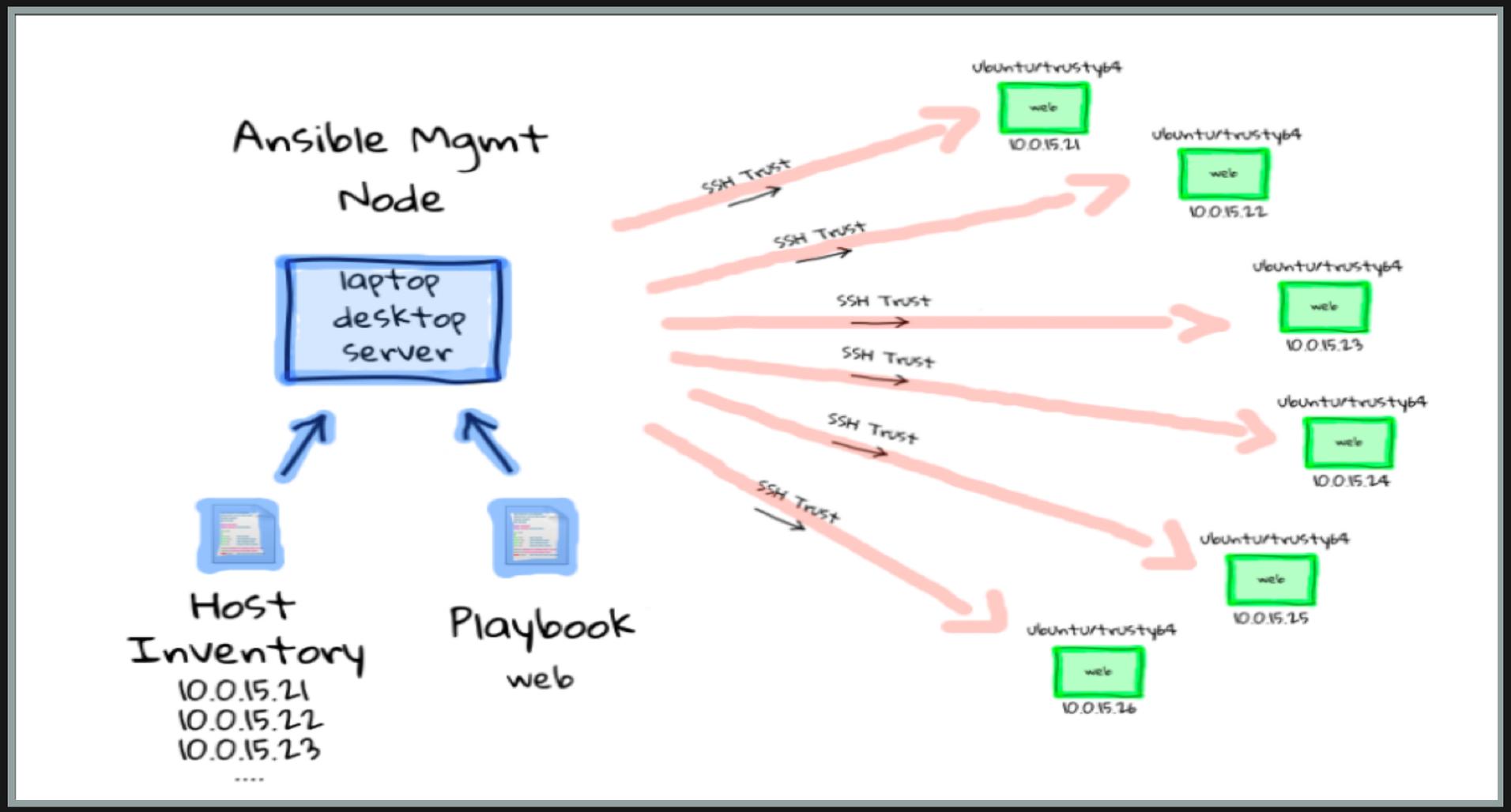
WHY ANSIBLE

- **Agentless:** you do not need any agent on the remote machine, as long as it has python installed and is accessible through ssh
- **Low Learning Curve:** Syntax and workflow are easy to understand, which makes it easy to learn for new users.
- **Simple:** Uses a simple syntax written in YAML.
- **Declarative:** Ansible works by you writing a description of the state of machine that you want
- **Idempotent:** No matter how many times you call the operation, the result will be same.

USE CASES

- Provisioning
- Configuration management.
- Application Deployment
- Continuous delivery
- Security and compliance
- Orchestration

ARCHITECTURE



CONTROL MACHINE

- Any machine with ansible installed
- Windows can't be used as a control machine
- We can run commands and playbooks from any control node
- We can have multiple control node- s
- System Requirements
  - Can run on any machine with python installed
  - Windows not supported

NODE MACHINE

- Machines we manage with ansible
- Ansible is not installed on node machines
- Node can be a virtual node, physical node, or other instance of softwares
- System Requirements
  - On managed nodes, we only need a way to communicate (ssh)

LAB - INSTALL ANSIBLE

DevOps course by Ali Kahoot - Dice Analytics

Install Required Packages by running the following command

```
sudo apt-get install -y python-pip libssl-dev
```

Install Ansible

```
pip install ansible
```

Check Ansible Version

```
ansible --version
```

LAB - SETUP ANSIBLE PLAYGROUND

- Generate SSH Keys
- Create Dockerfile
- Create docker-compose.yml
- Build Docker image
- Setup SSH agent
- Test SSH connection

