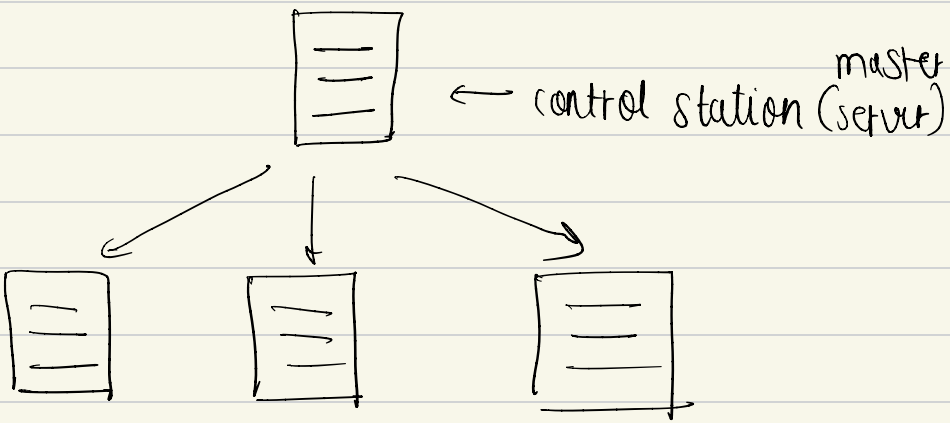
• Ansible Absolute Basics

used to control multiple servers at once



Ansible is

- python based
- agent less

? typically automation tool will require you to install some kind of agent or client on the machine

Ansible simply connects over SSH and runs commands

- push modul
  - pushes configurations and commands to other devices

→

cd /etc/ansible

has 3 main directories

- ansible.cfg - ansible config settings
- hosts - list of things we want to control our slave servers
- roles -

→ Editing host file

c/

```
[linux]
192.168.72.153
192.168.56.223
```

list of Ip addr's of servers we want to control

group of referencing

```
[linux:vars]
ansible_user=vagrant
ansible_password=vagrant
```

attributes we want to change for servers when we connect to them

- username and password of slave devices

- when you are just testing you can turn off host_key_checking=false

— Ping our servers

c/ ansible linux -m ping

module

c/ ansible linux -a 'cat /etc/os-release'

add/run commands on the fly on your slave servers

* More powerful way to use ansible is through Playbooks

- Playbooks - YAML file
  - plays
    - tasks - commands we run on linux machines

eg playbook

```
---
- name: install-nano
  hosts: linux
  tasks:
    - name: ensure nano is installed
      apt:
        name: nano
        state: latest
```

play

what we defined

tasks

module
↳ little small programs that define the state we want our server to be in

- run ansible playbook

ansible-playbook nano.yml

name of file

- if you run a command and get

changed = 0

as status
that means nothing changed

- Running ad-hoc commands

→ you can create your own ansible.cfg

c/
```
[defaults]
inventory = inventory.txt
private_key_file = ~/.ssh/ansible
```

inventory.txt: file where all slave server's IP addresses reside

~/.ssh/ansible: where ssh keys are stored

- some commands we can run

c/ ansible all -m ping

c/ ansible all --list-hosts
↳ list all hosts IP

c/ ansible all -m gather_facts
↳ retrieves all information of hosts

c| ansible all -m gather_facts --limit 172.16.250.132

just display it for a particular server

- Running elevated ad-hoc commands

what if we want to run sudo commands

c| ansible all -m apt -a update_cache=true --become --ask-become-pass

run as root user

basically just runs sudo apt update

installing package vim-nox

c| ansible all -m apt -a name=vim-nox --become --ask-become-pass

name of package we want to install

upgrading existing packages

```
c| ansible all -m apt -a 'name=vim state=latest'
   --become --ask-become-pass
```

use single quotes and put them together

upgrading all packages

```
c| ansible all -m apt -a 'upgrade=dist'
   --become --ask-become-pass
```

- Writing Playbooks

c/

```
---

- hosts: all
  become: true
  tasks:

    - name: Install apache2 package
      apt:
        name: apache2
        state: latest
```

(absent when uninstalling)

(optional) to keep it up to date

c/ ansible-playbook --ask-become-pass install-apache.yml

- The 'when' condition

what if we have servers of different distribution?
so deb commands now wont run
we need to need specific commands for specific distros

c/

```
- hosts: all
  become: true
  tasks:

    - name: Install apache2 package for Ubuntu
      apt:
        name: apache2
        state: latest
      when ansible_distribution == ['Debian', 'Ubuntu']

    - name: Install apache2 package for Centos
      dnf:
        name: httpd
        state: latest
      when ansible_distribution == 'Centos'
```

