TÌM HIỂU VỀ CÁCH THỨC HOẠT ĐỘNG CỦA ANSIBLE

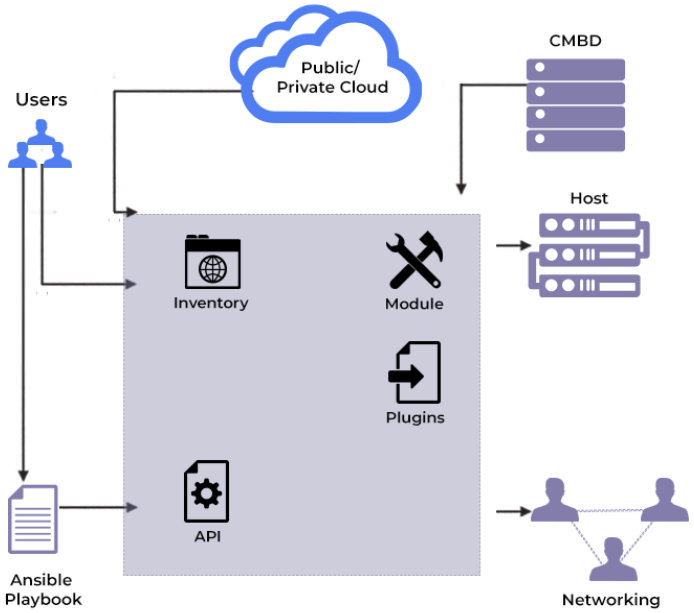
Thực hiện: Bùi Hoàng Vinh

Hà Nội - 2024

MỤC LỤC

Chương 1. Kiến trúc Ansible

Ansible là một công cụ tự động hóa đơn giản, tự động hóa việc quản lý cấu hình, triển khai ứng dụng, điều phối nội bộ dịch vụ và nhiều nhu cầu CNTT khác.



Thành phần tạo nên kiến trúc của Ansible:

Control Node: Control node là máy chủ nơi Ansible được cài đặt và từ đó các lệnh Ansible được chạy. Đây là điểm trung tâm quản lý và điều khiển tất cả các tác vụ và playbook.

Managed Nodes: Managed nodes là các máy chủ hoặc thiết bị được quản lý bởi Ansible. Ansible thực hiện các tác vụ trên các managed nodes thông qua SSH (hoặc giao thức khác nếu cần).

Inventory: Inventory là danh sách các managed nodes mà Ansible sẽ quản lý. Inventory có thể được định nghĩa trong các file dạng text hoặc được tạo động từ các nguồn khác (dynamic inventory).

Modules: Ansible kết nối các node và thực hiện các chương trình Modules Ansible. Ansible thực thi các Modules và loại bỏ sau khi hoàn thành. Các Modules này có thể nằm trên bất kỳ máy nào. Có thể làm việc với trình soạn thảo văn bản đã chọn hoặc hệ thống kiểm soát phiên bản hoặc thiết bị đầu cuối để theo dõi những thay đổi trong nội dung.

Plugins: Plugins mở rộng chức năng của Ansible, bao gồm các loại như callback, connection, lookup, và vars plugins. Chúng cung cấp các tính năng bổ sung như quản lý kết nối, tìm kiếm dữ liệu, và xử lý biến.

Playbooks: Playbooks là các file YAML định nghĩa các tác vụ cần thực hiện trên các managed nodes. Một playbook có thể chứa nhiều "plays", mỗi play định nghĩa các tác vụ cụ thể và các nhóm máy chủ mà các tác vụ này sẽ được thực hiện.

Roles: Roles giúp tổ chức và tái sử dụng cấu hình và tác vụ bằng cách gom nhóm các biến, files, templates, tasks và handlers vào trong các thư mục riêng biệt.

Networking: Ansible được sử dụng để tự động hóa các mạng khác nhau, an toàn và mạnh mẽ cho các hoạt động và phát triển CNTT. Nó sử dụng một loại mô hình dữ liệu tách biệt với công cụ tự động hóa Ansible, từ đó dễ dàng mở rộng phần cứng khác nhau.
Cloud: Private hoặc public cloud là tập hợp các máy chủ ở xa mà có thể sử dụng để thu thập, sắp xếp và xử lý thông tin. Thay vì lưu giữ dữ liệu trên máy chủ cục bộ, các hệ thống này được lưu trữ qua internet, triển khai các tài nguyên, liên kết chúng với cơ sở dữ liệu
API: API Ansible hoạt động như phương tiện vận chuyển cho các dịch vụ private hoặc public cloud
CMDB: CMDB là một loại kho lưu trữ hoạt động như một kho dữ liệu cho quá trình cài đặt CNTT.

Chương 2. Xây dựng Ansible inventory

Ansible inventory được sử dụng để định nghĩa các máy chủ được quản lý mà Ansible triển khai và định cấu hình

2.1. File Inventory cơ bản

File Inventory có thể sử dụng nhiều format khác nhau, phổ biến là INI và YAML, ví dụ về 1 cấu trúc cơ bản của file inventory tĩnh (file ini):

```
[webservers]
web1.example.com
web2.example.com

[dbservers]
db1.example.com
db2.example.com
```

Group là cách tổ chức các máy chủ (hosts) thành các nhóm logic để dễ quản lý và áp dụng các cấu hình hoặc playbook. Groups có thể được định nghĩa trong file inventory, và có thể sử dụng các nhóm này để áp dụng cấu hình chung cho một tập hợp các máy chủ. Đồng thời có thể định nghĩa các biến áp dụng cho toàn bộ nhóm

```
[webservers]
web1.example.com
web2.example.com

[dbservers]
db1.example.com
db2.example.com

[all:vars]
ansible_user=your_username
ansible_ssh_private_key_file=/path/to/your/private/key
```

Mỗi host có thể thuộc nhiều group khác nhau, thường được sử dụng khi cần áp dụng các cấu hình hoặc tác vụ khác nhau từ nhiều nhóm đến cùng một máy chủ
Việc định nghĩa group giúp sử dụng Playbook trở nên thuận tiện hơn khi thiết lập các tác vụ cho 1 nhóm host

```
- name: Deploy web servers
  hosts: webservers
  tasks:
    - name: Ensure Apache is installed
      apt:
        name: apache2
        state: present
```

Ví dụ trên về việc thực hiện tác vụ đảm bảo Apache được cài đặt trên tất cả các host thuộc group webservers

2.2. File Inventory động

File Inventory Động (Dynamic Inventory) trong Ansible cho phép tự động lấy thông tin về các máy chủ từ một nguồn bên ngoài thay vì phải duy trì một file inventory tĩnh. Điều này đặc biệt hữu ích khi làm việc với các môi trường cloud hoặc các hệ thống có thay đổi liên tục về số lượng và cấu hình máy chủ.

```
plugin: aws_ec2
regions:
  - us-east-1
filters:
  tag:Environment: production
hostnames:
  - tag:Name
compose:
  ansible_host: public_ip_address
groups:
  webservers: "'web' in tags['Role']"
  dbservers: "'db' in tags['Role']”
```

Trong ví dụ trên đã sử dụng bộ lọc chỉ lấy các máy chủ có tag Environment: production, sử dụng giá trị Name làm hostname cho các máy chủ, Xác định rằng ansible_host sẽ sử dụng public_ip_address, và tạo các nhóm webservers và dbservers dựa trên giá trị của tag Role. Đồng thời có thể tự động cập nhật thông tin về các máy chủ phù hợp với các bộ lọc đã định nghĩa trong file cấu hình:

```
ansible-inventory -i aws_ec2.yml --list
```

Ưu Điểm

- Tự động cập nhật thông tin về các máy chủ, giúp quản lý môi trường thay đổi liên tục một cách hiệu quả.
- Dễ dàng tích hợp với các dịch vụ đám mây và hệ thống quản lý khác, giúp quản lý các tài nguyên động.
- Thích hợp cho các hệ thống lớn với nhiều máy chủ, giúp giảm tải công việc quản trị.

Nhược Điểm

- Cấu hình ban đầu phức tạp hơn so với inventory tĩnh, cần hiểu biết về các plugin và script động.

Chương 3: Cách thức hoạt động của Ansible playbooks

3.1. Tổng quát về Ansible playbooks

Playbooks được thể hiện với định dạng YAML. Một playbook bao gồm một hoặc nhiều kịch bản thực thi trong một danh sách có thứ tự. Mỗi play thực hiện một phần mục tiêu tổng thể của playbook, thực hiện một hoặc nhiều task. Mỗi task gọi một Ansible module. Mỗi play cần định nghĩa 2 việc: các node mục tiêu (sử dụng pattern để chỉ định) và các task thực thi trên các node mục tiêu đó.

Thực thi tác vụ (task): Theo mặc định, Ansible thực thi từng tác vụ theo thứ tự, từng tác vụ một, đối với tất cả các máy phù hợp với mẫu máy chủ. Mỗi tác vụ thực thi một mô-đun với các đối số cụ thể. Khi một tác vụ được thực thi trên tất cả các máy mục tiêu, Ansible sẽ chuyển sang tác vụ tiếp theo, có thể sử dụng các chiến lược để thay đổi hành vi mặc định này. Trong mỗi play, Ansible áp dụng các chỉ thị nhiệm vụ giống nhau cho tất cả các máy chủ.