GENERATE SSH KEYS

Login to your linux machine, open a terminal and execute the following command

```
$ ssh-keygen
```

Leave all settings to default and hit enter until you see a command prompt. This will generate two files 'id_rsa' and 'id_rsa.pub' private and public respectively which will be used to make a secure connection to your remote machines. Once you are done with generating files, create an empty directory and copy the public key to that directory.

```
$ mkdir diceAnsible && cp ~/.ssh/id_rsa.pub diceAnsible/
```

CREATE DOCKERFILE

Once done with generating ssh private and public keys, create a file named Dockerfile with following content. We'll copy our public keys to our docker image so that we can make passwordless ssh connection

```
$ FROM ubuntu:latest
RUN apt-get update && \
    apt-get install -y openssh-server pwgen netcat net-tools curl wget && \
    apt-get clean all
RUN apt-get update && apt-get install -y \
        build-essential \
 python \
 python-dev \
 libxml2-dev \
 libxslt-dev \
 libssl-dev \
 zlib1g-dev \
 libyaml-dev \
 libffi-dev \
 python-pip
# Latest versions of python tools via pip
RUN pip install --upgrade pip \
 virtualenv \
 requests
RUN ln -sf /usr/share/zoneinfo/Europe/Madrid /etc/localtime
RUN mkdir /var/run/sshd
RUN sed -ri 's/^PermitRootLogin\s+.*/PermitRootLogin yes/' /etc/ssh/sshd_config
RUN sed -ri 's/UsePAM yes/#UsePAM yes/g' /etc/ssh/sshd_config
RUN mkdir /root/.ssh
COPY id_rsa.pub /root/.ssh/authorized_keys
RUN chmod 400 /root/.ssh/authorized_keys
EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

CREATE DOCKER-COMPOSE.YML

Again create a file named docker-compose.yml with following content. We will create two docker containers which will listen on 2224 and 2225 respectively for ssh.

```
version: '3'
services:
  server1.test:
    image: diceansible
    hostname: server1.test
    ports:
      - "2224:22"
server2.test:
    image: diceansible
    hostname: server2.test
    ports:
      - "2225:22"
```

BUILD DOCKER IMAGE

Once you're done with creating files, build a docker image named: diceansible using following command.
Make sure you're in your project directory where you've both Dockerfile and docker-compose.yml

```
$ docker build -t diceansible .
```

RUN DOCKER CONTAINERS

Once you're done with image building. Execute following command to launch containers in detached mode. Make sure you're in directory where you've created docker-compose.yml.

```
$ docker-compose up -d
```

Test running containers using

```
$ docker ps
```

SETUP SSH AGENT

Setup ssh agent using following commands

```
$ ssh-agent bash
$ ssh-add ~/.ssh/id_rsa
```

After executing commands you will get a message like "Identity added: /Users/kahootali/.ssh/id_rsa (PC-NAME)"

TEST SSH CONNECTION

Once you've successfully launched your containers you can remote login to your containers using ssh.

```
# For first container
$ ssh root@localhost -p 2224
# For second  container
$ ssh root@localhost -p 2225
```

ANSIBLE CONCEPTS

- Inventory
- Playbooks
- Tasks
- Variables
- Modules
- Roles

INVENTORY

- Simple text files, which describes your servers, IP addresses or DNS names grouped by names.
- Ansible relies on an inventory for its basic functionality.
- We can use multiple inventory files at same time and pull inventory from dynamic or cloud sources.
- Two types of inventories
  - Dynamic
  - Static
- List of target hosts located in /etc/ansible/hosts

INVENTORY

- Group your inventory logically
  - Best practice is to group servers by their What, Where and When eg. db_east_prod.
- Avoid spacing, hyphen or preceding numbers in inventory groups.
- Group names are case sensitive
- Group variables within inventory
- Group of group is called children

INVENTORY - SAMPLE

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

INVENTORY - SAMPLE

```
dsb-testd9dm0    ansible_host=172.20.1.5    ansible_ssh_user=hdpadmin ansible_ssh_private_key_file=ssh.
dsb-testd9dm1    ansible_host=172.20.1.6    ansible_ssh_user=hdpadmin ansible_ssh_private_key_file=ssh.
dsb-testd9dm2    ansible_host=172.20.1.7    ansible_ssh_user=hdpadmin ansible_ssh_private_key_file=ssh.
```

INVENTORY - SAMPLE

```
[atlanta]
host1
host2

[raleigh]
host2
host3

[southeast:children]
atlanta
raleigh

[southeast:vars]
some_server=foo.southeast.example.com
halon_system_timeout=30
self_destruct_countdown=60
escape_pods=2
```

LAB - WORKING WITH INVENTORIES

- Create a configuration file
- Create ssh.config
- Create inventory file

CREATE A CONFIGURATION FILE

There are multiple ways to define ansible configuration file. We will create configuration file in our working project directory.

```
$ mkdir ansiblelab && cd ansiblelab
```

Using your favourite text editor create a file named: ansible.cfg with the following content.

