Ansible

By
ADITYA PRABHAKARA

Introducing Myself

Aditya S P (sp.aditya@gmail.com)
Freelance trainer and technologist

Boring Stuff about me:

- 14+ years of experience in development and training
- Started with Java, moved to Android and now working on Big Data Technologies

Interesting Things about me:

- Actually Nothing !

Getting to know you

Show of hands please!

- What is the general experience of this group
  - 0-2 years, 2-5 years, 5 and above
- What area are you currently working on?
- What is your current technology stack?
- Who is your cloud provider?

Agenda

- Ad-hoc commands, Inventories, Modules, Roles
- Plays and Playbooks
- Deployment strategies
- Galaxy and Tower

Objective

Course Objectives

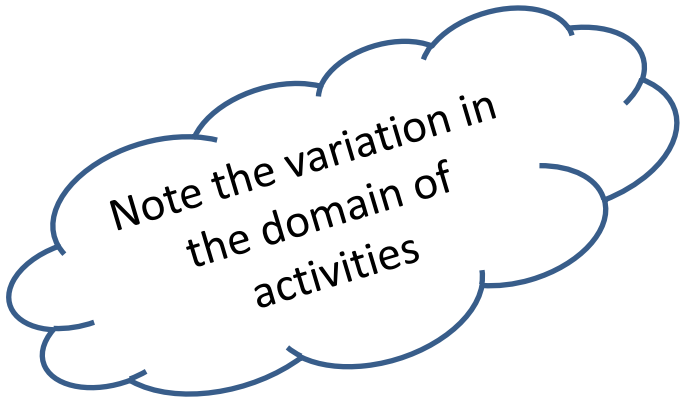
At the end of this you will be

- Very comfortable in Ansible
- Ability to re-factor and modularize ansible playbooks
- Performance affecting factors
- Different styles of deployment

Chapter: Introduction

Let us start with

- Some examples of Infrastructure activities
  - Setting up of bare metal servers
  - Install software
  - Set DB password using org policies
  - User creation
  - Deploy code to prod machines
  - .. Many more

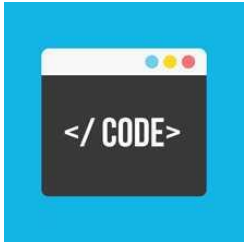


Historically…

- Development teams and Operations Team have worked as Silos
- Infrastructure is traditionally managed by
  - System gurus
  - Custom scripts or programs
  - Do this, then this… why not try this… ah ! May be we need to roll it back. Now we have it right and DO NOT TOUCH THIS…

Big Question? : How to make Dev and Ops team work seamlessly and how to make infra setup a reliable repeatable process ?

Reliable and Repeatable



- How to make Infra with all its varied tasks reliable and repeatable?
  - Represent infra as **CODE!!**
- What does it mean to represent infra as code?

What does it mean to represent it as code

Just means have a standard way to perform all infrastructure activities in an automated manner.

WIKIPEDIA defines it as

"**Infrastructure as code** (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools"

Advantages of representing infra as code

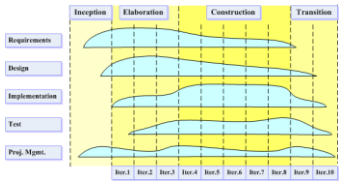
git

Version Control

Collaborate

Readable by everybody

</ CODE>



Unified Processes

SPEED

Enables CI / CD

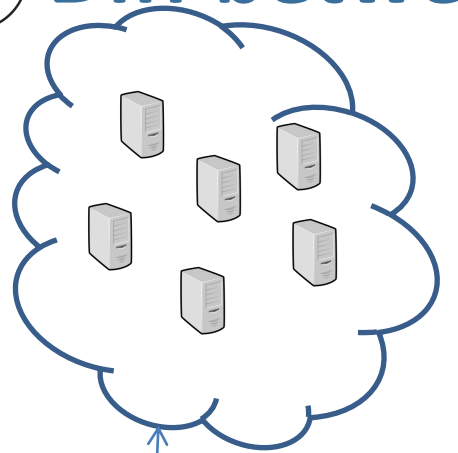
Some tools that make it possible

- Chef
- Puppet
- Ansible
- SaltStack
- Otter
- CFEngine

Why Ansible?