Chạy playbooks trong check mode Chế độ kiểm tra của Ansible cho phép thực thi một playbook mà không cần áp dụng bất kỳ thay đổi nào đối với hệ thống của mình. Có thể sử dụng chế độ kiểm tra để kiểm tra playbook trước khi triển khai chúng trong môi trường production, truyền thêm cờ -C hoặc --check cho lệnh ansible-playbook:

```
ansible-playbook --check playbook.yaml
```

Việc thực thi lệnh này sẽ chạy playbook một cách bình thường, nhưng thay vì thực hiện bất kỳ sửa đổi nào, Ansible sẽ chỉ cung cấp một báo cáo về những thay đổi mà nó sẽ thực hiện.

Ansible-Pull là một công cụ trong Ansible cho phép thực thi các playbook từ một repository trên máy đích, thay vì từ máy điều khiển như trong trường hợp sử dụng ansible command thông thường. Khi sử dụng Ansible-Pull, máy đích sẽ tự động kéo các cấu hình và playbook từ một kho lưu trữ như Git và sau đó thực thi chúng trên chính máy đó.

3.2. Làm việc với Ansible playbooks

3.2.1. Template (Jinja2)

Ansible sử dụng template Jinja2 để kích hoạt các biểu thức động và truy cập vào các ansible variables và ansible facts (ví dụ ansible_facts['hostname']).

Giả sử muốn tạo một file cấu hình cho ứng dụng web, trong đó cần phải thay đổi một số giá trị như địa chỉ IP của máy chủ cơ sở dữ liệu và cổng mạng. Thay vì tạo file cấu hình riêng cho từng máy chủ mỗi khi triển khai, có thể sử dụng Ansible template để tạo ra một file cấu hình tùy biến dựa trên các biến được định nghĩa trong playbook.

```
# File web_app.conf.j2
server {
    listen {{ web_app_port }};
    server_name {{ web_app_domain }};

    location / {
        proxy_pass http://{{ db_server_ip }}:{{ db_server_port }};
    }
}
```

```
# File deploy_web_app.yml
---
- name: Deploy web application configuration
  hosts: web_servers
  vars:
    web_app_domain: example.com
    db_server_port: 3306
  tasks:
    - name: Create web app configuration file
      template:
        src: web_app.conf.j2
        dest: /etc/nginx/conf.d/web_app.conf
      notify: Reload Nginx
```

Trong ví dụ trên giả sử mỗi host sẽ có 1 db_server_ip khác nhau, được cấu hình trong thư mục vars hoặc được cấu hình trong file inventory. Việc sử dụng template giúp tự động hóa quá trình tạo ra các file cấu hình và tăng khả năng tái sử dụng trong nhiều ngữ cảnh khác nhau.

3.2.2. Sử dụng Filter để thao tác dữ liệu

Filter cho phép chuyển đổi dữ liệu JSON thành dữ liệu YAML, tách URL để trích xuất tên máy chủ, lấy hàm băm SHA1 của chuỗi, cộng hoặc nhân số nguyên, v.v. để có thể thao tác với dữ liệu. Vì việc tạo khuôn mẫu xảy ra trên node điều khiển Ansible chứ không phải trên host đích nên các bộ lọc sẽ thực thi trên node điều khiển và chuyển đổi dữ liệu cục bộ.

Một số Filter phổ biến sử dụng

- Chuyển đổi chuỗi thành chữ hoa

```
- debug:
    msg: "{{ my_string | upper }}"
```

- Chuyển đổi chuỗi thành số nguyên

```
- debug:
    msg: "{{ my_string | int }}"
```

- Lọc các phần tử trong danh sách theo một điều kiện

```
- debug:
    msg: "{{ my_string | upper }}"
```

- Tìm kiếm các đối tượng trong danh sách có thuộc tính nhất định

```
- debug:
    msg: "{{ my_list | selectattr('age', '==', 30) | list }}"
```

- Sắp xếp một danh sách

```
- debug:
    msg: "{{ my_list | sort }}"
```

- Lấy phần tử đầu tiên của danh sách

```
- debug:
    msg: "{{ my_list | first }}"
```

3.2.3. Lookups

Các plugin Lookups truy xuất dữ liệu từ các nguồn bên ngoài như file, cơ sở dữ liệu, API và các dịch vụ khác. Giống như tất cả các khuôn mẫu, việc tra cứu sẽ thực thi và được đánh giá trên máy điều khiển Ansible.

Ví dụ về Lookup thông tin từ file, sử dụng lookup 'file' để lấy nội dung của 1 file

```
- debug:
    msg: "{{ lookup('file', '/path/to/file.txt') }}"
```

Ví dụ về Lookup Thông Tin Từ Một Cơ Sở Dữ Liệu (Database), sử dụng lookup sql để truy vấn thông tin từ cơ sở dữ liệu.

```
- debug:
    msg: "{{ lookup('sql', 'SELECT * FROM table_name',
dsn='mysql://user:password@localhost/database') }}"
```

3.2.4. Loops

Trong Ansible, việc sử dụng vòng lặp để lặp lại các tác vụ hoặc các phần của playbook trên một danh sách các phần tử. Điều này giúp tự động hóa việc thực hiện các tác vụ lặp lại trên nhiều máy chủ hoặc với nhiều cấu hình khác nhau.

Ví dụ sử dụng Ansible tạo ra một tệp cấu hình virtual host cho mỗi mục trong danh sách virtual_hosts, sử dụng template "virtualhost.conf.j2", và lưu chúng vào thư mục "/etc/httpd/conf.d/" trên host mục tiêu

```
- name: Configure multiple virtual hosts
  template:
    src: "virtualhost.conf.j2"
    dest: "/etc/httpd/conf.d/{{ item.name }}.conf"
  loop: "{{ virtual_hosts }}"
```

loop được sử dụng để chỉ ra danh sách các mục muốn lặp lại. Mỗi mục trong danh sách có thể là một chuỗi, một số, một dictionary, hoặc thậm chí là một biến Ansible. Ansible sẽ thực hiện các tác vụ được chỉ định bên trong khối lặp với mỗi mục trong danh sách.

3.2.5. Kiểm soát nơi các tác vụ được thực thi với Delegation và local actions

Trong Ansible, có hai cách chính để kiểm soát nơi các tác vụ được thực thi là sử dụng delegation và local actions.

Theo mặc định, Ansible thu thập dữ kiện và thực thi tất cả tác vụ trên các máy phù hợp với biến hosts định nghĩa trong playbook. Ngoài ra, Ansible có thể ủy quyền nhiệm vụ cho một máy hoặc nhóm khác, ủy quyền dữ liệu cho các máy hoặc nhóm cụ thể hoặc chạy toàn bộ playbook cục bộ

Ví dụ: khi cập nhật webservers, có thể cần tạm thời xóa chúng khỏi nhóm cân bằng tải và không thể thực hiện tác vụ này trên chính webservers.

```
---
- hosts: webservers
  serial: 5

  tasks:
    - name: Take out of load balancer pool
      ansible.builtin.command: /usr/bin/take_out_of_pool {{ inventory_hostname }}
      delegate_to: 127.0.0.1

    - name: Actual steps would go here
      ansible.builtin.yum:
        name: acme-web-stack
        state: latest

    - name: Add back to load balancer pool
      ansible.builtin.command: /usr/bin/add_back_to_pool {{ inventory_hostname }}
      delegate_to: 127.0.0.1
```

Do task thứ nhất và thứ ba trong play chạy trên IP 127.0.0.1, là node đang chạy Ansible. có thể sử dụng từ khóa local_action

```
---
# ...

  tasks:
    - name: Take out of load balancer pool
      local_action: ansible.builtin.command /usr/bin/take_out_of_pool {{
inventory_hostname }}

# ...

    - name: Add back to load balancer pool
      local_action: ansible.builtin.command /usr/bin/add_back_to_pool {{
inventory_hostname }}
```

3.2.6. Conditionals

Trong playbook, có thể muốn thực thi các nhiệm vụ khác nhau hoặc có các mục tiêu khác nhau, tùy thuộc vào giá trị của một fact (dữ liệu về hệ thống từ xa), một biến

hoặc kết quả của nhiệm vụ trước đó. Có thể muốn giá trị của một số biến phụ thuộc vào giá trị của các biến khác.