```
[defaults]
# This is the default location of the inventory file, script, or directory that Ansible will use to #de
inventory = inventory
#specify here the remote hosts so we do not need to config them in main ssh config
[ssh_connection]
transport=ssh
ssh_args= -F ssh.config
```

CREATE SSH.CONFIG

Remain in the current directory (ansiblelab) and create new file named ssh.config with the following content

```
Host server1.test
    HostName localhost
    User root
    Port 2224
Host server2.test
    HostName localhost
    User root
    Port 2225
```

This file is referred in ansible.cfg, If there's no ssh.config, ansible will use global ssh configuration on your machine. In our test each host corresponds with a Docker container, that resides within our local machine and it is mapped by its exposed port.

CREATE INVENTORY

Finally create an inventory file which will contain list of servers we will use for our lab. We will categorise them in to database and webserver respectively. Create a file named: inventory with the following content.

```
[dev]
server1.test
[prod]
server2.test
```

AD-HOC COMMANDS

- Single ansible task, need to perform quickly and don't need to save for later.
- Can also be used to perform things quickly you don't want to write a full script
- Not used for configuration management and deployment, because these commands are of one time usage.

LAB - AD-HOC COMMANDS

- Ping multiple hosts with single command
- Copy a file to multiple machines
- Managing packages

PING MULTIPLE HOSTS WITH SINGLE COMMAND

We can ping multiple node machines from our control machine to check their health status

```
$ ansible all -m ping
```

COPY A FILE TO MULTIPLE MACHINES

we can perform parallel operations on multiple servers using a single shell.

```
$ ansible all -m copy -a "src=PATH_TO_LOCALFILE dest=PATH_TO_SERVERS" -u root
```

MANAGE PACKAGES

Ad-hoc commands are available for apt and yum since we're on ubuntu we have to move with apt. We can install packages, delete packages or even we can check state of a package. Like with below command we can check either a package exits on remote machines or not.

```
$ ansible all -m apt -a "name=nginx state=absent" -u root
```

So, if you need to install a package on multiple machines in a single go you can do with below command.

```
$ ansible all -m apt -a "name=nginx state=present" -u root
```

FACTS

- Simply various properties regarding a remote system.
- Gathered by default in ansible playbook execution
  - Gather_fact keyword can be used to change fact gathering behaviour.
- We can print ansible facts in files using variables
- May be filtered using setup module ad-hoc by passing a value for filter parameter.
- It possible to use {{ ansible_facts }} for conditional plays based on facts

FACTS

```
ansible all -m setup
```


```
localhost | SUCCESS => {
    "ansible_facts": {
        "ansible_XHC0": {
            "device": "XHC0",
            "flags": [],
            "ipv4": [],
            "ipv6": [],
            "macaddress": "unknown",
            "mtu": "0",
            "type": "unknown"
        },
        "ansible_XHC1": {
            "device": "XHC1",
            "flags": [],
            "ipv4": [],
            "ipv6": [],
            "macaddress": "unknown",
            "mtu": "0",
            "type": "unknown"
        },
        "ansible_XHC20": {
            "device": "XHC20",
            "flags": [],
            "ipv4": [],
            "ipv6": [],
            "macaddress": "unknown",
            "mtu": "0",
            "type": "unknown"
```

TASKS

- A task is simply a call to preset ansible module
- Each playbook contains its own list of tasks
- Executed in order, one at a time againt all the machines matched by host pattern group.
- All hosts in a group receive same task directives.
- On failure tasks are taken out of the rotation for the entire playbook run.

LAB - TASKS

- Check ssh status
- Execute task

CHECK SSH STATUS

Create a file name sshstatus.yml with following content,

```
---
- hosts: all
  user : root
  tasks:
   - name: make sure apache is running
     service: name=ssh state=started
```

EXECUTE PLAYBOOK

Using ansible-playbook command execute your newly created playbook.

```
$  ansible-playbook sshstatus.yml
```

PLAY

- A set of instructions expressed in YAML.
- It targets a host or group of hosts.
- Provides series of tasks (basically ad-hoc commands) that are executed in a sequence.
- Goal is to map a group of hosts to well defined role.
- Kept in files known as playbooks
- A playbook can contain more than one play.
- Each play may have a number of options configured, e.g remote_user, become
- Most of errors are due to improper indentation.

PLAYBOOKS

- Files where ansible code is written (YAML format).
- Like a todo list for ansible that contains a list of tasks.
- Playbooks run sequentially and are building blocks for all the use case of ansible.
- Must be executed using ansible-playbook command.
- Ansible playbook command takes few basic arguments
  - Inventory file to use (indicated with -i)
  - Playbook to be executed.

PLAY VS PLAYBOOK

- Play
  - List of tasks and role that should be run
  - Entity that tracks an account as it moves through a playbook.
- Playbook
  - List of plays
  - Can contain one or more plays.
  - Workflow designed for specific activity.

PLAYBOOK - SAMPLE

```yaml
---
- name: play 1
  hosts: all
  become: true
  pre_tasks:
  - name: do something before roles
    debug: msg="this is run before a role"
  roles:
  - install_role

- name: play 2
  hosts: group2
  roles:
  - config_role
```

LAB - WORKING WITH PLAYBOOK

- Create a simple playbook
- Execute playbook

CREATE A SIMPLE PLAYBOOK

Using your favourite text editor create a file named: myfirstplaybook.yml with following content.

```yaml
---yaml # My First YAML playbook for ansible
- hosts: all
  user: root
  sudo: yes
  connection: ssh
  gather_facts: yes
```

EXECUTE PLAYBOOK

Using ansible-playbook command execute your newly created playbook.

```
$  ansible-playbook myfirstplaybook.yml
```

DEFINE VARIABLES AND THEIR VALUES

Finally create variable files with respective content which will be further used when we will move forward with ansible playbooks. Make sure you're in your project directory.