- Have a dead simple setup process and a **minimal learning curve**
- Manage machines very quickly and in parallel
- **Avoid custom-agents** and additional open ports,
  be agent less by leveraging the existing SSH daemon
- Describe infrastructure in a language that is both **machine and human friendly**
- Focus on security and easy auditability/review/rewriting of content
- Manage new remote machines instantly, without bootstrapping any software
- Allow module development in any dynamic language, not just Python
- Be usable as non-root
- Be the easiest IT automation system to use, ever.

Diff between Ansible and Chef

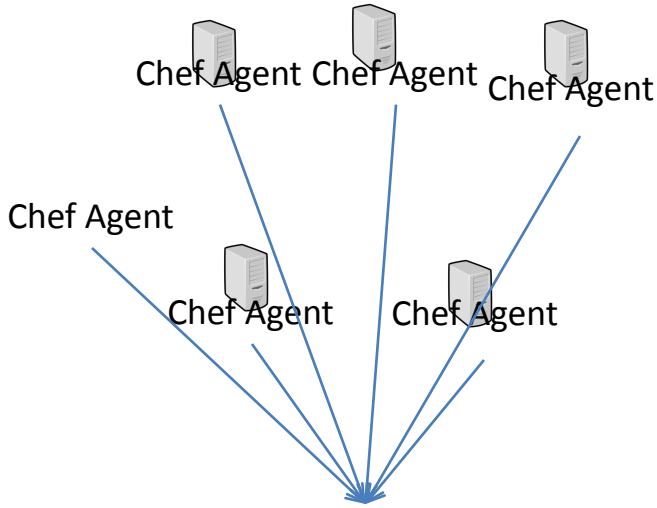


Pushes ops tasks

Control Machine

Ansible

CHEF



Workstation

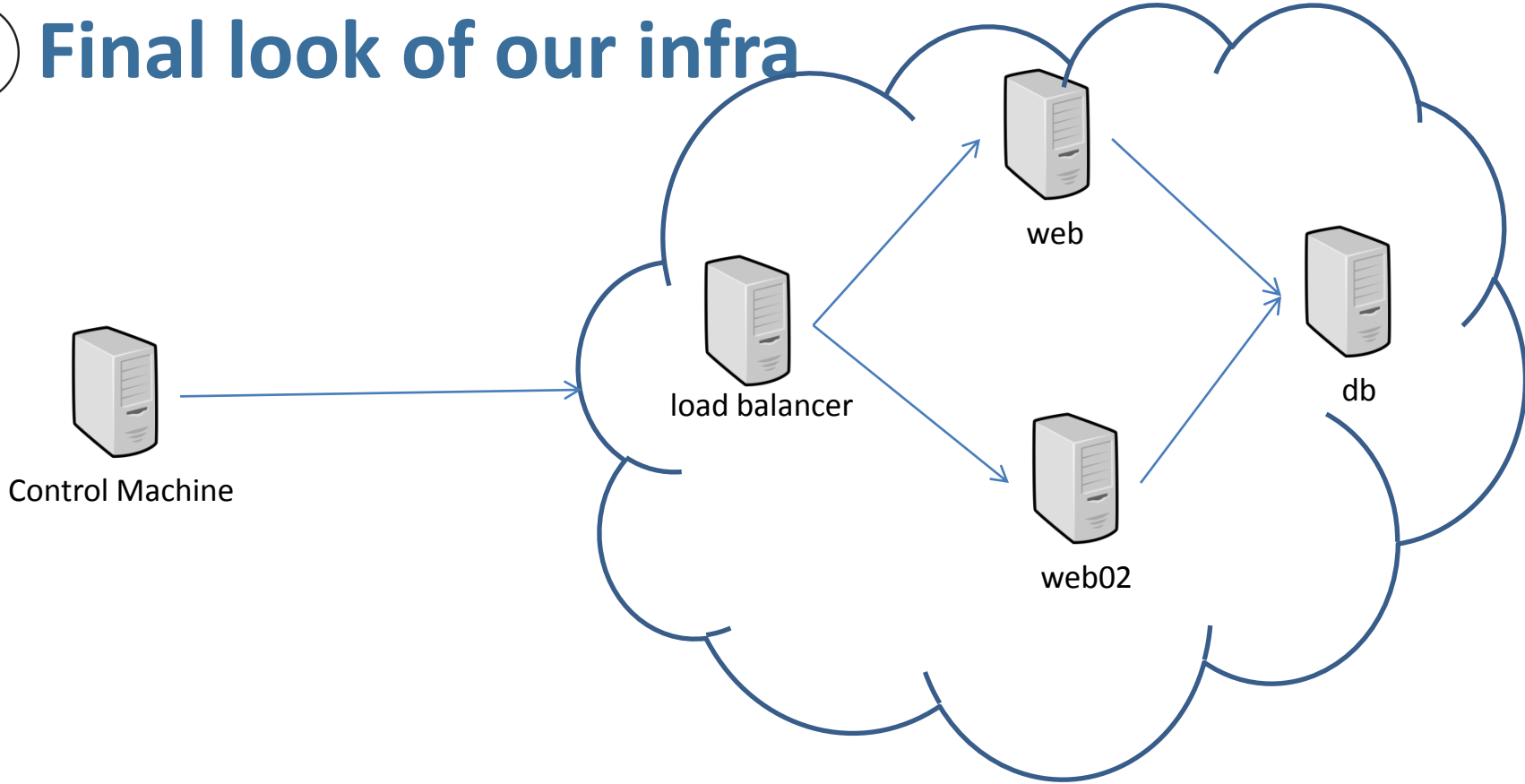
Chef Server

Chapter: Setup

Understanding out setup

- Vagrants and Virtual box

Final look of our infra



Chapter: Basics

Chapter: Inventory

Inventory

Default file : /etc/ansible/hosts

```
>> ansible --list-hosts all
```

```
[ctrl]
control
[database]
db
[webserver]
web
```

Lets now change the default file to : /home/vagrant/demo/hosts because

1. it's a pain to update in /etc/ansible/hosts
2. My configuration is also a part of my infra code!

