K8S Administration

Introduction to Ansible

Ansible

Understanding YAML

Understanding YAML

- YAML uses simple key-value pair to represent the data. The dictionary is represented in key: value pair.

- Note − There should be space between : and value.

- Example: A student record
- --- #Optional YAML start syntax

- james:
- name: james john
- rollNo: 34
- div: B
- sex: male

Understanding YAML

- Abbreviation
- You can also use abbreviation to represent dictionaries.

- Example
- James: {name: james john, rollNo: 34, div: B, sex: male}

Understanding YAML

- We can also represent List in YAML. Every element(member) of list should be written in a new line with same indentation starting with “- “ (- and space).

- Example
- ---
- countries:
- - America
- - China
- - Canada
- - Iceland
- …

Understanding YAML

- Abbreviation
- You can also use abbreviation to represent lists.

- Example
- Countries: [‘America’, ‘China’, ‘Canada’, ‘Iceland’]

Understanding YAML

- List inside Dictionaries
- We can use list inside dictionaries, i.e., value of key is list.

- Example
- ---
- james:
- name: james john
- rollNo: 34
- div: B
- sex: male
- likes:
- - maths
- - physics
- - english

Understanding YAML

- List inside Dictionaries
- We can use list inside dictionaries, i.e., value of key is list.

- Example
- ---
- james:
- name: james john
- rollNo: 34
- div: B
- sex: male
- likes:
- - maths
- - physics
- - english

Ansible

Usage

What is Ansible?

It's a **simple automation language** that can perfectly describe an IT application infrastructure in Ansible Playbooks.

It's an **automation engine** that runs Ansible Playbooks.

Ansible Tower is an **enterprise framework** for controlling, securing and managing your Ansible automation with a **UI and RESTful API**.

