**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'"

**37. Patching using Ansible Playbook ?**

Ans:-

### 1. Set Up Your Inventory

First, create an inventory file that lists the hosts you want to manage. This file, usually named hosts or inventory, specifies the target machines.

hosts

ini

Copy code

[ubuntu\_servers]

server1.example.com

server2.example.com

### 2. Write Your Ansible Playbook

Create an Ansible playbook to handle the patching process. The playbook will include tasks to update the package list, upgrade installed packages, remove unnecessary packages, and reboot if necessary.

patching.yml

yaml

Copy code

---

- name: Patching and upgrading Ubuntu servers

hosts: ubuntu\_servers

become: yes

tasks:

- name: Update the package list

apt:

update\_cache: yes

- name: Upgrade all packages

apt:

upgrade: dist

autoclean: yes

autoremove: yes

- name: Reboot the server if necessary

reboot:

msg: "Reboot initiated by Ansible for kernel updates."

connect\_timeout: 5

reboot\_timeout: 600

pre\_reboot\_delay: 0

post\_reboot\_delay: 30

when: "'{{ ansible\_kernel }}' not in ['4.15.0-142-generic', '4.15.0-141-generic']"

### 3. Execute the Playbook

Run the Ansible playbook to apply patches and updates to your servers.

sh

Copy code

ansible-playbook -i hosts patching.yml

### Detailed Explanation

* Inventory File: The hosts file lists the servers to be patched. Replace server1.example.com and server2.example.com with your actual server addresses or IPs.
* Playbook (patching.yml):
  + Update the package list: The apt module with update\_cache: yes ensures that the package list is up to date.
  + Upgrade all packages: The apt module with upgrade: dist performs a full upgrade, similar to apt dist-upgrade, which handles package dependencies and can install or remove packages as needed.
  + Reboot the server if necessary: The reboot module reboots the server if there are kernel updates. The when condition ensures that the server is only rebooted if the current kernel version is not in the specified list. Adjust the kernel versions as necessary based on your environment.

### Additional Considerations

* Dry Run: Test your playbook with a dry run to ensure it performs as expected without making any changes.

sh

Copy code

ansible-playbook -i hosts patching.yml --check

* Verbose Output: Use the -v option to get more detailed output for debugging purposes.

sh

Copy code

ansible-playbook -i hosts patching.yml -v

* Error Handling: Implement error handling in your playbook to manage failures gracefully. For example, you can use the ignore\_errors directive or the failed\_when condition to control the behavior of tasks.
* Security Updates Only: If you want to apply only security updates, you can adjust the playbook accordingly.

yaml

Copy code

- name: Upgrade security packages only

apt:

upgrade: safe

only\_upgrade: yes

By using Ansible for patching, you can automate the process, ensuring consistency and reducing the time and effort required to keep your systems up to date.

**38. What is forking & how does it work ?**

**Ans:-**

Forking in Ansible refers to the ability to run multiple tasks in parallel. By default, Ansible forks a specific number of processes to run tasks on multiple hosts concurrently. This number can be controlled using the forks parameter in the Ansible configuration file (ansible.cfg).

For example, in ansible.cfg:

ini

Copy code

[defaults]

forks = 10

This configuration sets the number of parallel processes to 10.

**39. How can you control the number of parallel tasks executed in Ansible ?**

**Ans:-**

The number of parallel tasks executed by Ansible can be controlled using the forks parameter in the ansible.cfg file. Additionally, you can override this setting by using the -f or --forks option when running an Ansible playbook.

Example:

bash

Copy code

ansible-playbook -f 20 myplaybook.yml

This command will run the playbook with 20 parallel forks.

**40. What are the potential issues with increasing the number of forks too high in Ansible, and how can you mitigate them?**

Ans:-

Increasing the number of forks too high can lead to resource exhaustion on the Ansible control node, such as CPU and memory overload. It can also lead to network congestion if the control node is sending too many simultaneous requests.

To mitigate these issues:

* Monitor the control node's resources and adjust the forks value accordingly.
* Use Ansible strategies like free or linear to better control task execution.
* Break down large playbooks into smaller ones to distribute the load.