```
$ mkdir group_vars
$ touch group_vars/host_group
```

VARIABLES

- Similar to variables in a programming language.
- helps you to use and assign a value to a variable and use that anywhere in the playbook.
- Variable's value is assigned using a colon, like so: foo: bar.
- Variable names should be letters, numbers and underscores.
- Should always start with a letter
- Can be scoped by a group, host, or within a playbook
- We can define variables in multiple ways
  - Via command line argument
  - Within variable file
  - Within a playbook
  - Within an inventory file

MODULES

- Units of code Ansible executes
- Each one has a particular use
- You can invoke a single module with a task
- There are several hundreds shipped with ansible
  - apt/yum, service, file, git, get_url etc
- You can create your own modules

TEMPLATES

- Provides simple means of maintaining various configuration files that may be customised on deployment.
- Files with ansible variables inside that are substituted on play execution.
- Uses template module

RUN COMMANDS

- If you don't find a module that suits your need, There are "Run commands" for this.
  - command: Takes the command and execute on the host
  - shell: Execute through a shell, like /bin/sh
  - script: Runs a local script on remote node
  - raw: Raw Module works like the Shell Module except it doesn't do any error checking

LAB - RUN COMMANDS

- Ansible shell module
- Execute playbook
- Ansible command module
- Execute playbook

ANSIBLE SHELL MODULE

We will perform pipe "|" operation using ansible shell module. Create a file named playbookshell.yml with following content.

```
- hosts: all
  tasks:
- name: Ansible shell module multiple commands
  shell: "cat {{ item }}"
  with_items:
    - hello.txt
    - hello2.txt
    - hello3.txt
  args:
    chdir: /root/
```

EXECUTE PLAYBOOK

Using ansible-playbook command execute your newly created playbook.

```
$  ansible-playbook playbookshell.yml -u root
```

ANSIBLE COMMAND MODULE

We will create simple text files on remote nodes using ansible command module. Create file named playbook.yml with following content.

```
- hosts: all
  tasks:
  - name: Ansible command module multiple commands
    command: "touch {{ item }}"
    with_items:
      - hello.txt
      - hello2.txt
      - hello3.txt
    args:
      chdir: /root/
```

EXECUTE PLAYBOOK

Using ansible-playbook command execute your newly created playbook.

```
$  ansible-playbook playbook.yml -u root
```

BLOCKS

- Logical grouping of tasks
- allows you to group related tasks together and apply particular task parameters on the block level.
- Allows you to handle errors like other programming languages.
  - block, runs first
  - rescue, runs on failure
  - always , always run
- Helps building reliable playbooks

ERROR HANDLING

- Ignoring Failed Commands
- Resetting Unreachable Hosts
- Controlling What Defines Failure
- Overriding The Changed Result
- Aborting the play
- Using blocks

LAB - ERROR HANDLING

DevOps course by Ali Kahoot - Dice Analytics

IGNORING FAILED COMMANDS

Playbooks will stop executing any more steps on a host that has a task fail. But you can ignore the error by adding ignore_error. Create ignore_error.yaml and paste the following

```
- name: Ignore Error
  hosts: localhost
  gather_facts: false
  tasks:
    - name: this will not be counted as a failure
      command: /bin/false
      ignore_errors: yes
```

Run the playbook

```
ansible-playbook ignore_error.yaml
```

CONTROLLING WHAT DEFINES FAILURE

Ansible lets you define what "failure" means in each task using the failed_when conditional. Create control_failure.yaml and paste the following

```
- name: Control Failure
  hosts: localhost
  gather_facts: false
  tasks:
    - name: Fail if return is not zero
      command: /bin/false
      register: cmd_output
      failed_when: cmd_output.rc != 0
```

Run the playbook

```
ansible-playbook control_failure.yaml
```

OVERRIDING THE CHANGED RESULT

When a shell/command or other module runs it will typically report "changed" status based on whether it thinks it affected machine state.Sometime you might want to override the "changed" result. Create override_change.yaml and paste the following

```
- name: Override Change Result
  hosts: localhost
  gather_facts: false
  tasks:
    - command: /bin/fake_command
      register: result
      ignore_errors: True
      changed_when:
        - result.rc == 2
```

Run the playbook

```
ansible-playbook override_change.yaml
```

USING BLOCKS

Blocks also introduce the ability to handle errors in a way similar to exceptions in most programming languages. Create file named: blocks.yml with following content.

```
---
- hosts: all
  tasks:
  - block:
    - debug:
        msg: 'I execute normally'
    - name: i force a failure
      command: /bin/false
  - debug:
      msg: 'I never execute, due to the above task failing, :-('
  rescue:
    - debug:
        msg: 'I caught an error'
    - name: i force a failure in middle of recovery! >:-)
      command: /bin/false
    - debug:
        msg: 'I also never execute :-('
  always:
    - debug:
        msg: "This always executes"
```

EXECUTE PLAYBOOK

Using ansible-playbook command execute your newly created playbook.

```
$  ansible-playbook blocks.yml -u root
```

CONDITIONALS

- Playbooks are capable of making actions conditional.
- The "when" keyword is used to test a condition in a playbook
- We can also use module output conditionally

LAB - CONDITIONALS

- Create playbook to create multiple users
- Execute playbook

CREATE A SIMPLE PLAYBOOK

Using your favourite text editor create a file named: conditionals.yml with following content. It will perform operations based upon your linux distribution.