SIMPLE

Human readable automation

No special coding skills needed

Tasks executed in order

Get productive quickly

POWERFUL

App deployment

Configuration management

Workflow orchestration

Orchestrate the app lifecycle

AGENTLESS

Agentless architecture

Uses OpenSSH & WinRM

No agents to exploit or update

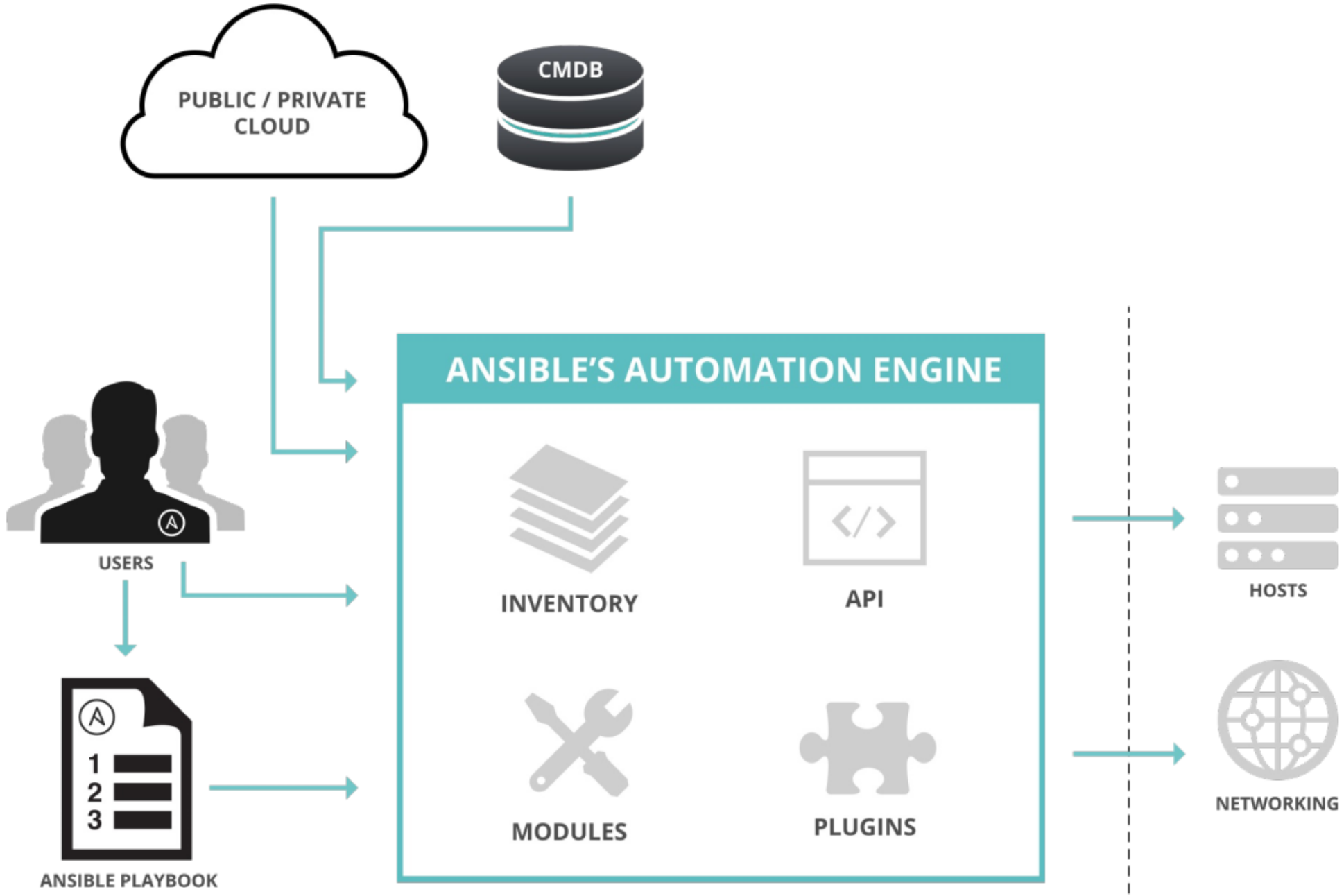
More efficient & more secure

Terms

- Some common words related to Ansible.
- Service/Server – A process on the machine that provides the service.
- Machine – A physical server, vm(virtual machine) or a container.
- Target machine – A machine we are about to configure with Ansible.
- Task – An action(run this, delete that) etc managed by Ansible.
- Playbook – The yml file where Ansible commands are written and yml is executed on a machine.

Installation

- Mainly, there are two types of machines when we talk about deployment –
- **Control machine** – Machine from where we can manage other machines.
- **Remote machine** – Machines which are handled/controlled by control machine.


PUBLIC / PRIVATE CLOUD
CMDB
ANSIBLE'S AUTOMATION ENGINE
USERS
INVENTORY
API
HOSTS
1
2
3
ANSIBLE PLAYBOOK
MODULES
PLUGINS
NETWORKING

Modules

- Modules are bits of code transferred to the target system and executed to satisfy the task declaration. Ansible ships with several hundred today!
  - apt/yum
  - copy
  - file
  - get_url
  - git
  - ping
  - debug
  - service
  - synchronize
  - template

Modules Documentation

- # List out all modules installed

  ansible-doc -l

  copy

  cron

  ...

- # Read documentation for installed module

  ansible-doc copy

> COPY

The [copy] module copies a file on the local box to remote locations. Use the [fetch] module to copy files from remote locations to the local box. If you need variable interpolation in copied files, use the [template] module.

* note: This module has a corresponding action plugin.

Options (= is mandatory):