- Viết câu điều kiện dựa trên ***ansible_facts***. Ví dụ chỉ muốn thực hiện tác vụ sử dụng gói apt trên hệ điều hành là Ubuntu, và áp dụng gói dnf cho CentOS.

```
# Ubuntu
- name: update repository index (Ubuntu)
  apt:
    update_cache: true
  when: ansible_facts['distribution'] == "Ubuntu"
```

```
# CentOS
- name: Install a package (CentOS 8)
  dnf:
     name: my_package
     state: present
    when: ansible_facts['distribution'] == 'CentOS' and
ansible_facts['distribution_major_version'] == '7'
```

- Viết câu điều kiện dựa trên ***registered variables***. Ví dụ về việc kiểm tra sự tồn tại của file docker-compose.yml, trước khi thực hiện docker compose up/down.

```
- name: Check if docker-compose.yml exists
  stat:
    path: /db/docker-compose.yml
  register: compose_file

- name: Run Docker Compose down
  shell: sudo docker compose down
  args:
    chdir: /db
  when: compose_file.stat.exists
```

3.2.7. Blocks

Trong Ansible, blocks (khối) cho phép nhóm các tác vụ lại với nhau và áp dụng một điều kiện cho toàn bộ nhóm tác vụ đó. Điều này giúp giảm sự lặp lại và làm cho playbook trở nên dễ đọc hơn. Blocks cũng cho phép áp dụng **rescue** và **always** block để xử lý các tình huống đặc biệt như lỗi hoặc việc dọn dẹp sau khi thực hiện các tác vụ.

Ví dụ về việc install và config Apache trên các host webservers. Khối rescue sẽ xử lý lỗi nếu trong block có bất kỳ lỗi nào xảy ra. Sau đó sẽ luôn restart lại server dù có lỗi hay không có lỗi để đảm bảo máy chủ hoạt động bình thường.

```
- name: Install and configure Apache
  hosts: webservers
  become: true
  tasks:
    - name: Ensure Apache package is installed
      block:
        - name: Install Apache on Ubuntu
          apt:
            name: apache2
            state: present
          when: ansible_distribution == 'Ubuntu'

      rescue:
        - name: Handle error
          debug:
            msg: "Error occurred while installing Apache"

      always:
        - name: Restart Apache
          service:
            name: apache2
            state: restarted
```

3.2.8. Handlers

Đôi khi muốn một tác vụ chỉ chạy khi có thay đổi được thực hiện trên máy. Ví dụ: có thể muốn khởi động lại một dịch vụ nếu một tác vụ cập nhật cấu hình của dịch vụ đó chứ không khởi động lại khi cấu hình không thay đổi. Handlers là những tác vụ chỉ chạy khi được notified

Ví dụ về việc khởi động lại server apache khi có thay đổi về file cấu hình trong /etc/httpd.conf, hay nói các khác sẽ không khởi động lại apache nếu file cấu hình không thay đổi.

```
---
- name: Verify apache installation
  hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
    - name: Ensure apache is at the latest version
      ansible.builtin.yum:
        name: httpd
        state: latest

    - name: Write the apache config file
      ansible.builtin.template:
        src: /srv/httpd.j2
        dest: /etc/httpd.conf
      notify:
        - Restart apache

    - name: Ensure apache is running
      ansible.builtin.service:
        name: httpd
        state: started

  handlers:
    - name: Restart apache
      ansible.builtin.service:
        name: httpd
        state: restarted
```

Một task có thể yêu cầu 1 hoặc nhiều handlers để thực thi sử dụng từ khóa notify. Chú ý, handlers được thực thi theo thứ tự được định nghĩa trong handlers section, chứ không phải theo thứ tự trong câu lệnh notify, tức là trong đoạn code dưới đây, Restart memcached sẽ được thực thi trước, Restart apache sẽ thực thi sau

```
notify:
    - Restart apache
    - Restart memcached
handlers:
  - name: Restart memcached
    ansible.builtin.service:
      name: memcached
      state: restarted

  - name: Restart apache
    ansible.builtin.service:
      name: apache
      state: restarted
```

Việc thông báo cho cùng một trình xử lý nhiều lần sẽ dẫn đến việc thực thi trình xử lý đó chỉ một lần bất kể có bao nhiêu tác vụ thông báo cho nó. Ví dụ: nếu

nhiều tác vụ cập nhật tệp cấu hình và thông báo cho trình xử lý khởi động lại Apache, Ansible chỉ trả lại Apache một lần để tránh những lần khởi động lại không cần thiết.

Xác định thời điểm task thay đổi, có thể kiểm soát thời điểm handlers được notified về sự thay đổi của 1 task bằng việc sử dụng **changed_when**.

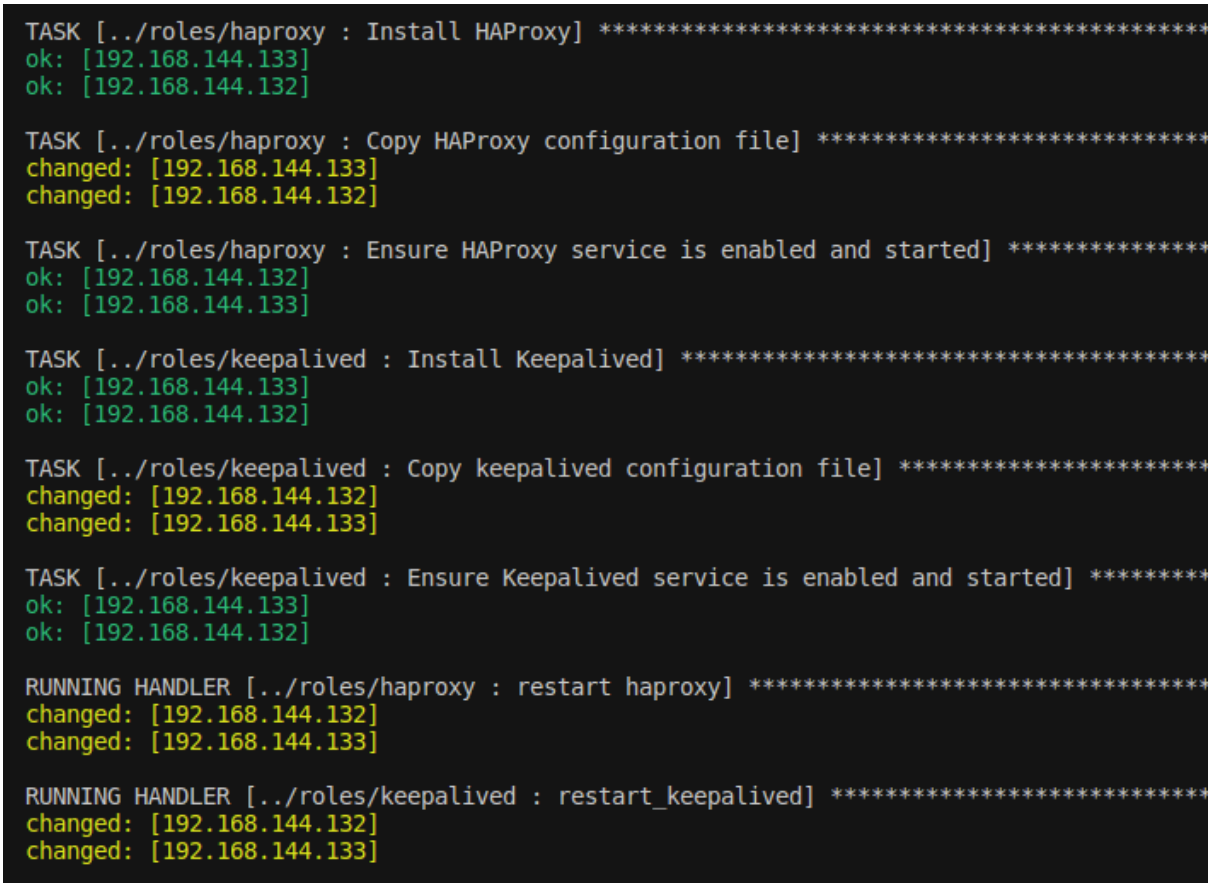
Thử nghiệm mỗi khi copy lại file cấu hình của dịch vụ HAProxy và keepalived, sẽ luôn thông báo cho handler restart lại dịch vụ HAProxy và keepalived

```
# playbook
- hosts: lb
  become: true
  roles:
    - ../roles/haproxy
    - ../roles/keepalived
```

```
# roles/haproxy
---
- name: Install HAProxy
  apt:
    name: haproxy
    state: present

- name: Copy HAProxy configuration
file
  copy:
    src: haproxy.cfg
    dest: /etc/haproxy/haproxy.cfg
    force: yes
  changed_when: true
  notify: restart haproxy

- name: Ensure HAProxy service is
enabled and started
  service:
    name: haproxy
    state: started
    enabled: yes

handlers:
  - name: restart haproxy
    service:
      name: haproxy
      state: restarted
```

```
# roles/keepalived
---
- name: Install Keepalived
  apt:
    name: keepalived
    state: present

- name: Copy keepalived
configuration file
  copy:
    src: "{{ config_file }}"
    dest:
/etc/keepalived/keepalived.conf
    force: yes
  changed_when: true
  notify: restart_keepalived

- name: Ensure Keepalived service
is enabled and started
  service:
    name: keepalived
    state: started
    enabled: yes

handlers:
  - name: restart_keepalived
    service:
      name: keepalived
      state: restarted
```

Kết quả thực thi playbook:



3.2.9. Error handling in playbooks

Error handling trong Ansible playbooks là một phần quan trọng để đảm bảo rằng bạn có thể xử lý các lỗi một cách hiệu quả và tiếp tục hoặc dừng việc thực thi playbook dựa trên các điều kiện lỗi.

