**1. What are Ansible roles & how to use it ?**

**Ans:-**

Ansible roles are a way to organize and manage playbooks by grouping related tasks, variables, files, templates, and handlers into reusable components. Roles help in structuring the playbooks, making them easier to maintain and reuse.

**2. What are Ansible playbook ?**

**Ans:-**

An Ansible playbook is a YAML file used to define a series of tasks to be executed on remote nodes. It is the core component of Ansible for automating configuration, deployment, and orchestration tasks.

**3. How do you manage Variable in Ansible ?**

**Ans:-**

In Ansible, variables are used to store values that can be reused throughout playbooks and roles. They help in customizing tasks, making playbooks more flexible and dynamic.

**4. How do you handle errors & debugging in Ansible ?**

**Ans:-**

Errors can be handles by using

ignore\_errors

failed\_when

rescue

always

Debugging can be handle by using

-v, -vv, -vvv

debug module

register & debug

--check

--syntax-check

**5. can you explain ansible strategy plugin**

**Ans:-**

Serial Strategy:

* Executes tasks in batches across hosts, defined by the serial keyword in the playbook.
* Allows controlling the number of hosts that can be active at a time.

Free Strategy:

* Executes tasks on all hosts simultaneously, without regard to the order defined in the playbook.
* Useful for scenarios where tasks are independent and order does not matter.

Linear Strategy (Default):

* Executes tasks one host at a time, in the order defined in the playbook.
* Useful for tasks that require strict ordering or synchronization.

6. What is Ansible and how does it work?

* + Answer: Ansible is an open-source automation tool used for configuration management, application deployment, and task automation. It works by connecting to nodes over SSH and pushing small programs, called "Ansible modules," to those nodes. These modules are written in Python, and they execute tasks defined in playbooks. Ansible does not require a client agent on the managed nodes, making it agentless.

7. Explain the structure of an Ansible playbook.

* + Answer: An Ansible playbook is written in YAML and is structured as a list of plays. Each play specifies a group of hosts to configure, the tasks to be executed on those hosts, and the roles to be applied. A playbook typically includes:
    - Hosts: Defines the target hosts.
    - Variables: Sets variables to be used.
    - Tasks: Lists the tasks to be performed.
    - Handlers: Defines tasks that run when notified.
    - Roles: Encapsulates multiple related tasks.

8. What are roles in Ansible and why are they useful?

* + Answer: Roles in Ansible are a way to organize playbooks by grouping related tasks, variables, files, templates, and handlers into separate directories. They promote reuse and simplify complex playbooks by providing a modular structure. Roles can be shared and reused across different projects, making them a powerful feature for large-scale automation.

9. How does Ansible manage variables and what is the precedence order of variables?

* + Answer: Ansible manages variables in various ways, including through inventory files, playbooks, roles, and special variable files. The precedence order from lowest to highest is:
    - Role defaults
    - Inventory file or script group vars
    - Inventory group\_vars/all
    - Playbook group\_vars/all
    - Inventory group\_vars/\*
    - Playbook group\_vars/\*
    - Inventory file or script host vars
    - Inventory host\_vars/\*
    - Playbook host\_vars/\*
    - Host facts
    - Play vars
    - Play vars\_prompt
    - Play vars\_files
    - Role vars
    - Block vars (only for tasks in block)
    - Task vars (only for the task)
    - Extra vars (always win precedence)

10. What is an inventory file in Ansible?

* + Answer: An inventory file is used by Ansible to define the hosts and groups of hosts on which the tasks in playbooks should be executed. Inventory files can be static or dynamic. A static inventory file is a simple text file, while a dynamic inventory file is generated by scripts that can query external data sources.

11. Scenario: You need to deploy an application on multiple servers and ensure that the service is restarted if any configuration changes. How would you achieve this using Ansible?

* + Answer:

yaml

Copy code

- hosts: webservers

tasks:

- name: Copy the configuration file

copy:

src: /path/to/config

dest: /etc/myapp/config

notify: restart myapp

handlers:

- name: restart myapp

service:

name: myapp

state: restarted

In this scenario, the copy module is used to transfer the configuration file. The notify directive triggers the restart myapp handler only if the file is changed.

12. Scenario: You need to create a new user on a set of servers and ensure that the user has sudo privileges. How would you do this?

* + Answer:

yaml

Copy code

- hosts: all

tasks:

- name: Create a new user

user:

name: newuser

state: present

groups: sudo

append: yes

This playbook uses the user module to create a new user and add them to the sudo group.

13. Scenario: How would you handle different configurations for development, staging, and production environments using Ansible?