**41. How can you use the async and poll keywords to manage long-running tasks in Ansible?**

**Ans:-**

The async keyword allows tasks to run asynchronously, meaning they can run in the background and not block the playbook execution. The poll keyword is used to control how frequently Ansible checks for the completion of the task.

Example:

yaml

Copy code

- name: Long running task

command: /usr/bin/long\_running\_task

async: 600

poll: 5

In this example, the task will run for up to 600 seconds (10 minutes) in the background, and Ansible will poll every 5 seconds to check its status.

**42. Can you describe a scenario where you would use the run\_once keyword in Ansible ?**

**Ans:-**

The run\_once keyword is used to ensure that a task is only run a single time, regardless of the number of hosts targeted by the play. This is useful for tasks that only need to be performed once, such as setting up a shared resource or generating a report.

Example:

yaml

Copy code

- hosts: all

tasks:

- name: Generate report

command: /usr/bin/generate\_report

run\_once: true

In this example, the generate\_report command will only be executed once, even if there are multiple hosts in the all group.

These questions and answers should help you demonstrate your knowledge of forking and parallel execution in Ansible during an interview for an experienced position.

**43. What is Control Persist in Ansible and why is it important ?**

**Ans:-**

Control Persist is an SSH feature that keeps connections to remote nodes open, allowing multiple tasks to reuse the same connection. This reduces the overhead of establishing a new SSH connection for each task, leading to faster execution of playbooks.

In Ansible, Control Persist is enabled by default. It helps improve performance, especially when running tasks that require multiple SSH connections.

**44. How do you enable or configure Control Persist in Ansible ?**

**Ans:-**

Control Persist is enabled by default in Ansible. However, you can customize its settings in the ansible.cfg file under the [ssh\_connection] section. The key parameters are control\_path, control\_path\_dir, pipelining, control\_persist, and control\_master.

Example configuration in ansible.cfg:

ini

Copy code

[ssh\_connection]

control\_path = %(directory)s/%%h-%%r

control\_path\_dir = ~/.ansible/cp

pipelining = True

control\_persist = 60s

* control\_path: Specifies the path to store the SSH control sockets.
* control\_path\_dir: Directory where control paths are stored.
* pipelining: Enables SSH pipelining to reduce the number of SSH operations required to execute commands.
* control\_persist: Sets the time for which the connection will be kept open after the last command has been executed.

**45. What is SSH pipelining in Ansible, and how does it relate to Control Persist?**

**Ans:-**

SSH pipelining is a feature that reduces the number of SSH operations needed to execute a task. By enabling pipelining, Ansible can execute commands without creating temporary files on the managed nodes, thus speeding up the playbook execution.

Pipelining is often used in conjunction with Control Persist to further optimize performance. Both features together minimize the overhead of SSH connections and command execution.

To enable pipelining in ansible.cfg:

ini

Copy code

[ssh\_connection]

pipelining = True

**46. What are the potential issues when using Control Persist, and how can you troubleshoot them ?**

**Ans:-**

Potential issues with Control Persist include:

* Stale Control Sockets: If a control socket remains after the playbook execution, it might cause connection failures.
* File Descriptors: A high number of parallel connections can exhaust the file descriptor limit on the control node.
* SSH Configuration Conflicts: Custom SSH configurations might conflict with Ansible's settings.

To troubleshoot:

* Stale Control Sockets: Clean up the control path directory (~/.ansible/cp) regularly.
* File Descriptors: Increase the file descriptor limit on the control node.
* SSH Configuration: Ensure that custom SSH configurations do not interfere with Ansible's SSH settings.

**47. Describe a scenario where disabling Control Persist might be beneficial ?**

**Ans:-**

Disabling Control Persist might be beneficial in scenarios where:

* Security Policies: Strict security policies require closing SSH connections immediately after use.
* Short-lived Commands: Playbooks with very short-lived commands might not benefit significantly from Control Persist.
* Resource Constraints: Systems with limited resources might be overwhelmed by keeping multiple SSH connections open.