↑ multiple arguments

to check multiple conditions

c) when ansible_distribution == 'Ubuntu'
and
when ansible_version == '8.2'

- we can check this ansible variables using:

c) ansible all -m gather_facts | grep 'ansible'

- Improving Playbook

↳ in our inventory we can pass variables for each server

c/

```
172.16.250.132   apache_package=apache2
172.16.250.243   apache_package=httpd
```

↳ now in our playbook

c/ ...

```
- name: Install apache2 and php package
  package:
    name:
      - "{{ apache_package }}"
      - "{{ php_package }}"
    state: latest
    update_cache: yes
```

→ just use package it will set the package according to the distribution automatically

} variable names

- Targetting specific nodes

↳ create groups in inventory

c/
```
[web_servers]
192.168.56.2

[db_servers]
192.168.56.11

[file_servers]
192.168.56.12
```

→ change the playbook accordingly

c/ ---

```
- hosts : all
  become : true
  pre_tasks :

    - name : Install updates
      package :
          update_only : yes
          update_cache : yes

- hosts : web_servers
  become : true
  tasks :

      :
      :
```

this says : run this tasks before running anything else

← explicitly specify a host

- Tags

tags allow us to add metadata to our plays
so that way, we can only run plays that we want to test

so. add a tag for task

```
- name: Install updates
  tag: always
  package:
    update_only: yes
    update_cache: yes
```

→ to list all attached tags

```
ansible-playbook --list-tags site.yml
```

→ to run task with specific tag

```
ansible-playbook --tags always --ask-become-pass
  site.yml
```

→ multiple tags

```
-- tags 'apache, maria'
```

- Managing Files

→ copying files from workstation to hosts

```
- name: copy files for website
  tags: apache, apache2, httpd
  copy:
    src: default_site.html
    dest: /var/www/html/index.html
    owner: root
    group: root
    mode: 0644
```

this is stored in 'files/...'

→ install unzip and install program from ut

c/
```
- name: install unzip
  package:
    name: unzip

- name: install terraform
  unarchive:
    src: https://...
    dest: /usr/local/bin
    remote_src: yes
    mode: 0755
    owner: root
    group: root
```

- Managing Services

→ Starting http servers

c/
```
name : start httpd
tags :
service :
    name: httpd
    state: started
    enabled: yes
```

→ now what if we make a change and want to restart service only if there is a change

c/

```
- name: change email addr for admin
  tags:
  lineinfile:
      path: /etc/httpd/conf/httpd.conf
      regexp: '^ServerAdmin'
      line: ServerAdmin yash@123.com
  when:
  register: httpd
```

search line which starts with ServerAdmin

the actual changes

this is the variable we store the state of this task
so we know if there are any changes that occurred

c/

```
- name : restart httpd
  tag :
  service:
     name: httpd
     state: restarted
  when: httpd.changed
```

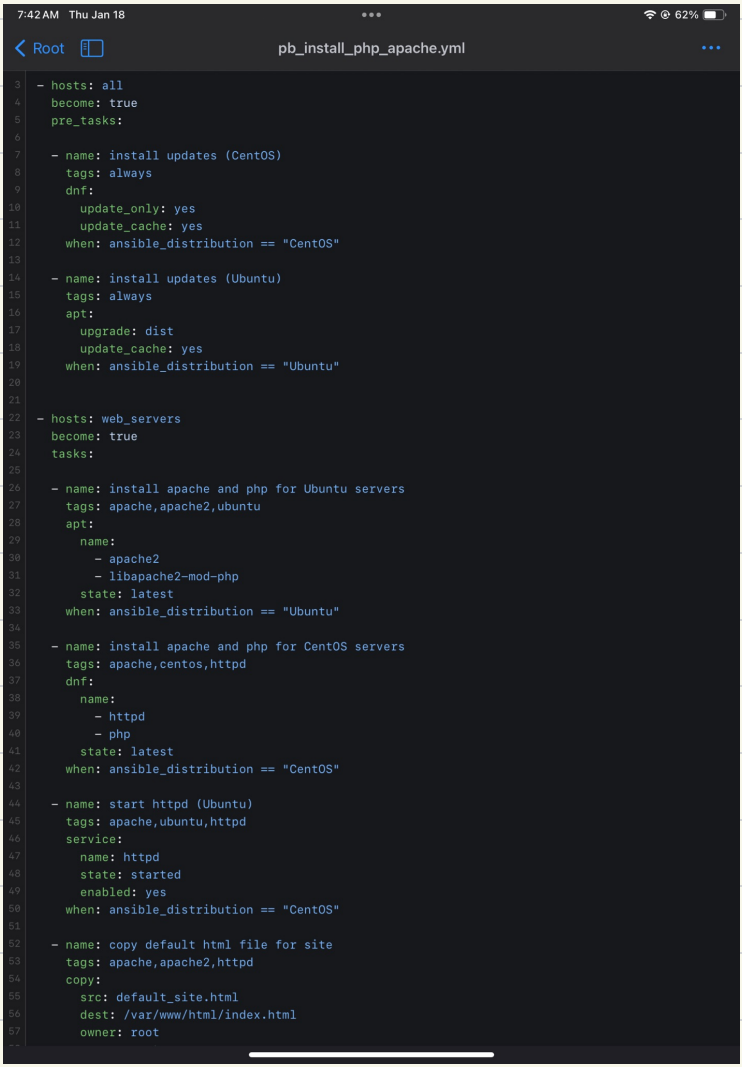
will only be restarted when there are any changes