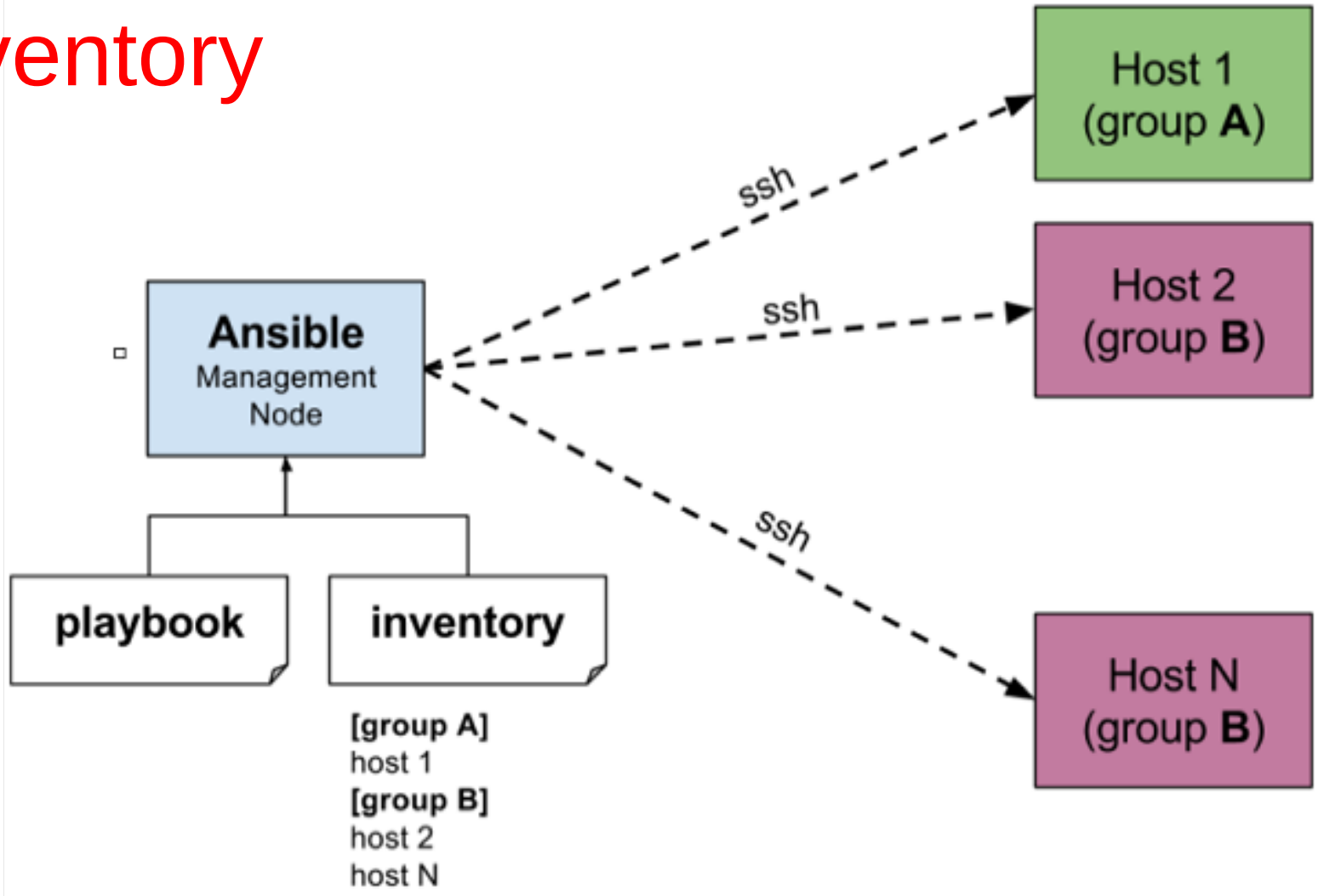
Modules: Run Commands

- If Ansible doesn't have a module that suits your needs there are the "run command" modules:
- command: Takes the command and executes it on the host. The most secure and predictable.
- shell: Executes through a shell like /bin/sh so you can use pipes etc. Be careful.
- script: Runs a local script on a remote node after transferring it.
- raw: Executes a command without going through the Ansible module subsystem.
- NOTE: Unlike standard modules, run commands have no concept of desired state and should only be used as a last resort

Inventory

- Inventory is a collection of hosts (nodes) with associated data and groupings that Ansible can connect and manage.
  - Hosts (nodes)
  - Groups
  - Inventory-specific data (variables)
  - Static or dynamic sources

Inventory


Host 1
(group A)
ssh
Host 2
(group B)
ssh
Ansible
Management
Node
ssh
playbook
inventory
Host N
(group B)
[group A]
host 1
[group B]
host 2
host N