To disable Control Persist, you can set control\_persist to no in the ansible.cfg file:

ini

Copy code

[ssh\_connection]

control\_persist = no

**48. What is the purpose of the ansible-config dump command ?**

**Ans:-**

The ansible-config dump command is used to display the current configuration settings that Ansible is using. This includes settings from the ansible.cfg file, environment variables, command-line options, and default settings. It helps users understand what configuration Ansible is applying during execution and is useful for troubleshooting and verification.

**49. How can you use the ansible-config dump command to troubleshoot configuration issues?**

**Ans:-**

You can use the ansible-config dump command to output all the configuration settings Ansible is currently using. By examining this output, you can identify any discrepancies between expected and actual configurations. This is particularly useful for resolving issues related to connection settings, inventory files, roles paths, and other configuration parameters.

Example command:

bash

Copy code

ansible-config dump

**50. Wha**t is the significance of different sources (like Defaults, Config File, Env, Command Line) in the ansible-config dump output?

Ans:-

The ansible-config dump output shows configuration settings along with their sources. The sources indicate where the settings are being derived from:

* Defaults: Default values set by Ansible.
* Config File: Values set in the ansible.cfg file.
* Env: Values set via environment variables.
* Command Line: Values set via command-line options.

This information helps in understanding the precedence of configuration settings and can aid in debugging issues where an expected setting is overridden by another source.

51. How can you filter the output of ansible-config dump to show only specific settings?

Ans:-

You can filter the output of ansible-config dump using the --only-changed option to display only the settings that have been changed from their default values. This helps in quickly identifying custom configurations that might be impacting Ansible's behavior.

Example command:

bash

Copy code

ansible-config dump --only-changed

52. What is fileglob in Ansible, and how is it used?

Ans:-

fileglob is a lookup plugin in Ansible that allows you to match file paths using wildcard patterns. It returns a list of file paths that match the given pattern. It is commonly used to iterate over files in a directory or to include multiple files in a task.

**53. How would you use fileglob to copy all .conf files from a local directory to a remote server?**

Ans:-

- name: Copy all .conf files to remote server

hosts: all

tasks:

- name: Copy configuration files

copy:

src: "{{ item }}"

dest: /etc/myapp/

with\_fileglob:

- /path/to/local/configs/\*.conf

**54. What are the differences between fileglob and with\_fileglob ?**

**Ans:-**

fileglob is a lookup plugin used to return a list of files matching a pattern. with\_fileglob is a looping construct that allows you to iterate over files returned by the fileglob lookup within a task.

**55. Can you use fileglob to match directories? If so, how?**

**Ans:-**

Yes, fileglob can match directories as well. You need to specify a pattern that matches directories. For example:

- name: List directories

hosts: localhost

tasks:

- name: Get list of directories

debug:

msg: "{{ lookup('fileglob', '/path/to/parent\_dir/\*') }}"

This will include both files and directories in the result. To filter directories specifically, additional logic is required.

**56. You want to archive all log files in the /var/logs/myapp/ directory that are older than 7 days using Ansible. How would you achieve this?**

**Ans:-**

- name: Archive old log files

hosts: all

tasks:

- name: Find old log files

find:

paths: /var/logs/myapp/

age: 7d

patterns: '\*.log'

register: old\_logs

- name: Archive log files

command: tar czf /var/logs/myapp/old\_logs.tar.gz {{ item.path }}

with\_items: "{{ old\_logs.files }}"

**57. What is the lineinfile module in Ansible, and when would you use it?**

**Ans:-**

The lineinfile module is used to ensure a particular line is present in a file or to remove a line from a file. It's commonly used to manage configuration files, ensuring specific content is present or absent without replacing the entire file. It is especially useful for making idempotent changes to files.