* + Answer:
    - Use different inventory files or groups for each environment.
    - Create environment-specific variable files.
    - Structure your playbooks to include these variable files based on the environment.

yaml

Copy code

- hosts: all

vars\_files:

- "vars/{{ ansible\_env }}.yml"

tasks:

- name: Apply configuration

template:

src: "{{ ansible\_env }}/myapp.conf.j2"

dest: /etc/myapp/config

Here, ansible\_env is a variable that specifies the environment (e.g., development, staging, production), and corresponding variable files and templates are used.

14. Scenario: A playbook fails when it reaches a particular task. How do you debug and fix the issue?

* + Answer:
    - Use -vvv (verbose) mode to get more details about the failure.
    - Check the output logs for specific error messages.
    - Verify the syntax and parameters of the failing task.
    - Use ansible-playbook --start-at-task="Task Name" to rerun the playbook starting from the failed task after fixing the issue.

15. Scenario: How do you ensure idempotency in your Ansible playbooks?

* + Answer: Ensure that tasks are written in a way that their repeated execution does not change the system state beyond the initial change. Use modules that support idempotency by default (e.g., file, package). Avoid using shell commands unless necessary, and use condition checks to verify the state before making changes.

16. Explain the concept of Ansible Galaxy and its uses.

* + Answer: Ansible Galaxy is a repository for Ansible roles. It allows users to share, discover, and reuse roles written by the community. Ansible Galaxy can be used to download and install roles from the command line using ansible-galaxy commands, which helps streamline playbook development by leveraging pre-built roles for common tasks.

17. How do you manage secret or sensitive data in Ansible?

* + Answer: Ansible manages sensitive data using Ansible Vault. It allows you to encrypt entire files or specific variables to protect sensitive information such as passwords, API keys, and certificates. You can use commands like ansible-vault encrypt, ansible-vault decrypt, and ansible-vault edit to manage encrypted files. During playbook execution, you provide a password or a password file to decrypt the sensitive data.

18. What are Ansible facts and how can you use them in your playbooks?

* + Answer: Ansible facts are system properties that Ansible automatically gathers from managed nodes using the setup module. Facts include details about the operating system, network interfaces, memory, and more. You can use these facts in playbooks by referencing them as variables, such as ansible\_os\_family, ansible\_hostname, etc., to create conditional tasks or templates based on the system's state.

19. How do you handle error handling in Ansible playbooks?

* + Answer: Ansible provides several mechanisms for error handling, including:
  + ignore\_errors: Ignores errors on a particular task.
  + failed\_when: Customizes failure conditions for a task.
  + rescue and always: Part of block directives, where rescue specifies tasks to run if a block fails, and always specifies tasks to run regardless of success or failure.

yaml

Copy code

- block:

- name: Attempt to run a command

command: /bin/false

rescue:

- name: Handle the error

debug:

msg: "The command failed"

always:

- name: Always run this task

debug:

msg: "This task runs regardless of the previous results"

20. What is the purpose of the become directive in Ansible, and how is it used?

* + Answer: The become directive in Ansible is used to gain elevated privileges when executing tasks. It is useful for performing administrative tasks that require root or sudo privileges. You can use it at the playbook level, the play level, or the task level.

yaml

Copy code

- hosts: all

become: yes

tasks:

- name: Update the package list

apt:

update\_cache: yes

21. Scenario: You need to deploy different versions of an application on different environments (development, staging, production) using Ansible. How would you achieve this?

* + Answer:

yaml

Copy code

- hosts: all

vars\_files:

- "vars/{{ ansible\_env }}.yml"

tasks:

- name: Deploy application

copy:

src: "/path/to/{{ app\_version }}/myapp.jar"

dest: "/opt/myapp/myapp.jar"

Here, ansible\_env and app\_version variables are defined in environment-specific variable files (e.g., development.yml, staging.yml, production.yml).

22. Scenario: How would you use Ansible to ensure that a service is always running on a set of servers and restart it if it stops?

* + Answer:

yaml

Copy code

- hosts: all

tasks:

- name: Ensure the service is running

service:

name: myservice

state: started

enabled: yes

This playbook uses the service module to ensure the service is started and enabled to run at boot.

23. Scenario: How would you roll back a deployment if something goes wrong using Ansible?

* + Answer: Implement a rollback strategy using Ansible handlers and notify directives.

To roll back a deployment using Ansible when something goes wrong, you can structure your playbook to include a mechanism that detects failures and automatically triggers a rollback to restore the previous version. Here’s a step-by-step approach:

### Ansible Playbook for Rollback

yaml

Copy code

- hosts: all

tasks:

- name: Save current version as backup

copy:

src: "/opt/myapp/myapp.jar"

dest: "/opt/myapp/myapp.jar.bak"

- block:

- name: Deploy new application version

copy:

src: "/path/to/new\_version/myapp.jar"

dest: "/opt/myapp/myapp.jar"

notify:

- restart myapp

rescue:

- name: Rollback myapp

copy:

src: "/opt/myapp/myapp.jar.bak"

dest: "/opt/myapp/myapp.jar"

notify: restart myapp

always:

- name: Cleanup temporary files

file:

path: /path/to/temp/files

state: absent

handlers:

- name: restart myapp

service:

name: myapp

state: restarted

24. Scenario: How do you handle dependencies between roles in Ansible?

* + Answer: Ansible roles can have dependencies specified in their meta/main.yml file. This ensures that certain roles are applied before others.

yaml

Copy code

dependencies:

- { role: common, some\_parameter: 3 }

- { role: apache }

25. Scenario: You need to execute a playbook that requires user input at runtime. How do you achieve this in Ansible?

* + Answer: Use the vars\_prompt directive to prompt the user for input.

yaml

Copy code

- hosts: all

vars\_prompt:

- name: "app\_version"

prompt: "Enter the application version to deploy"

private: no

tasks:

- name: Deploy application

copy:

src: "/path/to/{{ app\_version }}/myapp.jar"

dest: "/opt/myapp/myapp.jar"

**26**. What are Ansible plugins and how do you use them?

* + Answer: Ansible plugins extend the core functionality of Ansible. They can be used for inventory, cache, callback, connection, lookup, and other operations. Plugins are typically written in Python and can be custom-developed or sourced from the community. You use plugins by configuring them in the ansible.cfg file or referencing them directly in playbooks.

27. Explain the difference between static and dynamic inventories in Ansible.

* + Answer: A static inventory is a fixed list of hosts defined in a plain text file. It is simple but not suitable for dynamic environments. A dynamic inventory is generated at runtime using a script or plugin that pulls data from external sources (e.g., cloud providers, CMDBs). This allows Ansible to adapt to changes in infrastructure without manual updates.

28. What is Ansible Tower, and what benefits does it provide?

* + Answer: Ansible Tower is a commercial web-based solution that provides an enterprise-grade interface for Ansible. It includes features like role-based access control, job scheduling, real-time job status updates, logging, and notifications. It helps organizations manage complex deployments and ensures better collaboration among teams.

29. How do you handle idempotency in Ansible modules?

* + Answer: Idempotency ensures that applying the same Ansible module multiple times has the same effect as applying it once. Many built-in Ansible modules are inherently idempotent. To ensure idempotency, you should:
  + Use state-based modules (e.g., file, service, package).
  + Check for conditions before making changes (e.g., using when).
  + Avoid using shell commands unless necessary, and ensure they check the current state before making changes.

30. What is the difference between include and import statements in Ansible?

* + Answer: The include statement is used to include tasks or playbooks dynamically at runtime, meaning they can be controlled with conditionals. The import statement is used to include tasks or playbooks statically at parse time, meaning they are included and parsed before any tasks are run. Use import for simpler, static inclusion and include when you need more dynamic control.

31. How do you manage large projects with Ansible?

* + Answer: Managing large projects in Ansible can be done by:
  + Organizing playbooks and roles into a clear directory structure.
  + Using roles to modularize tasks and variables.
  + Grouping variables and inventory configurations by environment.
  + Using Ansible Galaxy to manage and share roles.
  + Implementing CI/CD pipelines to automate testing and deployment of playbooks.

32. Scenario: How would you ensure that a specific package version is installed on all servers?

* + Answer:

yaml

Copy code

- hosts: all

tasks:

- name: Ensure specific package version is installed

package:

name: mypackage

state: present

version: 1.2.3

33. Scenario: You need to perform a rolling update of an application across multiple servers to minimize downtime. How would you achieve this?

* + Answer:

yaml

Copy code

- hosts: all

serial: 1

tasks:

- name: Update application

command: /path/to/update\_script.sh

Using the serial directive ensures that updates are performed on one server at a time, reducing the risk of widespread downtime.

34. Scenario: How do you ensure that a playbook only runs if a specific condition is met (e.g., a file exists on the target node)?

* + Answer:

yaml

Copy code

- hosts: all

tasks:

- name: Check if the file exists

stat:

path: /path/to/file

register: file\_check

- name: Run playbook if file exists

include\_tasks: other\_playbook.yml

when: file\_check.stat.exists

35. Scenario: How would you handle different configurations for different environments (e.g., development, staging, production) within the same playbook?

* + Answer:

yaml

Copy code

- hosts: all

vars\_files:

- "vars/{{ ansible\_env }}.yml"

tasks:

- name: Apply configuration

template:

src: "templates/myapp.conf.j2"

dest: "/etc/myapp/config"

Here, ansible\_env is a variable that specifies the environment, and corresponding variable files and templates are used.

36. Scenario: How do you perform a task only if a service is running on the target node?

* + Answer:

yaml

Copy code

- hosts: all

tasks:

- name: Check if the service is running

service\_facts:

- name: Perform task if service is running

debug:

msg: "Service is running"

when: "'myservice' in ansible\_facts.services and ansible\_facts.services.myservice.state == 'running'"