```
--- # when playbook example
- hosts: all
  user: root
  sudo: yes
  connection: ssh
gather_facts: yes
vars:
  playbook_type: conditional example

tasks:
  - name: Install apache to the distribution type(Centos7)
    command: yum install httpd -y
    when: ansible_os_family == "RedHat"
  - name: Install apache to the distribution type(RedHat)
    command: apt-get install apache2 -y
    when: ansible_os_family == "Debian"
```

EXECUTE PLAYBOOK

Using ansible-playbook command execute your newly created playbook.

```
$  ansible-playbook conditionals.yml
```

LOOPS

- loop keyword May be used to express a repeated action more precisely.
- loop can also operate with a list variable
- It is also possible to combine loops and conditionals

LAB - LOOPS

- Create playbook to create multiple users
- Execute playbook

CREATE A SIMPLE PLAYBOOK

Using your favourite text editor create a file named: loops.yml with following content.

```
--- # LOOP PLAYBOOK EXAMPLE
- hosts: all
  connection: ssh
  gather_facts: yes
  tasks:
  - shell: "echo {{ item }}"
    loop:
      - "one"
      - "two"
  - name: add several users
    user:
      name: "{{ item }}"
      state: present
    loop:
      - testuser1
      - testuser2
```

EXECUTE PLAYBOOK

Using ansible-playbook command execute your newly created playbook.

```
$  ansible-playbook loops.yml
```

Exec into container and run `cat /etc/passwd` to see if user is added.

HANDLERS

- Handlers are just like regular tasks in ansible playbooks but only run if task contain a notify directive and also indicates that it has changed something.
- May be defined similarly to tasks
- Independent and Idempotent
- May be called any time after configuration was done in any order

HANDLERS

Hereâ□□s an example of restarting two services when the contents of a file change, but only if the file changes:

```
- name: template configuration file
  template:
    src: template.j2
    dest: /etc/foo.conf
  notify:
     - restart memcached
     - restart apache
```

LAB - HANDLERS

- Create playbook to create multiple users
- Execute playbook

CREATE A SIMPLE PLAYBOOK

Using your favourite text editor create a file named: handlers.yml with following content. It will perform operations based upon your linux distribution.

```
---
- hosts: all
tasks:
 - name: Install Nginx
   apt: pkg=nginx update_cache=true
   notify:
    - Start Nginx

handlers:
 - name: Start Nginx
   service: name=nginx state=started
```

EXECUTE PLAYBOOK

Using ansible-playbook command execute your newly created playbook.

```
$  ansible-playbook handlers.yml
```

ROLES

- Primary mechanism of breaking playbook into multiple files
- Improves resusibility
- Each role is limited to particular functionality
- Small functionalities which can be used independently within playbooks
- Bits of code transferred to the target system and executed to satisfy task declaration

ROLES STRUCTURE

```
httpd/
  tasks/
       main.yaml
  handlers/
       main.yaml
  files/
       template.txt
templates/
       temp.j2
vars/
       main.yaml
defaults/
       main.yaml
meta/
       main.yaml
```

ROLES STRUCTURE

- **tasks**: contains the main list of tasks to be executed by the role.
- **handlers**: contains handlers, which may be used by this role or even anywhere outside this role.
- **defaults**: default variables for the role
- **vars**: other variables for the role
- **files**: contains files which can be deployed via this role.
- **templates**: contains templates which can be deployed via this role.
- **meta**: defines some meta data for this role.

USING ROLES

```yaml
- hosts: webservers
  roles:
    - common
    - webservers
```

TAGS

- Run only a specific part of it rather than running everything
- You can filter tasks based on tags in two ways:
  - Command line
  - Ansible configuration settings
- Same tag name to more than one task
- Adding tags to a play or to statically imported tasks and roles, adds those tags to all
- Special keywords for tags
  - tagged
  - untagged
  - all

LAB - TAGS

DevOps course by Ali Kahoot - Dice Analytics

In this lab you are going to create a playbook with tags. First create a tags-example.yaml file and paste the following

```
- name: Play with tags
  hosts: localhost
  gather_facts: false
  tags:
      - test-tag
  tasks:
      - name: With Tag
        debug: msg="With Tag"

- name: Play without tags
  hosts: localhost
  gather_facts: false
  tasks:
      - name: Without Tag
        debug: msg="Without Tag"
```

Next you are going to run the test-tag play by specifying test-tag as shown below

```
ansible-playbook tags-example.yaml --tags test-tag
```

You can skip tasks by specifying the tags to skip as shown below

```
ansible-playbook tags-example.yaml --skip-tags test-tag
```

You can run only the tagged tasks by running the following command

```
ansible-playbook tags-example.yaml --tags tagged
```

You can run only the untagged tasks by running the following command

```
ansible-playbook tags-example.yaml --tags untagged
```

You can run all the tasks by running the following command

```
ansible-playbook tags-example.yaml --tags all
OR
ansible-playbook tags-example.yaml
```

ANSIBLE GALAXY

- Hub for finding and sharing Ansible content.
- Can use the site to share roles that you create
- Reusable Roles for server configuration or application installation
- Jump-start your automation project

GALAXY COMMANDS

- **ansible-galaxy list**: displays a list of installed roles
- **ansible-galaxy remove [role]**: removes an installed role
- **ansible-galaxy init**: can be used to create a role template suitable for Ansible Galaxy

LAB - GALAXY

In this lab you are going to install role from galaxy, then use that role. First you are going to download the role using the following command

