Ansible

Notes:

* ansible-playbook first-playbook.yaml -i inventory --extra-vars "node1\_ip=10.221.242.53 username=root root\_password=Gyp.s8m" -v
* ansible -i inventory -m ping all --extra-vars "node1\_ip=10.221.242.53 username=root root\_password=Gyp.s8m" -v
* ansible -i inventory -m shell -a "ls /etc" all --extra-vars "node1\_ip=10.221.242.53 username=root root\_password=Gyp.s8m" -v
* role can be imported from ansible galaxy. It acts like docker hub.

ansible-galaxy role init test

* install roles from ansible galaxy

ansible-galaxy role install <role\_name>

#by running above command role will be available in ~/.ansible/role folder.

#import custom roles to ansible galaxy

#create a role and push the whole dircetory to github repo.

#import the role to galaxy using below command

#(api-token can be generated from ansible galaxy website - collections->api-token->load-token)

ansible-galaxy import <github\_username> <repo\_name> <api\_token>

* install new collections (collection\_name can be found on official documentation)

ansible-galaxy collection install <collection\_name>

* error handling in ansible

ansible playbook will fail even if a single task fails. If we want to avoid this then we can use

ignore\_error: true in our playbook.

Also, we can handle whether task should be performed or not by adding conditionals "failed\_when"

let's say we are doing cd or ls to a directory which does not exist then we can use failed\_when like -

- 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: Check if a file exists in temp and fail task if it does

ansible.builtin.command: ls /tmp/this\_should\_not\_be\_here

register: result

failed\_when:

- result.rc == 0

- '"No such" not in result.stdout'

* ansible-vault

It is like a bank locker. We can store sensitive info inside the vault and then access directly inside the playbook.

Steps to create and use vault ->

let's say we have to hide aws credentials.

1) create a random password for vault. -> openssl rand -base64 2048 > vault.pass

2) create a vault and pass the above password -> ansible-vault create group\_vars/all/aws-creds.yaml --vault-password-file vault.pass

a vault (vi editor window) will open where we need to put your credentials in "key: value" pairs. Now save the file and use the same keys in playbook to access the values. If we run -> cat aws-creds.yaml then we can see encryoted data.

If we want to see decrypted data then run ->

ansible-vault decrypt aws-creds.yaml --vault-password-file vault.pass = note that we need password to decrypt.

**Interview questions:**

1. What is configuration management?
2. Why Ansible? Over chef and puppet?

* Language used is yaml which is easy and readable.
* Agentless (uses SSH) Architecture
* Simple Installation as no agents needed
* Ideal for multi-cloud environments and **ad-hoc task execution**.

1. Push architecture vs pull architecture:

In a **push architecture**, the server actively sends (or "pushes") updates, data, or configurations to the clients. Clients do not need to request updates; instead, they receive them automatically from the server.

In a **pull architecture**, clients request (or "pull") updates, data, or configurations from a server. The clients actively check in with the server at specified intervals or upon certain conditions to see if any new information is available.

**Ansible is based on push architecture.**

1. Static vs dynamic inventory in ansible:

A screenshot of a black screen

Description automatically generated

1. How ansible works internally?
2. Explain ansible-vault.
3. Ansible workflow:

* You execute a playbook using the command = ansible-playbook -i inventory hosts playbook.yml  
  Ansible reads the inventory file(s) specified either static host file or dynamic host file.
* Ansible reads the playbook YAML file, parsing the plays and tasks defined within.
* Ansible establishes an SSH connection to each target host (or uses WinRM for Windows hosts).
* By default, Ansible collects system facts about the target hosts, which includes information like OS type, IP addresses, and hardware details. This can be disabled if not needed.
* Ansible iterates over the defined hosts in the playbook.
* Tasks within each play are executed in the order they are defined.
* Each task is executed with the goal of achieving the desired state without causing unintended changes if the state is already correct.
* Ansible collects results from each task execution, including whether it was successful, changed, or failed.
* Results are displayed in the console, showing the status of each task on each host.

1. Ansible tower:

**1. Introduction**

* Ansible Tower is a web-based interface and management tool for Ansible, designed to simplify the management and automation of IT tasks.

**2. Key Features**

* **User Interface**:
  + Graphical dashboard for managing playbooks, inventories, and jobs.
* **Role-Based Access Control (RBAC)**:
  + Define user roles and permissions for secure collaboration.
* **Job Scheduling**:
  + Schedule playbook runs at specific times or intervals.
* **Inventory Management**:
  + Tools for managing inventories and syncing with dynamic sources.
* **Job Templates**:
  + Create templates for running playbooks with specific configurations.
* **Integrated Logging**:
  + Capture detailed logs for troubleshooting and auditing.
* **Notifications**:
  + Send notifications upon job completion or failure.
* **API Access**:
  + RESTful API for programmatic access and integration.
* **Integration with Other Tools**:
  + Seamless integration with CI/CD tools and other DevOps systems.

**3. Benefits**

* **Ease of Use**:
  + Intuitive interface reduces the learning curve.
* **Collaboration**:
  + Managed roles and permissions enhance team collaboration.
* **Visibility and Control**:
  + Improved tracking and management of automation processes.
* **Scalability**:
  + Suitable for enterprise environments managing numerous hosts.

**4. Use Cases**

* **Automated Deployments**:
  + Consistent application deployments across environments.
* **Configuration Management**:
  + Manage configuration changes efficiently.
* **Infrastructure Provisioning**:
  + Provision infrastructure as code with cloud providers.
* **Continuous Integration/Continuous Deployment (CI/CD)**:
  + Automate parts of the software development lifecycle.

1. Write simple ansible playbook to install nginx:

---

- name: Playbook to install nginx

hosts: webservers

become: yes

tasks:

- name: Install nginx on webservers

apt:

name: nginx

state: present

- name: Start nginx service

service:

name: nginx

state: started