changed_when : false

we can add this to apt update/upgrade
so no matter what happens
running this will not affect 'CHANGED' status

- Adding Roles

As we can see our playbook is getting really complicated so we can create roles for each servrs



For creating roles create a similar folder structure

- roles
  - base
    - tasks
      - main.yml
  - db_servers
    - tasks
      - main.yml
  - web_servers
    - tasks
      - main.yml
    - files
      - default_site.yml
  - file_server
    - tasks
      - main.yml

* main.yml ⟶ taskbooks
only contain tasks

c/eg :

```
- name: Install Samba Package
  tags: samba
  package:
      name: samba
      state: latest
```

→ now our main-playbook also changes

c/

```
---

- hosts: all
  become: true
  pre_tasks:

    - name: update repo index
      tags: always
      apt:
        update_cache: yes

- hosts: all
  become: true
  roles:
    - web-servers
```

pre tasks remain the same. Where we want to run the tasks on all hosts

this is a role which will run tasks from taskbooks

→ running this new playbook

c/ ansible-playbook site.yml

- Host variables

we need a directory to store variables

host_vars

inside this directory we need to create seperate files for each server

name of the file can be
- → Ip address
- → host name
- → DNS name

of the server

touch 192.168.0.113.yml

we can set variables inside this file for that particular ip

c/ apache_package_name: apache2
apache_servie: apache2
php_package_name: libapache2-mod-php

roles ↳ web_servers ↳ tasks ↳ main.yml

in main.yml

c/

```
- name : ——"——
  tags : ——"——
  package:
    name:
      - "{{ apache_package_name }}"
      - "{{ php-package_name }}"
    state : latest
```

- Handlers

instead of register we can use handler

Handler is a task that runs only if another tasks notifies it
Register is used to store the output of a task for later reference or conditional actions

so in

roles ↳ web_servers ↳ tasks ↳ main.yml
only restart apache server when something is changed so

c/
```
name: ——"——
tags: ——"——
lineinfile:
    path: ——"——
    regexp: ——"——
    line: ——"——
notify: restart_apache
```

now we will have to define this

so we create a new directory

roles ↳ web_servers ↳ handlers ↳ main.yml

and define our handler there

```
- name: restart_apache
  service:
    name: "{{ apache_service }}"
    state: restarted
```

this the name we defined earlier

- Template

templates have variable in it that we can use to apply to multiple hosts and then we can independently change the variable on each host as we see fit

so to create a template
we have to create a new directory

roles ↳ web_servers ↳ templates ↳ main.yml

where we can copy ssh file to convert it into a template

cp /etc/ssh/sshd_config sshd_config_ubuntu.j2
jinja2 format

we can add a variable in it sshd_config_ubuntu.j2

AllowUsers {{ ssh_users }}

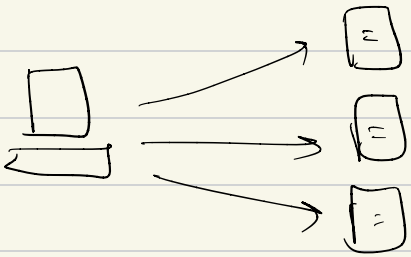
now we need to give values to these variables
so according to the server
variable values can differ for each server

so in

host_vars ↳ 192.168.x.x.yml
—"— .yml
...

c/ ssh_users: 'Yash Raj' ← variable value
ssh_template_file: sshd_config_ubuntu.j2
↑ name of variable file

further we want to run this template from host



to other hosts

so in

roles ↳ base ↳ tasks ↳ main.yml

```
- name: generate sshd-config file from template
  tags: ssh
  template:
    src: "{{ ssh_template_file }}"
    dest: /etc/ssh/sshd_config
    owner: root
    group: root
    mode: 0644
  notify: restart_sshd
```

↳ so now in

roles ↳ base ↳ handlers ↳ main.yml

we have to define this handler

c/

```
- name: restart_sshd
  service:
    name: sshd
    state: restarted
```