```
ansible-galaxy install marcinpraczko.hello_world
```

You should see the role at ~/.ansible/roles

```
ls ~/.ansible/roles
```

Next create a playbook galaxy-text.yaml file and paste the following

```
- hosts: localhost
  roles:
     - { role: "marcinpraczko.hello_world" }
```

Run the playbook by running the following command

```
ansible-playbook galaxy-test.yaml
```

ANSIBLE BEST PRACTICES

- Separate inventory files for different environments.
- Make use of alternate directory layout.
- When using cloud, use dynamic inventories
- Use ansible-galaxy for role dependencies management.
- Keep plays simple and avoid adding tasks in main play.
- Try keep all variables at one place
- Keep name consistancy b/w your plays, inventories, roles and group variables
- Use prefixes and Human meaningful names in variables
- Clean up your debugging messages

LAB - BRINGING ALL TOGETHER

- Create host inventory
- Create Playbook
- Create group_vars
- Create mysql role
- Create nginx role
- Create php role
- Create wordpress role
- Validate your project structure
- Run playbook
- Continue installation

We are much familiar with ansible basic concepts. In this lab we will try to bring some concepts together to automate a wordpress application. Remove your containers and launch new one with docker compose. Please make sure that you're using freshly launched containers

CREATE HOST INVENTORY

Create an empty project directory, navigate in to that directory and create a file named: hosts with following content.

```
[webservers]
localhost:2224
```

The hosts file will contain address of node machines grouped together with respect to their usage type.Create playbook Now being in the same directory create a new file named playbook.yml with the following content.

```
- hosts: webservers
  roles:
    - nginx
    - wordpress
    - mysql
    - php
```

CREATE GROUP_VARS

Create a new directory named: group_vars and inside that directory create a new file named: all with following content.

```
# WordPress database settings
wp_db_name: wordpress
wp_db_user: wordpress
wp_db_password: wordpress_db_password

mysql_port: 3306
mysql_root_password: root


auto_up_disable: true

server_hostname: YOUR_CONSTAINER_IP_HERE

# WordPress Version
wp_version: 4.7.5
wp_sha1sum: fbe0ee1d9010265be200fe50b86f341587187302

core_update_level: true
```

Use docker inspect command to copy ip address of your container and place that on place of "YOUR_CONTAINER_IP_HERE" in above file.

CREATE MYSQL ROLE

In the root directory of your project create a directory named: roles. Navigate in to role and create another directory named: mysql. We'll place all mysql related stuff in this directory.

Inside mysql directory create a directory named: templates, Navigate in to templates and create a file named: .my.cnf and paste following content.

```
[client]
user=root
password={{ mysql_root_password }}
```

Inside mysql directory create tasks directory and create a new file named main.yml inside tasks directory and paste following.

```
---
- name: Install mysql
  become: yes
  apt:
    name: "{{ item }}"
  with_items:
    - mysql-server
    - python-mysqldb

- name: Start the MySQL service
  command: service mysql start

- name: Set root user password
  become: true
  become_user: root
  mysql_user:
    name: root
    host: "{{ item }}"
    password: "{{ mysql_root_password }}"
    priv: "*.*:ALL,GRANT"
  with_items:
    - "{{ ansible_hostname }}"
    - 127.0.0.1
    - ::1
    - localhost
- name: Copy .my.cnf file with root password credentials
  become: true
  become_user: root
  template: src=roles/mysql/templates/.my.cnf dest=/root/.my.cnf owner=root mode=0600
```

```
- name: Create mysql user
  become: true
  mysql_user:
    name: "{{ wp_db_user }}"
    password: "{{ wp_db_password }}"
    priv: "*.*:ALL"

- name: Create mysql database
  become: true
  mysql_db:
    name: "{{ wp_db_name }}"
    state: present

- name: Delete anonymous MySQL server user for $server_hostname
  become: true
  action: mysql_user user="" host="{{ server_hostname }}" state="absent"

- name: Delete anonymous MySQL server user for localhost
  become: true
  action: mysql_user user="" state="absent"

- name: Remove the MySQL test database
  become: true
  action: mysql_db db=test state=absent

- name: Update mysql root password for all root accounts
  become: true
  mysql_user: name=root host={{server_hostname}} password={{mysql_root_password}} priv=*.*:ALL,GRANT
  with_items:
    - "{{ ansible_hostname }}"
    - 127.0.0.1
    - ::1
    - localhost
```

Inside mysql create another directory named: handlers and create a new file in handlers directory named: main.yml

```
---
#handler for mysql, we'll place mysql handlers here if needed.
```

CREATE NGINX ROLE

As we have created handlers, tasks and templates respectively for mysql, in the same way we will create all these for nginx. In the roles directory create a directory named: nginx.

Navigate in to nginx and create a new directory named: handlers. Inside handlers create a file named main.yml with the following content.

```
---
- name: restart nginx
  service: name=nginx state=restarted
  become: yes
```

Inside nginx directory create another directory named: tasks and in the tasks directory create a file named: main.yml with the following content

```
---
# tasks file for nginx

- name: Update apt cache
  apt: update_cache=yes cache_valid_time=3600
  become: yes

- name: Install nginx
  apt: name={{ item }} state=present
  become: yes
  with_items:
    - nginx
    - git

- name: Start nginx
  become: yes
  service:
    name: nginx
    state: started

- name: Update nginx confs for WordPress + PHP
  template: "src=../templates/default-site.conf dest=/etc/nginx/sites-available/{{server_hostname}} own
  become: yes

- name: Enable site
  file: src=/etc/nginx/sites-available/{{server_hostname}} dest=/etc/nginx/sites-enabled/{{server_hostn
  notify:
    - restart nginx
  become: yes
```

Inside nginx create another directory named: templates just like we did in the case of mysql. Inside templates directory create a file named: default-site,conf with the following content. This will help us to serve our wordpress site.