Inventory

Groups with host ranges

```
[webserver]
web[01-99].prod.app.com
web-[a-f].prod.app.com
```

Groups with host ranges

```
[applayer:children]
web
db
```

Supports numeric ranges and alpha ranges

Inventory

Ungrouped hosts

```
web[01-99].prod.app.com
```

Default host names

```
all
ungrouped.all
```

Supports numeric ranges and alpha ranges

Configuration file priority

- ANSIBLE_CONFIG (an environment variable)
- ansible.cfg (in the current directory)
- .ansible.cfg (in the home directory)
- /etc/ansible/ansible.cfg

Inventory

Default file : /etc/ansible/hosts

```
>> ansible --list-hosts all
```

```
[control]
control
[database]
db
[webserver]
web
```

Change the default file to : /home/vagrant/demo/hosts

STEP 1: Create a file called ansible.cfg in /home/vagrant/demo
STEP 2: point the inventory configuration to the req hosts file

```
[defaults]
inventory =
/home/vagrant/demo/hosts
```

Host selection

Different ways of selecting hosts

```
>> ansible --list-hosts all
>> ansible --list-hosts “*”
>> ansible --list-hosts database
>> ansible --list-hosts db
>> ansible --list-hosts “con*”
>> ansible --list-hosts db:control
>> ansible --list-hosts webserver[0]
>> ansible --list-hosts \!control
```