Static Inventory Example

- 10.42.0.2
- 10.42.0.6
- 10.42.0.7
- 10.42.0.8
- 10.42.0.100
- host.example.com

Static Inventory Example with group

- [control]

  control ansible_host=10.42.0.2
- [web]

  node-[1:3] ansible_host=10.42.0.[6:8]
- [haproxy]

  haproxy ansible_host=10.42.0.100
- [all:vars]

  ansible_user=vagrant

  ansible_ssh_private_key_file=~/.vagrant.d/insecure_private_key

Ad-Hoc Commands

- An ad-hoc command is a single Ansible task to perform quickly, but don’t want to save for later.

check all my inventory hosts are ready to be managed by Ansible

```
ansible all -m ping
```

collect and display the discovered facts for the localhost

```
ansible localhost -m setup
```

run the uptime command on all hosts in the web group

```
ansible web -m command -a "uptime"
```

Ad-Hoc Commands

- An ad-hoc command is a single Ansible task to perform quickly, but don’t want to save for later.

To run reboot for all your company servers in a group, 'abc', in 12 parallel forks –

```
ansible abc –m command  -a "/sbin/reboot" -f 12
```

Transferring file to many servers/machines

```
ansible abc -m copy -a "src = /etc/yum.conf dest = /tmp/yum.conf"
```

The following command checks if yum package is installed or not, but does not update it.

```
ansible abc -m yum -a "name = demo-tomcat-1 state = present"
```

Sidebar: Discovered Facts

- Facts are bits of information derived from examining a host systems that are stored as variables for later use in a play.

```
ansible localhost -m setup
localhost | success >> {
        "ansible_facts": {
        "ansible_default_ipv4": {
        "address": "192.168.1.37",
        "alias": "wlan0",
        "gateway": "192.168.1.1",
        "interface": "wlan0",
        "macaddress": "c4:85:08:3b:a9:16",
        "mtu": 1500,
        "netmask": "255.255.255.0",
        "network": "192.168.1.0",
        "type": "ether"
},
```

Variables

- Ansible can work with metadata from various sources and manage their context in the form of variables.
  - Command line parameters
  - Plays and tasks
  - Files
  - Inventory
  - Discovered facts
  - Roles

Defining and Using Variables

You can define a simple variable using standard YAML syntax.

```
remote_install_path: /opt/my_app_config
```

After you define a variable, use Jinja2 syntax to reference it. Jinja2 variables use double curly braces. For example, the expression My amp goes to {{ max_amp_value }} demonstrates the most basic form of variable substitution. You can use Jinja2 syntax in playbooks. For example:

```
ansible.builtin.template:
    src: foo.cfg.j2
    dest: '{{ remote_install_path }}/foo.cfg'
```

Defining and Using Variables

You can define variables with multiple values using YAML lists. For example:

```
region:
  - northeast
  - southeast
  - midwest
```

When you use variables defined as a list (also called an array), you can use individual, specific fields from that list. The first item in a list is item 0, the second item is item 1. For example:

```
region: "{{ region[0] }}"
```

Referencing nested variables

Many registered variables (and facts) are nested YAML or JSON data structures. You cannot access values from these nested data structures with the simple {{ foo }} syntax. You must use either bracket notation or dot notation. For example, to reference an IP address from your facts using the bracket notation:

{{ ansible_facts["eth0"]["ipv4"]["address"] }}

To reference an IP address from your facts using the dot notation:

{{ ansible_facts.eth0.ipv4.address }}

Using variables: inventory

host or group variables can be defined within inventory file

/etc/ansible/host_vars/hostname/*.yaml

/etc/ansible/group_vars/group_name/*.yaml

Variables

```
block:
  - name: Install Tomcat artifacts
    action:
    yum name = "demo-tomcat-1" state = present
    register: Output

  always:
    - debug:
      msg:
        - "Install Tomcat artifacts task ended with message: {{Output}}"
        - "Installed Tomcat artifacts - {{Output.changed}}"
```

Tasks

- Tasks are the application of a module to perform a specific unit of work.
  - file: A directory should exist
  - yum: A package should be installed
  - service: A service should be running
  - template: Render a configuration file from a template
  - get_url: Fetch an archive file from a URL
  - git: Clone a source code repository

Example Tasks in a Play

```
tasks:
        - name: httpd package is present
                yum:
                        name: httpd
                        state: latest
        - name: latest index.html file is present
                copy:
                        src: files/index.html
                        dest: /var/www/html/
        - name: restart httpd
                service:
                        name: httpd
                        state: restarted
```