```
server {
  listen 80;
  listen [::]:80;
  server_name {{ server_hostname }};
  root /srv/www/{{ server_hostname }};
  server_tokens off;
  include /etc/nginx/nginx-wp-common.conf;
}
```

Inside templates directory create another file named: nginx-wp-common.conf with the following content.

```
charset utf-8;
location / {
  index index.php index.html;
  try_files $uri $uri/ /index.php?$args;
}
error_page 404 /404.html;
error_page 500 502 503 504 /50x.html;
location = /50x.html {
  root /usr/share/nginx/html;
}
# Disallow .php files in uploads
location ~* /(?:uploads|files)/.*\.php$ {
  deny all;
}
# Add trailing slash to */wp-admin requests
rewrite /wp-admin$ $scheme://$host$uri/ permanent;
# Prevent hidden files (beginning with a period) from being served
location ~ /\. {
  access_log off;
  log_not_found off;
  deny all;
}
```

```
# Pass uploaded files to wp-includes/ms-files.php
rewrite /files/$ /index.php last;
if ($uri !~ wp-content/plugins) {
  rewrite /files/(.+)$ /wp-includes/ms-files.php?file=$1 last;
}
# Rewrite multisite in a subdirectory '.../wp-.*' and '.../*.php'
if (!-e $request_filename) {
  rewrite ^/[_0-9a-zA-Z-]+(/wp-.*) $1 last;
  rewrite ^/[_0-9a-zA-Z-]+.*(/wp-admin/.*\.php)$ $1 last;
  rewrite ^/[_0-9a-zA-Z-]+(/.*\.php)$ $1 last;
}
location ~ \.php$ {
  try_files $uri =404;
  include /etc/nginx/fastcgi_params;
  fastcgi_read_timeout 3600s;
  fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
  fastcgi_pass unix:/run/php/php7.2-fpm.sock;
  fastcgi_split_path_info ^(.+\.php)(/.+)$;
  fastcgi_index index.php;
}
```

CREATE PHP ROLE

Back in the roles directory create directory named: php. We will create handlers and tasks just like we did for other services.

Inside php directory create a new directory named: handlers and create a new file in handlers directory named: main.yml with the following content.

```
---
- name: restart nginx
  service: name=nginx state=restarted
  become: yes
```

Back in to php directory and create another file named: tasks and inside tasks directory create a new file named: main.yml with following content.

```
---
# tasks file for php
- name: Install php extensions
  apt:
    name: "{{ item }}"
    state: present
  become: yes
  with_items:
    - php
    - php-mysql
    - php-gd
    - php-ssh2
    - php-fpm
- name: Setup php-fpm
  replace: dest=/etc/php/7.2/fpm/php.ini regexp="(;cgi.fix_pathinfo=1)" replace="cgi.fix_pathinfo=0"
  notify:
    - restart nginx
  become: yes
- name: start php-fpm
  command: service php7.2-fpm start
  #service: name=php7.2-fpm state=started
- name: Add php settings
  template: src=../../nginx/templates/nginx-wp-common.conf dest=/etc/nginx/nginx-wp-common.conf owner=w
  notify:
    - restart nginx
  become: yes
```

CREATE WORDPRESS ROLE

Back in to the roles directory create a new directory named: wordpress. Inside wordpress directory create three directories handlers, tasks and templates respectively.

Inside handlers create a file named: main.yml with following content.

```
---
- name: restart nginx
  service: name=nginx state=restarted
  become: yes
```

Inside tasks directory create a file named: main.yml with following content.

```
---
- name: Create webroot
  file:
    state: directory
    path: /srv/www/
    owner: root
    group: root
    mode: 0755
  become: yes
- name: Check if WordPress directory exists in /srv/www
  stat: path=/srv/www/{{server_hostname}}
  register: check_path
- name: Download WordPress
  get_url:
    url: https://wordpress.org/wordpress-{{ wp_version }}.tar.gz
    dest: /tmp/wordpress-{{ wp_version }}.tar.gz
    checksum: "sha1:{{ wp_sha1sum }}"
  become: yes
  when: not check_path.stat.exists
```