Chapter: Tasks

Tasks or Modules or Ad-hoc execution

```
>> ansible all –m “ping”
>> ansible ctrl –m “ping”
>> ansible all –m “command” –a “hostname”
>> ansible all –m “apt” –a “update_cache=yes”
```

Learning about our env

```
>> ansible web -m setup >> /tmp/output.txt
>> ansible all -m command -a “df -h”
>> ansible all -m command -a “df -h”
>> ansible all -m command -a “free -m”
```

Some file operations

create file /tmp/trial.txt

```
ansible ctrl -m file -a "path=/tmp/trial.txt state=touch owner=vagrant
group=vagrant mode=0644"
```

create a directory /tmp/trialdirectory

```
>> ansible ctrl -m file -a "path=/tmp/trialdirectory state=directory
owner=vagrant group=vagrant"
```

delete file /tmp/trial.txt

```
ansible ctrl -m file -a "path=/tmp/trial.txt state=delete"
```

Two important points to note

- Declarative style of programming
  - Expresses the logic of programming without defining a control flow
- Idempotence
  - Property of certain operations in math and comp sci that can be applied multiple times without affecting the outcome
  - Decreases side effects
  - Will always leave in the system in the same desired end result

Configuring our servers

```
Refer envsetup.xls provided with material
```

Chapter: Playbooks

Playbooks

- Metaphor of Ansible to describe bunched up tasks
- These tasks are also called as plays
- These playbooks can be played on a host or groups of host
- Ad-hoc commands themselves are powerful enough but playbooks help in orchestration
- Play books are written as YAML files

YAML files

- Human readable and machine parsable
- YAML Ain't a Markup Language
- Light weight data representation
- Can hold multiple data structures – strings, integers, lists, dictionaries
- YAML can easily be converted to corresponding Python's datastructures

Open link http://yaml-online-parser.appspot.com/
Let us practice some YAML !

Ansible's YAML structure

```
---
- hosts: all
  become: true
  tasks:
  - name: install apache2
    apt : name=apache2 state=present update_cache=yes
```

- Keep an eye on how options are provided for the module apt
- This style of providing option is called as Single line key value pair

Two more ways to format tasks

- Folded scalar – multi line options

```
tasks:
- name: install apache2
  apt : >
    name=apache2
    state=present update_cache=yes
```

- Structured Map

```
tasks:
- name: install apache2
  apt :
    name: apache2
    state: present
    update_cache: yes
```

Chapter: First milestone

Let us first get our playbooks ready

```
Refer envsetup.xls provided with material
Build web.yml, db.yml, control.yml, app.yml
Modules used
  1. apt
  2. service
  3. lineinfile
```

apt

```
- name: install mysql-server
  apt: name=mysql-server state=present update_cache=yes
```

service

```
- name: ensure mysql started
  service: name=mysql state=started enabled=yes

- name: restart mysql
  service: name=mysql state=restarted
```

lineinfile

```
 - name: ensure mysql listening on all ports
   lineinfile: dest=<Path to file> regexp=^bind-address line="bind-
address = 0.0.0.0”
```

Create db – mysql specific

```
- name: create demo database
  mysql_db: name=demo state=present
    - name: create demo user
  mysql_user: name=demo password=demo priv=demo.*:ALL host='%'
state=present
```

Understand a playbook execution

- Executed in a top down approach
- Parses and builds an entire playbook before beginning an execution
- Order of execution of
  - Variable loading
  - Pre tasks
  - Roles
  - Tasks
  - Post_tasks
- Module + arguments + variable + ansible's code willl get loaded to memory as an object
- Ansible uses ssh connection and moves the object in memory to the host (temp dir)
- Then it executes the object memory, collects results and deletes the temp dir
- Results are provided as json data

Env setup Iteration : Loops with-item

```
- name: Install apache2
  apt: name={{ item }} state=present update_cache=yes
  with_items:
    - apache2
    - php5
    - libapache2-mod-php5
    - php5-mysql
```