Ignoring failed commands

Theo mặc định, Ansible dừng thực thi các tác vụ trên máy chủ khi một tác vụ không thành công trên máy chủ đó. Có thể sử dụng `ignore_errors` để tiếp tục dù có lỗi xảy ra

```
- name: Do not count this as a failure
  ansible.builtin.command: /bin/false
  ignore_errors: true
```

Ignoring unreachable host errors

Có thể bỏ qua lỗi tác vụ do host instance là 'UNREACHABLE' với từ khóa `ignore_unreachable`. Ansible bỏ qua các lỗi tác vụ nhưng vẫn tiếp tục thực thi các tác vụ trong tương lai đối với máy chủ không thể truy cập được.

```
- name: This executes, fails, and the failure is ignored
  ansible.builtin.command: /bin/true
  ignore_unreachable: true
```

```
- name: This executes, fails, and ends the play for this host
  ansible.builtin.command: /bin/true
```

Defining failure

Ansible cho phép xác định "failure" nghĩa là gì trong mỗi tác vụ bằng cách sử dụng điều kiện **failed_when**, các điều kiện có thể kết hợp với nhau qua từ khóa and/or Thông thường có thể kiểm tra failure bằng việc tìm kiếm từ khóa trong output của 1 command (ví dụ tìm từ 'FAILED' trong command_result) hoặc dựa vào return code

```
- name: Fail task when the command error output prints FAILED
  ansible.builtin.command: /usr/bin/example-command -x -y -z
  register: command_result
  failed_when: "'FAILED' in command_result.stderr"
```

```
- name: Fail task when both files are identical
  ansible.builtin.raw: diff foo/file1 bar/file2
  register: diff_cmd
  failed_when: diff_cmd.rc == 0 or diff_cmd.rc >= 2
```

Setting a maximum failure percentage

Theo mặc định, Ansible tiếp tục thực thi các tác vụ miễn là có máy chủ chưa bị lỗi. Trong một số trường hợp, chẳng hạn như khi thực hiện cập nhật luân phiên, bạn có thể muốn hủy quá trình phát khi đã đạt đến một ngưỡng lỗi nhất định. Cài đặt **max_fail_percentage** áp dụng cho từng batch khi sử dụng với **Ansible serial**. Trong ví dụ dưới, nếu hơn 3 trong số 10 máy chủ trong lô máy chủ bất kỳ nào bị lỗi thì phần còn lại của play sẽ bị hủy bỏ (aborted).

```
---
- hosts: webservers
  max_fail_percentage: 30
  serial: 10
```

3.2.10. Roles

Roles cho phép tự động tải các vars, files, tasks, handlers và các artifact Ansible khác có liên quan dựa trên cấu trúc tệp đã biết. Sau khi nhóm nội dung của mình thành các Role, có thể dễ dàng sử dụng lại chúng và chia sẻ chúng với những người dùng khác.

```
roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies
        library/          # roles can also include custom modules
        module_utils/     # roles can also include custom module_utils
        lookup_plugins/   # or other types of plugins, like lookup in this case

    webtier/              # same kind of structure as "common" was above
    monitoring/           # ""
```

Theo mặc định, Ansible sẽ tìm trong hầu hết các thư mục role để tìm tệp main.yml để tìm nội dung liên quan:
task/main.yml - Danh sách các nhiệm vụ mà vai trò cung cấp cho play để thực hiện.
handlers/main.yml - Các trình xử lý được nhập vào vở kịch gốc để sử dụng theo vai trò hoặc các vai trò và nhiệm vụ khác trong vở kịch.
defaults/main.yml - giá trị ưu tiên rất thấp cho các biến do vai trò cung cấp. Giá trị mặc định của riêng một vai trò sẽ được ưu tiên hơn giá trị mặc định của vai trò khác, nhưng bất kỳ/tất cả các nguồn biến khác sẽ ghi đè lên giá trị mặc định này.
vars/main.yml - Các biến có độ ưu tiên cao do vai trò cung cấp cho vở kịch
files/config.txt - một hoặc nhiều tệp có sẵn cho vai trò và đó là vai trò con.
templates/something.j2 - các mẫu để sử dụng trong vai trò hoặc vai trò con.
meta/main.yml - metadata cho role

3.2.11. Module defaults

Trong Ansible, module_defaults là một tính năng cho phép thiết lập các giá trị mặc định cho các tùy chọn của các module. Điều này giúp giảm thiểu việc lặp lại cùng một cấu hình trong nhiều tác vụ và làm cho playbook gọn gàng hơn. module_defaults có thể được sử dụng ở cấp độ playbook hoặc role, và nó sẽ áp dụng cho tất cả các tác vụ trong phạm vi của nó.

- **Thiết lập Module defaults ở Cấp Độ Playbook**

Có thể thiết lập các giá trị mặc định cho các module ở cấp độ playbook. Điều này có nghĩa là tất cả các tác vụ trong playbook sẽ sử dụng các giá trị mặc định này trừ khi có giá trị cụ thể được cung cấp cho các tác vụ đó.

```
---
- name: Example playbook with module_defaults
  hosts: all

  module_defaults:
    apt:
      update_cache: yes
      cache_valid_time: 3600
    service:
      enabled: yes

  tasks:
    - name: Install a package
      apt:
        name: nginx

    - name: Ensure Nginx is running
      service:
        name: nginx
        state: started
```

Trong ví dụ này trên, mọi tác vụ sử dụng module apt sẽ tự động có update_cache: yes và cache_valid_time: 3600, mọi tác vụ sử dụng module service sẽ tự động có enabled: yes.

- **Thiết lập Module defaults trong Roles**

  Cấu trúc của role:

  my_role/
  ├── defaults/
  │ └── main.yml
  ├── tasks/
  │ └── main.yml
  └── handlers/
  └── main.yml

```
# roles/my_role/defaults/main.yml
module_defaults:
  apt:
    update_cache: yes
    cache_valid_time: 3600
  service:
    enabled: yes
```

```
# roles/my_role/tasks/main.yml
---
- name: Install a package
  apt:
    name: nginx

- name: Ensure Nginx is running
  service:
    name: nginx
    state: started
```

Role my_role sẽ áp dụng các giá trị mặc định cho apt và service giống như trong ví dụ ở cấp độ playbook.

Nếu một giá trị được chỉ định trực tiếp trong task, giá trị đó sẽ ghi đè giá trị mặc định được thiết lập trong module_defaults. module_defaults chỉ áp dụng trong phạm vi mà nó được định nghĩa (playbook hoặc role).

Ưu điểm của việc sử dụng module_defaults

- Giảm thiểu việc phải lặp lại cùng một cấu hình nhiều lần trong các tác vụ khác nhau.
- Tạo ra các cấu hình tập trung, dễ dàng điều chỉnh khi cần thay đổi.
- Đảm bảo các giá trị mặc định được áp dụng nhất quán trong toàn bộ playbook hoặc role.

3.2.12. Variables

Ansible sử dụng các biến để quản lý sự khác biệt giữa các hệ thống. Với Ansible, có thể thực thi các task và playbook trên nhiều hệ thống khác nhau chỉ bằng một lệnh. Để thể hiện sự khác biệt giữa các hệ thống khác nhau đó, có thể tạo các biến bằng cú pháp YAML tiêu chuẩn, bao gồm list và dictionary. Có thể xác định các biến này trong playbook, inventory, files, roles và có thể sử dụng tại command line. Cũng có thể tạo các biến trong khi chạy playbook bằng cách đăng ký giá trị trả về hoặc các giá trị của task như 1 biến mới

Định nghĩa 1 biến:

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

Tham chiếu tới giá trị của 1 biến:

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

Định nghĩa và tham chiếu 1 biến dạng list

```
# Định nghĩa 1 biến dạng list
region:
  - northeast
  - southeast
  - midwest
```

```
# Tham chiếu tới 1 phần tử của list
region: "{{ region[0] }}"
```

Định nghĩa và tham chiếu 1 biến dạng dictionary

```
# Định nghĩa 1 biến dạng dictionary
foo:
  field1: one
  field2: two
```

```
# Tham chiếu tới item của dictionary
foo['field1']
foo.field1
```

Registering variables: Khởi tạo các variable từ output của các Ansible task với từ khóa register, và có thể sử dụng các registered variable này trong các task phía sau trong play. Ví dụ về việc kiểm tra sự tồn tại của file docker-compose.yml, trước khi thực hiện docker compose up/down, ếu không tồn tại file docker-compose sẽ skip task

kế tiếp.

```
- name: Check if docker-compose.yml exists
  stat:
    path: /vdt-db/docker-compose.yml
  register: compose_file

- name: Run Docker Compose down
  shell: sudo docker compose down
  args:
    chdir: /vdt-db
  when: compose_file.stat.exists
```

3.3. Thực thi Ansible playbooks

3.3.1. Validating tasks: check mode and diff mode

Trong Ansible, có hai chế độ chính mà có thể sử dụng để xác thực các tác vụ và đảm bảo playbook hoạt động như mong đợi: check mode và diff mode.