Handler Tasks

- Handlers are special tasks that run at the end of a play if notified by another task when a change occurs.
- If a configuration file gets changed notify a service restart task that it needs to run tasks:

Example Handler Task in a Play

```
tasks:
        - name: httpd package is present
                yum:
                        name: httpd
                        state: latest
                notify: restart httpd
        - name: latest index.html file is present
                copy:
                        src: files/index.html
                        dest: /var/www/html/
        handlers:
        - name: restart httpd
                service:
                        name: httpd
                        state: restarted
```

Plays & Playbooks

- Plays are ordered sets of tasks to execute against host selections from your inventory.
- A playbook is a file containing one or more plays.

Playbook Example

```
---
        - name: install and start apache
                hosts: web
                become: yes
                vars:
                        http_port: 80
        tasks:
        - name: httpd package is present
                yum:
                        name: httpd
                        state: latest
        - name: latest index.html file is present
                copy:
                        src: files/index.html
                dest: /var/www/html/
```

Doing More with Playbooks

- Here are some more essential playbook features that you can apply:
  - Templates
  - Loops
  - Conditionals
  - Tags
  - Blocks

Templates

- Ansible embeds the that can be used to dynamically:
- Set and modify play variables
- Conditional logic
- Generate files such as configurations from variables

Templates

- Templates allow you to create new files on the nodes using predefined models based on the Jinja2 templating system.
- Ansible templates are typically saved as .tpl files and support the use of variables, loops, and conditional expressions.
- Templates are commonly used to configure services based on variable values that can be set up on the playbook itself, in included variable files, or obtained via facts.
- This enables you to create more versatile setups that adapt behavior based on dynamic information.

Templates

- General purpose template language
- Python library

- Delimiters
- {% expression %} → logic
- {{ expression }} → output results
- {# expression #} → comment

Template example

- <!doctype html>
- <html lang="en">
- <head>
- <meta charset="utf-8">
- <title>{{ page_title }}</title>
- <meta name="description" content="Created with Ansible">
- </head>
- <body>
- <h1>{{ page_title }}</h1>
- {{ page_description }}
- </body>
- </html>

Template usage

- This template uses two variables that must be provided whenever the template is applied in a playbook: page_title and page_description.

- The following playbook sets up the required variables, installs Nginx, and then applies the specified template to replace the existing, default Nginx landing page located at /var/www/html/index.nginx-debian.html.
- The last task uses the ufw module to enable tcp access on port 80, in case you have your firewall enabled as recommended in our initial server setup guide.

Template usage

```
---
- hosts: all
  become: yes
  vars:
    page_title: My Landing Page
    page_description: This is my landing page description.
  tasks:
    - name: Install Nginx
      apt:
        name: nginx
        state: latest
    - name: Apply Page Template
      template:
        src: files/landing-page.html.j2
        dest: /var/www/html/index.nginx-debian.html
    - name: Allow all access to tcp port 80
      ufw:
        rule: allow
        port: '80'
        proto: tcp
```

Loops

- Loops can do one task on multiple things, such as create a lot of users, install a lot of packages, or repeat a polling step until a certain result is reached.

```
- yum:
        name: "{{ item }}"
        state: latest
        with_items:
                - httpd
                - mod_wsgi
```

Loops

- Avoid loops where possible
- When module supports lists as input parameter
- package, dnf, apt, apk, etc . . .
- Standard loops over a list → with_items
- Loop over a nested list → with_nested
- Loop over a dictionary → with_dict
- with_file → over contents of file list
- with_fileglob → over matched file list
- with_sequence → over number sequence

Loops

- Loops can do one task on multiple things, such as create a lot of users, install a lot of packages, or repeat a polling step until a certain result is reached.

```
tasks:
    - name: Install Apache
    shell: "ls *.war"
    register: output
    args:
      chdir: /opt/ansible/tomcat/demo/webapps

    - file:
      src: '/opt/ansible/tomcat/demo/webapps/{{ item }}'
      dest: '/users/demo/vivek/{{ item }}'
      state: link
    with_items: "{{output.stdout_lines}}"
```