Chapter: Handlers

Handlers

- Part of a basic event system in Ansible
- Just like tasks but require some other task to trigger using a notify
- Important for idempotence

```
# from with in a task
notify: restart apache2

# at the same level as a tasks
handlers:
  - name: restart apache2
    service: name=apache2 state=restarted
```

Chapter: Variables

Variables

- Variables are place holders of actual data just like any other systems
- Should always begin with a alpha character and can contain _ and numbers

variables in playbooks

```
- hosts: all
  vars_files:
    - vars.yml
  vars:
    message: Aditya
    fullmsg: "Hello {{ message }}"
  tasks:
    - debug: msg="{{ fullmsg }} {{ greeting }} {{ newvar }}
```

```
ansible-playbook debug.yml --extra-vars="newvar=newvalue"
ansible-playbook debug.yml --extra-vars="@var.json"
ansible-playbook debug.yml --extra-vars="@var.yml"
```

Variables – which are already available

- A few values are available to us which change based on remote machines

```
ansible -m setup ctrl >> /tmp/facts
```

```
- hosts: all
  vars_files:
    - [vars_{{ ansible_os_family }}.yml , vars_defalut.yml]
  tasks:
    - debug: msg="{{ ansible_os_family }} {{ greeting }}"
```

Variable files –conditional include

- Apache service is apache2 in debian and httpd in RedHad

```
- hosts: example
  vars_files:
    - [ "apache_{{ ansible_os_family }}.yml", "apache_default.yml" ]
  tasks:
    - service: name={{ apache }} state=running
```

Variables – From Inventory file

- From inventory
  - Infact, Ansible recommends not to store variables with inventory files

```
[ctrl]
control var1="From Inventory" api_version="2.3.4"

[ctrl:vars]
api_version="2.3.4"
```

Variables – group_vars and host_vars

- Variable files for hosts
  - /etc/ansible/host_vars/control → affects everybody
  - <current folder>/host_vars/control → playbook specific
- Variable files for groups
  - /etc/ansible/group_vars/ ctrl → affects everybody
  - <current folder>/group_vars/ctrl → playbook specific

Variables – Registered variables

- When we run a command we may need to store and use
  - Return code
  - Stdout
  - Stderr
- This is nice. Check the output for the below.

```
- hosts: ctrl
  tasks:
    - command: echo Hello There
      register: echo_out

    - name: Print echo_out
      debug: var=echo_out
```

Facts – A better look

- Ansible first gathers all information from each hosts
- That is the gathering facts section of the output you see while running ansible
- If facter or ohai is installed on the remote machine ansible adds them up too
- This is also called as gathering facts
- Gathering of facts can be turned on or off based on the need of the task

Prompt

- Possible to ask the user for input

```
- hosts: ctrl
  vars_prompt:
    - name: user_name
      prompt: “What is you name?”
      private: no
  tasks:
    - name: use user_name
      debug: var=user_name
```

Variable precedence

Lowest : role defaults [1]
inventory INI or script group vars [2]
inventory group_vars/all
playbook group_vars/all
inventory group_vars/*
playbook group_vars/*
inventory INI or script host vars [2]
inventory host_vars/*
playbook host_vars/*
host facts
play vars
play vars_prompt
play vars_files
role vars (defined in role/vars/main.yml)
block vars (only for tasks in block)
task vars (only for the task)
role (and include_role) params
include params
include_vars
set_facts / registered vars
Highest: extra vars (always win precedence)

Chapter: Jinja2 Templates

Jinja

- We have infact already used a lot of jinja
- Where ever we used a {{ }} - Mustache handle bar we have used jinja
- Jinjas true power lies in
  - If – else blocks
  - for loop control
  - filters

Chapter: Task Conditions

Controlling Task Conditionals

- Four statues of tasks – OK, Changed, Failed or Skipped
- Determines if further tasks should be executed or not
- Tasks can refer to the status of the previous task to control execution
- Two kinds of control
  - What defines a failure
  - What defines a change

ignore_errors

- A boolean value – true, yes, on, 1 and its capital variants

```
- name: “check status of apache2"
  command: service apache2 status
  ignore_errors: true
```

failed_when

- Much more fine grained. We can say when to consider it as a failure
- Note the change in syntax if I were to use directly use “error” in place of es in when condition

```
vars:
  es: error

. . .
. . .
. . .
-command: "echo Hi"
  register: output
  failed_when: es in output.stdout
```