- **Sử dụng Check mode**

  Check mode chỉ là mô phỏng. Nó sẽ không tạo đầu ra cho các tác vụ sử dụng điều kiện dựa trên các registered variables (kết quả của các task trước đó). Cú pháp chạy check mode:

  ```
  ansible-playbook ansible.yml --check
  ```

  Thử nghiệm chạy ansible playbook triển khai dịch vụ vdt_db (database) lên 1 VM:

```
# roles/vdt_db
- name: Check if docker-compose.yml exists
  stat:
    path: /vdt-db/docker-compose.yml
  register: compose_file

- name: Run Docker Compose down
  shell: sudo docker compose down
  args:
    chdir: /vdt-db
  when: compose_file.stat.exists

- name: Remove directory
  file:
    path: /vdt-db
    state: absent

- name: Create directory vdt-db
  file:
    path: /vdt-db
    state: directory
    mode: '0755'

- name: Render Docker Compose template
  template:
    src: docker-compose.yml.j2
    dest: /vdt-db/docker-compose.yml


- name: Run Docker Compose up
  shell: sudo docker compose up -d
  args:
    chdir: /vdt-db
```

Kết quả thử nghiệm:

```
PLAY [vdt_db] ***********************************************************

TASK [Gathering Facts] **************************************************
ok: [192.168.144.135]

TASK [../roles/vdt_db : Check if docker-compose.yml exists] ***********
ok: [192.168.144.135]

TASK [../roles/vdt_db : Run Docker Compose down] ***********************
skipping: [192.168.144.135]

TASK [../roles/vdt_db : Remove directory] ******************************
changed: [192.168.144.135]

TASK [../roles/vdt_db : Create directory vdt-db] ***********************
ok: [192.168.144.135]

TASK [../roles/vdt_db : Render Docker Compose template] ****************
ok: [192.168.144.135]

TASK [../roles/vdt_db : Run Docker Compose up] *************************
skipping: [192.168.144.135]
```

Nhận xét: Sau khi chạy ansible-playbook, trên VM đã không chạy dịch vụ database, đồng thời một số task bị skipping do một số task phía trước không được thực hiện thật sự do đó registered variable đã trả về giá trị false khi kiểm tra file docker-compose có tồn tại

Ngoài ra, Ansible hỗ trợ việc cấu hình một số tác vụ nhất định luôn chạy hoặc không bao giờ chạy ở chế độ kiểm tra, bất kể chạy playbook có hay không có --check, có thể thêm tùy chọn check_mode vào các tác vụ đó:

```
tasks:
  - name: This task will always make changes to the system
    ansible.builtin.command: /something/to/run --even-in-check-mode
    check_mode: false

  - name: This task will never make changes to the system
    ansible.builtin.lineinfile:
      line: "important config"
      dest: /path/to/myconfig.conf
      state: present
    check_mode: true
    register: changes_to_important_config
```

- **Sử dụng Diff mode**

Diff mode trong Ansible cho thấy sự khác biệt giữa trạng thái hiện tại của tài nguyên và trạng thái mà Ansible sẽ thực thi. Điều này đặc biệt hữu ích để hiểu những thay đổi nào đang được đề xuất bởi một playbook hoặc role. Để sử dụng Diff mode, thêm tùy chọn --diff vào lệnh ansible-playbook.

```
ansible-playbook my_playbook.yml --diff
```

Điều này đặc biệt hữu ích cho các tệp cấu hình (configuration files), nơi nó có thể hiển thị chính xác các dòng sẽ được thêm, xóa hoặc sửa đổi.
Thử nghiệm tương tự chạy ansible playbook triển khai dịch vụ vdt_db (database) lên 1 VM có kết quả như sau:
Đối với task Remove directory

```
TASK [../roles/vdt_db : Check if docker-compose.yml exists] ****************
ok: [192.168.144.135]

TASK [../roles/vdt_db : Run Docker Compose down] ***************************
changed: [192.168.144.135]

TASK [../roles/vdt_db : Remove directory] ***********************************
--- before
+++ after
@@ -1,10 +1,4 @@
 {
     "path": "/vdt-db",
-    "path_content": {
-        "directories": [],
-        "files": [
-            "/vdt-db/docker-compose.yml"
-        ]
-    },
-    "state": "directory"
+    "state": "absent"
 }

changed: [192.168.144.135]
```

Đối với task Remove directory (xóa workspace cũ), có thể thấy trước đó thư mục có trạng thái là directory (có tồn tại), lúc sau có trạng thái là absent (đã bị xóa)

Đối với task Create directory vdt-db

```
TASK [../roles/vdt_db : Create directory vdt-db] ***************************
--- before
+++ after
@@ -1,4 +1,4 @@
 {
     "path": "/vdt-db",
-    "state": "absent"
+    "state": "directory"
 }

changed: [192.168.144.135]
```

Đối với task Create directory vdt-db, trước đó thư mục đã bị xóa đi nên có trạng thái là absent, sau khi thực thi task tạo mới thư mục đã có trạng thái là directory (tồn tại thư mục)

Đối với task Render Docker compose template (tạo file docker compose từ 1 template có sẵn và chỉnh sửa cấu hình dựa vào variable)

```
TASK [../roles/vdt_db : Render Docker Compose template] ****************************************
--- before
+++ after: /home/vinh/.ansible/tmp/ansible-local-21999kw8hwtou/tmphoemtvip/docker-compose.yml.j2
@@ -0,0 +1,13 @@
+version: '3'
+
+services:
+  vdt-db:
+    image: "vinhbh/vdt_db:1.0"
+    container_name: "vdt-db"
+    ports:
+      - "5432:5432"
+    volumes:
+      - vdt-data:/var/lib/postgresql/data
+
+volumes:
+  vdt-data:

changed: [192.168.144.135]
```

Ansible hỗ trợ việc kết hợp cả Check Mode và Diff Mode. Điều này hữu ích cho việc xem xét kỹ lưỡng trước khi áp dụng bất kỳ thay đổi nào.

```
ansible-playbook my_playbook.yml --check --diff
```

3.3.2. Tags

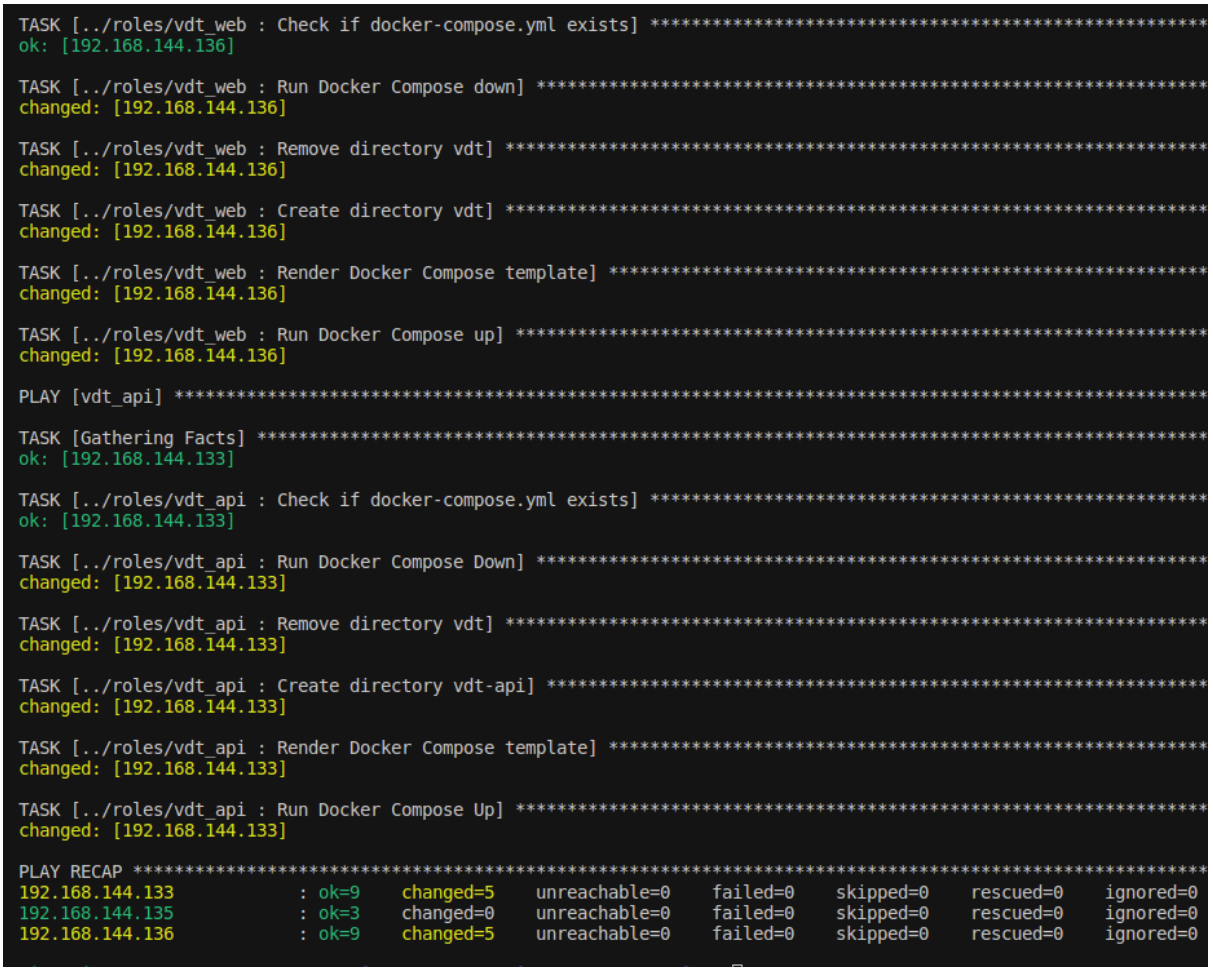
Nếu có một playbook lớn, có thể sẽ hữu ích nếu chỉ chạy các phần cụ thể của playbook thay vì chạy toàn bộ playbook. Có thể làm điều này với Ansible tag. Sử dụng tag để thực thi hoặc bỏ qua các task đã chọn là một quy trình gồm hai bước:

- Thêm thẻ vào các task, riêng lẻ hoặc kế thừa thẻ từ một blocks, play, role hoặc import.
- Chọn hoặc bỏ qua thẻ khi chạy playbook.