Conditionals

- when → only perform task if condition is true
- when processed for each item in loop
- Can be used with
- import*, include*
- roles
- fail, and assert exit play based on conditions

Conditionals

- When you add a conditional to an import statement, Ansible applies the condition to all tasks within the imported file. This behavior is the equivalent of Tag inheritance: adding tags to multiple tasks. Ansible applies the condition to every task and evaluates each task separately. For example, if you want to define and then display a variable that was not previously defined, you might have a playbook called main.yml and a tasks file called other_tasks.yml:

- # all tasks within an imported file inherit the condition from the import statement
- - hosts: all
- tasks:
- - import_tasks: other_tasks.yml # note "import"
- when: x is not defined

Tags

- Tags are useful to be able to run a subset of a playbook on-demand.

```
- yum:
        name: "{{ item }}"
        state: latest
  with_items:
        - httpd
        - mod_wsgi
  tags:
        - packages
- template:
        src: templates/httpd.conf.j2
        dest: /etc/httpd/conf/httpd.conf
  tags:
        - configuration
```

Tags usage

- The following command helps in using tags

  ansible-playbook -i hosts <your yaml> --tags "install" -vvv

Exception Handling in Playbooks

```
tasks:
  - name: Name of the task to be executed
    block:
      - debug: msg = 'Just a debug message , relevant for logging'
      - command: <the command to execute>

    rescue:
      - debug: msg = 'There was an exception.. '
      - command: <Rescue mechanism for the above exception occurred)
always:
      - debug: msg = "this will execute in all scenarios. Always will get logged"
```

Roles

- In Ansible, the role is the primary mechanism for breaking a playbook into multiple files. This simplifies writing complex playbooks, and it makes them easier to reuse. The breaking of playbook allows you to logically break the playbook into reusable components.
- Roles are not playbooks. Roles are small functionality which can be independently used but have to be used within playbooks. There is no way to directly execute a role. Roles have no explicit setting for which host the role will apply to.
- Top-level playbooks are the bridge holding the hosts from your inventory file to roles that should be applied to those hosts.

```
$ ansible-galaxy init vivekrole
```

```
- vivekrole was created successfully

$ tree vivekrole/
vivekrole/
├── defaults
│   └── main.yml
├── files ├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md ├── tasks
│   └── main.yml
├── templates ├── tests │   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
```

Utilizing Roles in Playbook

- This is the code of the playbook we have written for demo purpose. This code is of the playbook vivek_orchestrate.yml. We have defined the hosts: tomcat-node and called the two roles – install-tomcat and start-tomcat.
- The problem statement is that we have a war which we need to deploy on a machine via Ansible.

```
---
        - hosts: tomcat-node
        roles:
                - {role: install-tomcat}
                - {role: start-tomcat}
```

Configuring Ansible Vault

- The ansible-vault encrypt_string command encrypts and formats any string you type (or copy or generate) into a format that can be included in a playbook, role, or variables file. To create a basic encrypted variable, pass three options to the ansible-vault encrypt_string command:

- a source for the vault password (prompt, file, or script, with or without a vault ID)
- the string to encrypt
- the string name (the name of the variable)

Configuring Ansible Vault

- ansible-vault
- create → new encrypted file
- encrypt_string → STDIN/STDOUT or argument
- {en,de}crypt → file
- view → decrypt to STDOUT
- edit → decrypt, edit, encrypt
- rekey → decrypt, encrypt with new password

Configuring Ansible Vault

- The pattern looks like this:
- For example, to encrypt the string ‘foobar’ using the only password stored in ‘a_password_file’ and name the variable ‘the_secret’:
- ansible-vault encrypt_string --vault-password-file a_password_file 'foobar' --name 'the_secret'

- The command above creates this content:
- the_secret: !vault |
- $ANSIBLE_VAULT;1.1;AES256
- 623133653966623430613934643361633837643737646136336536343062313864336264366233 61
- 613433366535396636353433363266653533376166613162 0a663537646436643839616531643561
- 633962653339663861663736326265393261663539653632626330303336303133386463353036 30
- 343862666666137650a353638643435666633633964366338633066623234616432373231333331
- 6564

Questions?

andry.cheredarchuk@gmail.com