When

- Run a task conditionally or skip running the task

```
- name: update if Debian
  apt: update_cache=yes
  when: ansible_os_family == 'Debian'

- name: update if RedHat
  yum: update_cache=yes
  when: ansible_os_family == 'RedHat'
```

When with registered variables

- Run a task conditionally or skip running the task
- The below is only an example. You can use “service” module which does the same

But expand this solution and imagine you have your own shell script to run and then check Its output and decide what to run next

```
- name: "update cache"
  command: service apache2 status
  register: output
  ignore_errors: true

- name: restart apache if output says not running
  service: name=apache2 state=started
  when: '"not running" in output.stdout'
```

Purely fail when!

- Many a times we will want to fail the playbook run purely if a previous command did not run well

```
- command: "echo error"
  register: output

- fail: msg="Something is not echoing"
  when: '"error" in output.stdout'
```

changed_when

- Similar to a task failure it is possible to define a “change”

```
- name: “update cache"
  apt: update_cache=yes
  changed_when: false
```

Conditionals

- If elif else structure.
- Resembles pythons structure except for endif
- Conditionals are same as Python conditionals

```
{% if env_type == "PROD" %}
Listen {{ server_port }}
{% else %}
Listen 80
{% endif %}
```

```
- template:
  src: 3.j2
  dest: /tmp/ifstatement.txt
```

jinja loops

- Iterative for – very similar to foreach
- Loops through and repeats the text block after variable sub out

```
{% if env_type == "PROD" %}
        {% for app_port in server_port_list %}
Listen {{ app_port }}
        {% endfor %}
{% else %}
Listen 80
{% endif %}
```

jinja filters

- Transform data in a template expression
- Documentation has a huge list of jinja template filters

```
{{ server_port | mandatory }}
{{ server_port | default(8) }}
{{ list | join(" ") }}
{{ myvar | type_debug }}
```

Chapter: Organizing playbooks

Organizing playbooks

- Modularizing playbooks
- Composing multiple playbooks to solve a infra problem
- Identifying the right variable set to be used
- The below makes it happen
  - 1. Includes
  - 2. Roles
  - 3. Role requirements or dependencies

Chapter: Includes

Including tasks

- Yaml files which have multiple tasks can be included in tasks

```
tasks:
  - name: First task
    debug:
      msg: "Just a free lying task"
  - include: other-tasks.yaml
```

```
tasks:
  - name: First task
    debug:
      msg: "Just a free lying task"
  - include: other-tasks.yaml
    server_port: "9091"
    env_type: "PROD"
```

Including tasks - conditionally

- Yaml files which have multiple tasks can be included in tasks
- IMPORTANT: This does not conditionally include the file. A file is always included

It only means that the when condition is applied to every of the tasks

```
tasks:
  - name: First task
    debug:
      msg: "Just a free lying task"
  - include: other-tasks.yaml
    when: ansible_os_family == 'Debian'
```

Including handlers

- Similar to inclusion of tasks
- Conditionals can be used and it works the same way it works with tasks
- Only difference being: no parameters can be passed

Chapter: Roles

Roles

- Taking all the inclusions a bit further is the concept of Roles
- Provides a way to build fully independent collection of variables tasks files templates and modules

Roles are not playbooks ! => I cannot execute a role directly

To execute a role , they have to be a part of a playbook

Roles Structure

- Roles have specified pre-determined structure
- All roles should be under a folder called roles
- Roles can further have specified subdirectories which are self explanatory
- Tasks : **roles/<role name>/tasks/main.yml**
- Handlers: **roles/<role name>/handlers/main.yml**
- Variables: **roles/<role name>/vars/main.yml**
- role defaults are from **roles/<role name>/defaults/main.yml**
- Files: **roles/<role name>/files/<filename>** **->** a filename can be just used without adding any prefixes. Further subdirectories can be used as relative reference