Mỗi task có thể gán 1 hoặc nhiều tag khác nhau, có thể thêm tag vào tasks trong playbook, hoặc task files hoặc trong 1 role. Và có thể gán cùng 1 tag cho nhiều task khác nhau.
Thử nghiệm gắn tag cho role

```
---

- hosts: all
  become: true
  tasks:
  - name: update repository index (Ubuntu)
    apt:
      update_cache: true
    when: ansible_distribution == "Ubuntu"

- hosts: all
  become: true
  roles:
    - ../roles/common

- hosts: vdt_db
  become: true
  roles:
    - ../roles/vdt_db

- hosts: vdt_web
  become: true
  roles:
    - ../roles/vdt_web
  tags:
    - vdt_student_website

- hosts: vdt_api
  become: true
  roles:
    - ../roles/vdt_api
  tags:
    - vdt_student_website
    - vdt_student_backend
```

Kết quả chạy tag vdt_student_website:
ansible-playbook playbooks/ansible.yml -K --tags vdt_student_website



Nhận xét: Chỉ những role có tag vdt_student_website được thực thi

Tag đặc biệt: always và never

Ansible dành hai tên tag cho hành vi đặc biệt: always và never.

- Nếu gán tag always cho một task hoặc play, Ansible sẽ luôn chạy task hoặc play đó, trừ khi bỏ qua nó một cách cụ thể (--skip-tags always).

```
- hosts: all
  become: true
  roles:
    - ../roles/common
  tags: ['always']
```

Trong ví dụ trên Task common sẽ luôn được thực thi

- Nếu chỉ định tag never cho một task hoặc play, Ansible sẽ bỏ qua task hoặc play đó trừ khi yêu cầu cụ thể (--tags never).

```
tasks:
  - name: Run the rarely-used debug task
    ansible.builtin.debug:
     msg: '{{ showmevar }}'
    tags: [ never, debug ]
```

Trong ví dụ trên Task debug hiếm khi được sử dụng, chỉ chạy khi yêu cầu cụ thể thẻ `debug` hoặc `never`.

3.3.3. Debugging tasks

Trình Debugging không được bật theo mặc định. Nếu muốn gọi trình gỡ lỗi trong quá trình thực thi playbook, cần kích hoạt bằng một số cách như:

- Sử dụng keyword debug
- Cấu hình trong biến environment hoặc file configuration
- Sử dụng strategy

Kích hoạt bằng sử dụng keyword debug

Có thể kích hoạt trình gỡ lỗi trên các tác vụ mới hoặc cập nhật. Nếu chúng thất bại, có thể sửa lỗi một cách hiệu quả. Từ khóa trình gỡ lỗi chấp nhận năm giá trị: always, never, on_failed, on_unreachable, on_skipped

Kích hoạt bằng sử dụng keyword debug

Có thể kích hoạt trình debug tác vụ bằng cài đặt trong ansible.cfg hoặc bằng một biến môi trường. Nếu đặt tùy chọn cấu hình hoặc biến môi trường thành True, Ansible sẽ chạy trình gỡ lỗi trên các tác vụ không thành công theo mặc định.

```
# File ansible.cfg
[defaults]
enable_task_debugger = True
```

Kích hoạt bằng sử dụng keyword debug

Có thể thực hiện việc này ở cấp độ play, trong ansible.cfg hoặc với biến môi trường ANSIBLE_STRATEGY=debug. Có thể sử dụng một trong hai cách

```
# Trong play
- hosts: test
  strategy: debug
  tasks:
```

```
# File ansible.cfg
[defaults]
strategy = debug
```

Thử nghiệm kích hoạt bằng sử dụng keyword khi cấu hình api server

- Trong playbook đặt biến debugger: on_failed

```
- hosts: vdt_api
  become: true
  roles:
    - ../roles/vdt_api
  debugger: on_failed
```

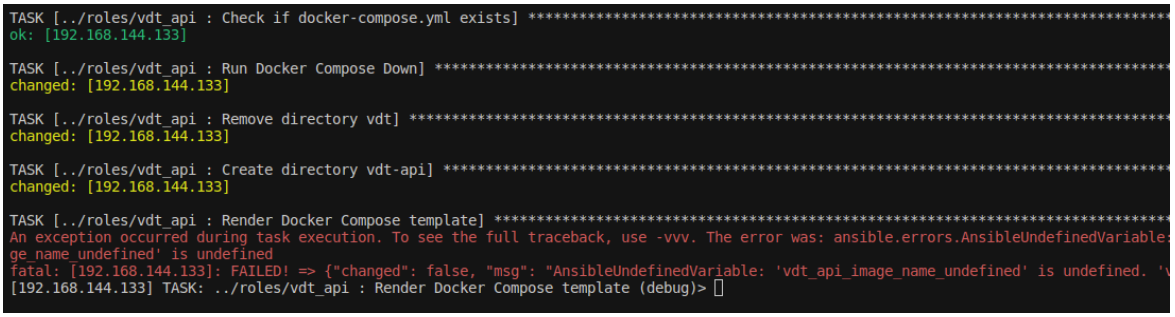
- File variables:

```
vdt_api_image_name: "vinhbh/vdt_api"
vdt_api_image_version: "2.0"
vdt_api_container_name: "vdt-api"
vdt_api_port: 8000
vdt_api_volume: "vdt-api-data"
```

- Tạo 1 lỗi nhỏ trong file template do tham chiếu tên biến sai
  File template docker-compose.yml.j2

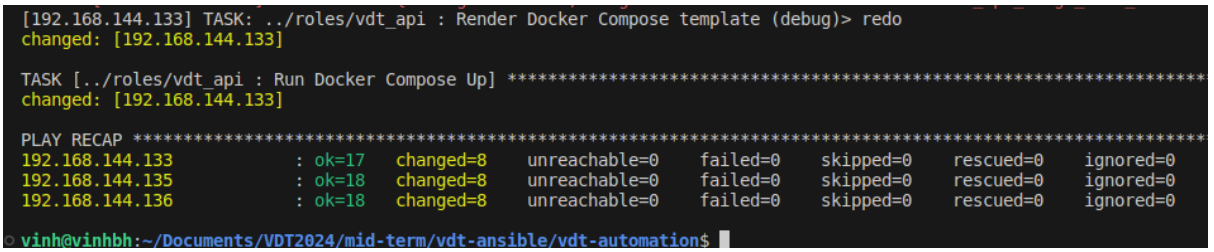
```
version: '3.8'

services:
  vdt-api:
    image: "{{ vdt_api_image_name_undefined }}:{{
vdt_api_image_version }}"
    container_name: "{{ vdt_api_container_name }}"
    command: python manage.py runserver 0.0.0.0:8000
    ports:
      - {{ vdt_api_port }}:8000
    volumes:
      - {{ vdt_api_volume }}:/app

volumes:
  {{ vdt_api_volume }}:
```

Kết quả thực hiện câu lệnh chạy playbook



Thông báo lỗi biến *vdt_api_image_name_undefined* là undefined, và xuất hiện yêu cầu debug

- Tiến hành sửa lại biến từ *vdt_api_image_name_undefined* thành *vdt_api_image_name*, và chạy lại debug với command: redo



Nhận xét: Playbook đã chạy tiếp task bị lỗi sau khi sửa lỗi mà không chạy lại playbook từ đầu.

3.3.4. Kiểm soát luồng thực thi với một số keyword

Setting the batch size with serial

Theo mặc định, Ansible chạy song song với tất cả các host theo mẫu đặt trong trường host của play. Nếu chỉ muốn quản lý một số lượng host nhất định tại một thời điểm, chẳng hạn như trong quá trình cập nhật liên tục, có thể xác định số lượng máy chủ Ansible sẽ quản lý cùng một lúc bằng cách sử dụng từ khóa **serial**

Có thể thiết lập serial ở dạng số lượng, dạng phần trăm, hoặc dạng list.