```
- name: Extract WordPress
  unarchive:
    src: /tmp/wordpress-{{ wp_version }}.tar.gz
    dest: /tmp
    owner: www-data
    group: www-data
    copy: no
  become: yes
  when: not check_path.stat.exists
- name: Move WordPress install files
  command: mv /tmp/wordpress /srv/www/{{server_hostname}}
  become: yes
  when: not check_path.stat.exists
- name: Fetch random salts for WordPress config
  local_action: command curl https://api.wordpress.org/secret-key/1.1/salt/
  register: "wp_salt"
  become: no
- name: Add wp-config
  template: "src=wp-config.php dest=/srv/www/{{server_hostname}}/wp-config.php"
  become: yes
- name: Update WordPress config file
  lineinfile:
    dest: "/srv/www/{{server_hostname}}/wp-config.php"
    regexp: "{{ item.regexp }}"
    line: "{{ item.line }}"
  with_items:
    - {'regexp': "define\\('DB_NAME', '(.)+'\\);", 'line': "define('DB_NAME', '{{wp_db_name}}');"}
    - {'regexp': "define\\('DB_USER', '(.)+'\\);", 'line': "define('DB_USER', '{{wp_db_user}}');"}
    - {'regexp': "define\\('DB_PASSWORD', '(.)+'\\);", 'line': "define('DB_PASSWORD', '{{wp_db_password
  become: yes
```

Finally in the templates directory create a file named: wp-config.php with the following content.

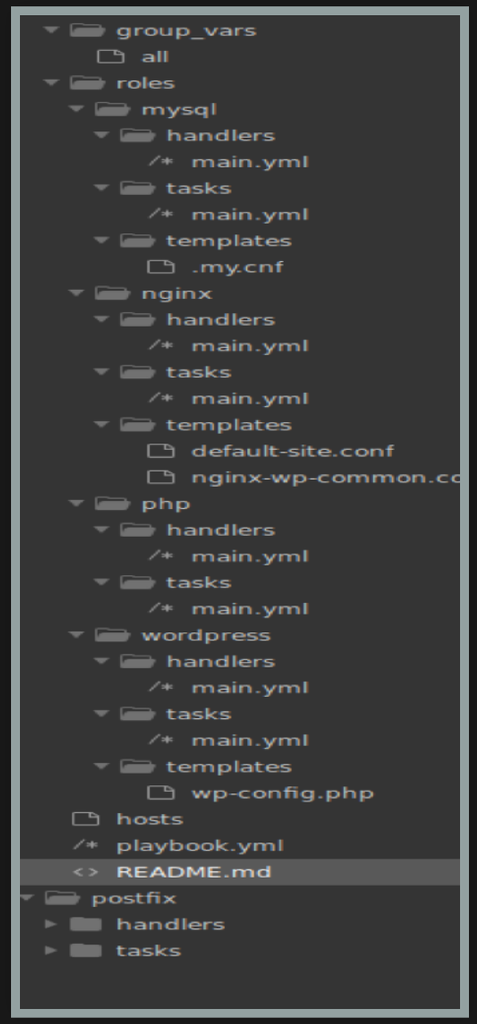
```
<?php
/**
 * The base configurations of the WordPress.
 *
 * This file has the following configurations: MySQL settings, Table Prefix,
 * Secret Keys, WordPress Language, and ABSPATH. You can find more information
 * by visiting {@link http://codex.wordpress.org/Editing_wp-config.php Editing
 * wp-config.php} Codex page. You can get the MySQL settings from your web host.
 *
 * This file is used by the wp-config.php creation script during the
 * installation. You don't have to use the web site, you can just copy this file
 * to "wp-config.php" and fill in the values.
 *
 * @package WordPress
 */
// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define('DB_NAME', '{{ wp_db_name }}');
/** MySQL database username */
define('DB_USER', '{{ wp_db_user }}');
/** MySQL database password */
define('DB_PASSWORD', '{{ wp_db_password }}');
/** MySQL hostname */
define('DB_HOST', 'localhost');
/** Database Charset to use in creating database tables. */
define('DB_CHARSET', 'utf8');
/** The Database Collate type. Don't change this if in doubt. */
define('DB_COLLATE', '');
```

```
/**#@+
 * Authentication Unique Keys and Salts.
 *
 * Change these to different unique phrases!
 * You can generate these using the {@link https://api.wordpress.org/secret-key/1.1/salt/ WordPress.org
 * You can change these at any point in time to invalidate all existing cookies. This will force all us
 *
 * @since 2.6.0
 */
{{ wp_salt.stdout }}
/**#@-*/
/**
 * WordPress Database Table prefix.
 *
 * You can have multiple installations in one database if you give each a unique
 * prefix. Only numbers, letters, and underscores please!
 */
$table_prefix  = 'wp_';
/**
 * WordPress Localized Language, defaults to English.
 *
 * Change this to localize WordPress. A corresponding MO file for the chosen
 * language must be installed to wp-content/languages. For example, install
 * de_DE.mo to wp-content/languages and set WPLANG to 'de_DE' to enable German
 * language support.
 */
```

```
define('WPLANG', '');
/**
 * For developers: WordPress debugging mode.
 *
 * Change this to true to enable the display of notices during development.
 * It is strongly recommended that plugin and theme developers use WP_DEBUG
 * in their development environments.
 */
define('WP_DEBUG', false);
/** Disable Automatic Updates Completely */
define( 'AUTOMATIC_UPDATER_DISABLED', {{auto_up_disable}} );
/** Define AUTOMATIC Updates for Components. */
define( 'WP_AUTO_UPDATE_CORE', {{core_update_level}} );
/* That's all, stop editing! Happy blogging. */
/** Absolute path to the WordPress directory. */
if ( !defined('ABSPATH') )
  define('ABSPATH', dirname(__FILE__) . '/');
/** Sets up WordPress vars and included files. */
require_once(ABSPATH . 'wp-settings.php');
```

VALIDATE YOUR PROJECT STRUCTURE

After following all above step your project structure should look like below



RUN PLAYBOOK

Once you're done with creating all files run your playbook with following command from root directory of your project.

```
$ ansible-playbook  playbook.yml  -i hosts -u root
```

CONTINUE INSTALLATION

Once you've successfully executed your playbook on your remote machine. Enter the ip address of your node machine in the browser and complete the setup.