Templates: **roles/<role name>/templates/<templatename>**

Role play

- Role always requires a playbook – also called as application playbook or role application

```
- hosts: ctrl
  roles:
    - role: role1
```

Chapter: Ansible Galaxy

Ansible galaxy

- Role always requires a playbook – also called as application playbook or role application

```
>> ansible-galaxy install -p roles/ kunik.haproxy
>> ansible-galaxy init -p roles/ myownrole
```

Chapter: Create a role with ansible galaxy

Chapter: Vault

Ansible-vault

- A way to encrypt data
- Vault can encrypt
  - group_vars/ files
  - host_vars/ files
  - vars_files
  - role variables
  - Role defaults
  - Task files
  - Handler files

Anything represented as yaml can be encrypted with ansible-vault

But commonly used to encrypt passwords

Creating new encrypted files

```
Information needed
1. Password to use to encrypt
2. File to encrypt

export EDITOR=vi
# the below will ask for a password. mysecret.yml should not exists
>> ansible-vault create mysecret.yml

# password can come from a file too. mysecret.yml should not exists
>> echo ansible > my_vault_pass.txt
>> ansible-vault create mysecret.yml --vault-password-file
my_vault_pass.txt

# password can be output from an executable too. mysecret.yml should
not exists
>> ansible-vault create mysecret.yml --vault-password-file
password.sh
```

Encrypting existing files

```
>> ansible-vault encrypt mysecret.yml

# password can be fed from a file too
>> echo ansible > my_vault_pass.txt
>> ansible-vault encrypt mysecret.yml --vault-password-file
my_vault_pass.txt

# password can be output from an executable too
>> ansible-vault encrypt mysecret.yml --vault-password-file
password.sh
```

editing encrypted files

```
>> ansible-vault edit mysecret.yml

# password can be fed from a file too
>> echo ansible > my_vault_pass.txt
>> ansible-vault edit mysecuret.yml --vault-password-file
my_vault_pass.txt

# password can be output from an executable too
>> ansible-vault edit mysecuret.yml --vault-password-file
password.sh
```

Executing a playbook with encryption

```
>> ansible-playbook myplay.yml --ask-vault-pass

# password can be fed from a file too
>> echo ansible > my_vault_pass.txt
>> ansible-playbook myplay.yml --vault-password-file
my_vault_pass.txt

# password can be output from an executable too
>> ansible-playbook myplay.yml --vault-password-file password.sh
```

A tip about encrypted data

- Encryption severely hinders readability
- Do not encrypt the whole file – unless un till it warrants
- You can try the below way

```
File1.yml
secret_db_pass: demo_user

Encrypt file1.yml. Because the file is encrypted we will not even
know that it contains a variable name called secret_db_pass. This
hinders readablity

File2.yml
db_pass: "{{ secret_db_pass }}"

use variable name db_pass in your playbooks
```

Chapter: Troubleshooting tips

Trouble shooting tips

- Increase verbosity with
- Levels of verbosity
  - None: default level of verbosity
  - -v : one v where return data is displayed
  - -vv : input data aswell
  - -vvv: details of connection attempts
  - -vvvv: will pass extra verbosity to underlying connection modules
- Logging:
  - By default logs to console (or stdout)
  - set “log_path” as the log file name in ansible.cfg
  - Having a log file will put the stdout to a log file as well as console
- Debug debug debug: Debug module solves a lot of issues

Trouble shooting tips - continued

- If the playbook is huge you can start the execution at

```
ansible-playbook playbook.yml --start-at-task="install packages"
```

- Execute ansible-playbook step by step aks interactively

```
ansible-playbook playbook.yml --step
```

Chapter: Role Req to manage dependency

Chapter: Delegation

Chapter: Local actions

Chapter: Pauses