Giả sử có 6 host nhưng muốn thực hiện lần lượt 3 host, thiết lập serial: 3

```
---
- name: test play
  hosts: webservers
  serial: 3
  gather_facts: False

  tasks:
    - name: first task
      command: hostname
    - name: second task
      command: hostname
```

Trong ví dụ trên, nếu có 6 máy chủ trong nhóm `webservers`, Ansible sẽ thực hiện phát hoàn toàn (cả hai nhiệm vụ) trên 3 máy chủ trước khi chuyển sang 3 máy chủ tiếp theo

Thiết lập serial dạng phần trăm:

```
---
- name: test play
  hosts: webservers
  serial: "30%"
```

Nếu số lượng máy chủ không chia đều cho số lượt đi thì lượt cuối cùng chứa phần còn lại. Trong ví dụ này, nếu có 20 máy chủ trong nhóm máy chủ web thì batch đầu tiên sẽ chứa 6 máy chủ, batch thứ hai sẽ chứa 6 máy chủ, batch thứ ba sẽ chứa 6 máy chủ và batch cuối cùng sẽ chứa 2 máy chủ.

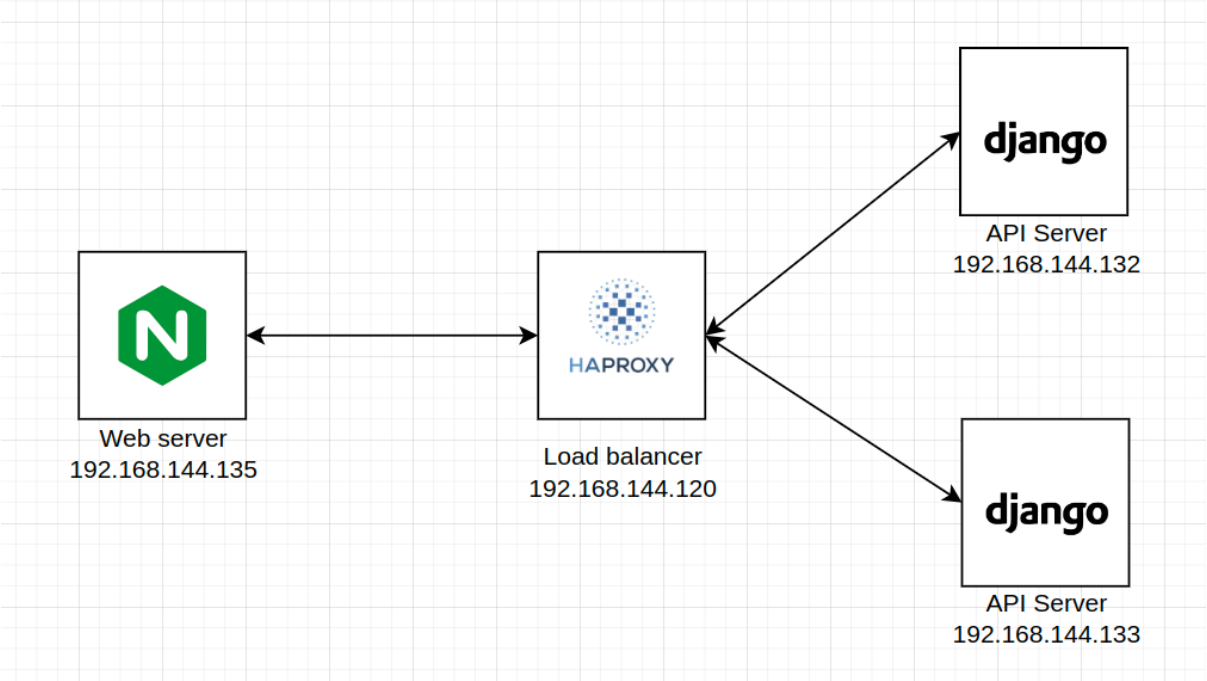
Thiết lập serial dạng list:

```
---
- name: test play
  hosts: webservers
  serial:
    - 1
    - 5
    - 10
```

Trong ví dụ trên, batch đầu tiên sẽ chứa một máy chủ duy nhất, batch tiếp theo sẽ chứa 5 máy chủ và (nếu còn sót lại bất kỳ máy chủ nào), mỗi batch tiếp theo sẽ chứa 10 máy chủ hoặc tất cả các máy chủ còn lại, nếu ít hơn 10 máy chủ vẫn.

3.4. Thử nghiệm Ansible serial triển khai lần lượt các server

Mô hình gồm 3 roles: Web server, Load Balancer và API Server



Kịch bản thử nghiệm: Web server sẽ call api tới load balancer, tại đây HA Proxy sẽ điều phối tải tới 2 instance của API Server chạy trên 2 VM khác nhau.

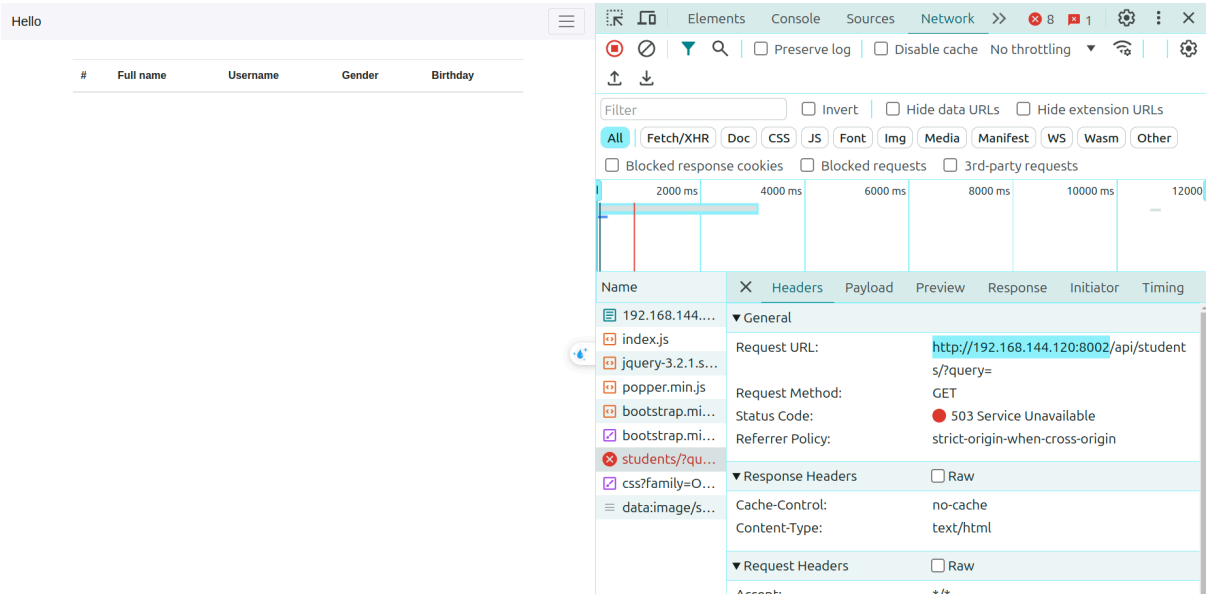
Giả sử có yêu cầu triển khai code mới lên 2 API Server (sử dụng docker), tiến hành lần lượt docker compose down từng API Server, sau đó docker compose up lại, không thực hiện đồng thời cùng 1 lúc, vì nếu làm như vậy sẽ dẫn tới có một khoảng down-time tại API Server. Việc down lần lượt từng API Server sẽ hạn chế down-time. Hay nói cách khác, trong quá trình triển khai, sẽ luôn có ít nhất 1 API Server đang hoạt động.

Sử dụng keyword serial trong việc chia lần lượt việc thực thi ansible task tới 2 VM, sử dụng serial: 50% (tương đương với 01 trên 02 VM)

Nếu không down lần lượt từng VM, sẽ có khoảng thời gian khi cả 2 API Server đều đang thực hiện docker compose down

```
TASK [../roles/web_clients : Run Docker Compose Down] ***************************************************************
changed: [192.168.144.133]
changed: [192.168.144.132]

TASK [../roles/web_clients : Run Docker Compose Up] *****************************************************************
changed: [192.168.144.133]
changed: [192.168.144.132]
```

Kết quả trên trình duyệt



Sau khi sử dụng serial: 50% trong việc down lần lượt từng API Server, kết quả thực hiện ansible-playbook:

```
TASK [../roles/web_servers : Remove directory] **********************
changed: [192.168.144.135]

TASK [../roles/web_servers : Create directory api-project] **********
changed: [192.168.144.135]

TASK [../roles/web_servers : Copy django-nginx directory] ***********
changed: [192.168.144.135]

TASK [../roles/web_servers : Run Docker Compose Down] ***************
changed: [192.168.144.135]

TASK [../roles/web_servers : Run Docker Compose Up] *****************
changed: [192.168.144.135]

PLAY [web_servers] **************************************************

TASK [Gathering Facts] **********************************************
ok: [192.168.144.136]

TASK [../roles/web_servers : Remove directory] **********************
changed: [192.168.144.136]

TASK [../roles/web_servers : Create directory api-project] **********
changed: [192.168.144.136]

TASK [../roles/web_servers : Copy django-nginx directory] ***********
changed: [192.168.144.136]

TASK [../roles/web_servers : Run Docker Compose Down] ***************
changed: [192.168.144.136]

TASK [../roles/web_servers : Run Docker Compose Up] *****************
changed: [192.168.144.136]
```

Nhận xét: Trong một thời điểm sẽ luôn còn lại 50% số lượng API Server đang hoạt động, tránh tình trạng down time khi có request tới API Server.

Restricting execution with throttle

Từ khóa throttle giới hạn số lượng workers cho 1 task cụ thể, nó có thể được thiết lập ở level block hoặc task. Sử dụng nhằm hạn chế các tác vụ có thể sử dụng nhiều CPU hoặc tương tác với API giới hạn tốc độ. Để đạt được hiệu quả, throttle phải được thiết lập nhỏ hơn serial nếu kết hợp cùng nhau.

```
- name: Deploy application
  hosts: all
  tasks:
  - name: Start application service
        service:
          name: myapp
          state: started
        throttle: 5
```

Ví dụ trên sử dụng *throttle: 5* nhằm đặt giới hạn tối đa 5 host khởi động dịch vụ tại cùng một thời điểm
Nhận xét: Việc điều chỉnh có thể giúp quản lý tải nhưng có thể tăng thời gian chạy tổng thể của playbook khi các tác vụ được thực thi theo đợt nhỏ hơn.

Chương 4. Bảo vệ các dữ liệu nhạy cảm với Ansible vault

4.1. Tổng quan về Ansible Vault

Ansible Vault mã hóa các biến và file để có thể bảo vệ nội dung nhạy cảm như mật khẩu hoặc khóa thay vì để nội dung đó hiển thị dưới dạng văn bản gốc trong playbook hoặc role. Để sử dụng Ansible Vault, cần một hoặc nhiều mật khẩu để mã hóa và giải mã nội dung. Nếu lưu trữ mật khẩu vault của mình trong công cụ của bên thứ ba, chẳng hạn như trình quản lý bí mật, cần có tập lệnh để truy cập chúng. Sử dụng mật khẩu bằng công cụ dòng lệnh ansible-vault để tạo và xem các biến được mã hóa, tạo file được mã hóa, mã hóa file hiện có hoặc chỉnh sửa, khóa lại hoặc giải mã file. Sau đó, có thể đặt nội dung được mã hóa dưới sự kiểm soát và chia sẻ nội dung đó một cách an toàn hơn.

4.2. Thực hiện mã hóa với Ansible Vault

Một số thao tác cơ bản khi làm việc với Ansible Vault

- Tạo 1 file mới và mã hóa nó bằng lệnh **ansible-vault create.** Lệnh này sẽ mở trình soạn thảo mặc định của để nhập nội dung vào file secret.yml. Sau khi lưu và đóng trình soạn thảo, file sẽ được mã hóa.

```
ansible-vault create secret.yml
```

- Mã hóa 1 file đã tồn tại với **ansible-vault encrypt**

```
ansible-vault encrypt existing_file.yml
```

- Giải mã 1 file với **ansible-vault decrypt**

```
ansible-vault decrypt secret.yml
```

- Chỉnh sửa 1 file đã mã hóa với **ansible-vault edit**

```
ansible-vault edit secret.yml
```

- Chạy playbook với Ansible Vault: Có 2 cách phổ biến để chạy playbook, sử dụng tùy chọn --ask-vault-pass hoặc sử dụng file mật khẩu để tránh nhập mật khẩu vault nhiều lần

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

```
ansible-playbook playbook.yml --vault-password-file .vault_pass.txt
```

4.3. Thử nghiệm mã hóa dữ liệu thông tin database khi triển khai dịch vụ API Server

Kịch bản thử nghiệm: API Server sử dụng docker trong quá trình triển khai, API Server có yêu cầu kết nối tới server database thông qua các biến được lưu trữ trong thư mục vars của roles.

Cấu trúc thư mục

```
├── tasks
│   └── main.yml
├── templates
│   └── docker-compose.yml.j2
└── vars
    ├── main.yml
    └── vault.yml
```

File tasks/main.yml

```
- name: Check if docker-compose.yml exists
  stat:
    path: /vdt-api/docker-compose.yml
  register: compose_file

- name: Run Docker Compose Down
  shell: sudo docker compose down
  args:
    chdir: /vdt-api
  when: compose_file.stat.exists

- name: Remove directory vdt
  file:
    path: /vdt-api
    state: absent

- name: Create directory vdt-api
  file:
    path: /vdt-api
    state: directory
    mode: '0755'

- name: Render Docker Compose template
  template:
    src: docker-compose.yml.j2
    dest: /vdt-api/docker-compose.yml

- name: Run Docker Compose Up
  shell: sudo docker compose up -d
  args:
    chdir: /vdt-api
```

Thông tin play

```
- hosts: vdt_api
  become: true
  roles:
    - ../roles/vdt_api
  vars_files:
    - ../roles/vdt_api/vars/vault.yml
    - ../roles/vdt_api/vars/main.yml
  tags:
    - vdt_backend
```

Thông tin template sử dụng các biến nhạy cảm
File docker-compose.yml.j2

```
version: '3.8'

services:
  vdt-api:
    image: "{{ vdt_api_image_name }}:{{ vdt_api_image_version }}"
    container_name: "{{ vdt_api_container_name }}"
    command: python manage.py runserver 0.0.0.0:8000
    environment:
      - DATABASE_NAME={{ DATABASE_NAME }}
      - DATABASE_USER={{ DATABASE_USER }}
      - DATABASE_PASSWORD={{ DATABASE_PASSWORD }}
      - DATABASE_HOST={{ DATABASE_HOST }}
      - DATABASE_PORT={{ DATABASE_PORT }}
    ports:
      - {{ vdt_api_port }}:8000
    volumes:
      - {{ vdt_api_volume }}:/app

volumes:
  {{ vdt_api_volume }}:
```

Thông tin nhạy cảm trước khi được mã hóa Ansible Vault:
File roles/vdt_api/vars/vault.yml

vdt-automation > roles > vdt_api > vars > ! vault.yml

```
1  DATABASE_NAME: "vdt_db"
2  DATABASE_USER: "vinhbh"
3  DATABASE_PASSWORD: "123456789"
4  DATABASE_HOST: "192.168.144.135"
5  DATABASE_PORT: 5432
```

Tiến hành mã hóa file vault.yml

```
ansible-vault encrypt roles/vdt_api/vars/vault.yml
```

Kết quả sau khi mã hóa file vault.yml với Ansible Vault:
File roles/vdt_api/vars/vault.yml

vdt-automation > roles > vdt_api > vars > ! vault.yml

```
$ANSIBLE_VAULT;1.1;AES256
3938333338353466323338336661636365373638353061306164316236303839393532333865346 2
6331663035313637386461326530636638356135383839310a626138313730353666336635363534
3237316137343962373335653831333332623133663937643639663566316165353464326465623 5
3831653332343130340a376631653861396630739623633656338366234646539613836346165 34
6433343233326262323336383332316265356438343361653663393431373038313761326634646 2
3561306433306635623362653662383061396534323463326337626332646265613764363723823 8
3136626661656235356265376530353666313566366363393164363164383465666265313463633 9
3534633564343633386565396538326339343061376663313133636534343833633564373134643 5
6538346131326638623536343064363261636135323061613631323135333931333839323532663 8
3835653832313839343835373938613333663335323537393562326465373033356465356261313 8
353064643335326132336562623033663863
```

Tiến hành chạy playbook kèm tùy chọn --ask-vault-pass và điền mật khẩu Vault

```
ansible-playbook playbooks/ansible.yml --ask-vault-pass -K
```

```
vinh@vinhbh:~/Documents/VDT2024/mid-term/vdt-ansible/vdt-automation$ ansible-playbook playbooks/ansible.yml --ask-vault-pass -K
BECOME password:
Vault password:

PLAY [all] ********************************************************************************************************************************************

TASK [Gathering Facts] ********************************************************************************************************************************
ok: [192.168.144.135]
ok: [192.168.144.136]
ok: [192.168.144.133]

TASK [update repository index (Ubuntu)] ***************************************************************************************************************
changed: [192.168.144.133]
```

Thử nghiệm giải mã file vault.yml

```
ansible-vault decrypt roles/vdt_api/vars/vault.yml
```

Kết quả sau khi giải mã:

vdt-automation > roles > vdt_api > vars > ! vault.yml

```
DATABASE_NAME: "vdt_db"
DATABASE_USER: "vinhbh"
DATABASE_PASSWORD: "123456789"
DATABASE_HOST: "192.168.144.135"
DATABASE_PORT: 5432
```

Nhận xét: Giải mã giống với nội dung trước khi được mã hóa Vault

Ansible Vault cung cấp một phương pháp an toàn và linh hoạt để quản lý dữ liệu nhạy cảm trong các dự án Ansible. Bằng cách mã hóa thông tin nhạy cảm và sử dụng mật khẩu hoặc file mật khẩu để giải mã, có thể bảo vệ dữ liệu của mình một cách hiệu quả và an toàn.