**58.** How do you ensure a line is present in a file using the lineinfile module?

Ans:- You can ensure a line is present using the lineinfile module by specifying the path to the file and the line you want to add. For example:

yaml

Copy code

- name: Ensure a line is present in /etc/hosts

ansible.builtin.lineinfile:

path: /etc/hosts

line: '127.0.0.1 myhostname'

**59.** How can you ensure that a specific line is absent from a file?

Ans:-You can ensure a line is absent by setting the state parameter to absent. For example:

- name: Ensure a line is absent in a configuration file

ansible.builtin.lineinfile:

path: /etc/someconfig.conf

line: 'unwanted\_line'

state: absent

**60. Explain how you can replace a line that matches a regular expression ?**

**Ans:-**

To replace a line matching a regular expression, use the regexp parameter to specify the pattern and the line parameter for the new line. For example:

yaml

Copy code

- name: Modify a line in a configuration file

ansible.builtin.lineinfile:

path: /etc/myapp.conf

regexp: '^parameter='

line: 'parameter=new\_value'

**61. How do you insert a line after a specific pattern using the lineinfile module?**

**Ans:-**

Use the insertafter parameter to specify the pattern after which the new line should be inserted. For example:

- name: Insert a line after a match

ansible.builtin.lineinfile:

path: /etc/myapp.conf

insertafter: '^# Insert lines after this comment'

line: 'new\_configuration\_line'

**62. What is the difference between insertafter and insertbefore in the lineinfile module?**

**Ans:-**

The insertafter parameter inserts the line after the first match of the specified pattern, while insertbefore inserts the line before the first match of the specified pattern. For example:

- name: Insert a line after a match

ansible.builtin.lineinfile:

path: /etc/myapp.conf

insertafter: '^# Insert lines after this comment'

line: 'new\_configuration\_line'

- name: Insert a line before a match

ansible.builtin.lineinfile:

path: /etc/myapp.conf

insertbefore: '^# Insert lines before this comment'

line: 'new\_configuration\_line'

**63. How can you ensure that a file is created if it doesn’t exist, using the lineinfile module?**

**Ans:-**

By setting the create parameter to yes, you can ensure that the file is created if it does not exist. For example:

yaml

Copy code

- name: Ensure a file is created and a line is added

ansible.builtin.lineinfile:

path: /etc/myapp.conf

line: 'configuration\_line'

create: yes

**64. Describe a scenario where using lineinfile would be preferred over using copy or template modules ?**

**Ans:-**

The lineinfile module is preferred when you need to make small changes to an existing file without replacing the entire file, which might contain other configurations or comments that need to be preserved. For example, updating a single parameter in a large configuration file where other settings should remain unchanged.

**65. Can you use lineinfile to modify multiple lines at once? If not, what would you recommend?**

**Ans:-**

No, lineinfile modifies only one line at a time. To modify multiple lines, you can use multiple lineinfile tasks in your playbook or consider using the blockinfile module if you need to add or manage a block of lines.

66. Explain a scenario where you would need to use retries in a playbook ?

Ans:-

Retries are useful in scenarios where a task might fail due to temporary issues, such as waiting for a service to start, checking for the availability of an external resource, or dealing with transient network issues. For example, ensuring that a web server is up and running after a restart:

yaml

Copy code

- name: Ensure web server is running

ansible.builtin.uri:

url: http://localhost

status\_code: 200

register: result

until: result.status == 200

retries: 10

delay: 5

**67. What is the purpose of the meta module in Ansible?**

**Ans:-**

The meta module is used to control the flow of playbook execution. It provides various functionalities such as ending a play, flushing handlers, clearing facts, refreshing inventory, and resetting connections. It is also used to define role dependencies within a role.

**68. How do you end the execution of a play using the meta module?**

**Ans:-**

**You can end the execution of a play by using the meta: end\_play directive within a task.**

- name: End the current play

meta: end\_play

**69. When would you use meta: flush\_handlers in a playbook?**

**Ans:-**

You would use meta: flush\_handlers when you want to force all notified handlers to run immediately, instead of waiting until the end of the play. This is useful when a change needs to take effect immediately to ensure subsequent tasks run correctly.

- name: Force handlers to run

meta: flush\_handlers

**70.** How can you clear a specific fact or all facts from a host using the meta module?

**Ans:-**

You can clear a specific fact by specifying the key parameter, or clear all facts by omitting the key parameter.

Example: Clearing a specific fact

- name: Clear a specific fact

meta: clear\_facts

key: my\_fact

**71. Explain the use of meta: refresh\_inventory and provide an example scenario ?**

**Ans:-**

The meta: refresh\_inventory directive is used to refresh the inventory during playbook execution. This is useful in scenarios where the inventory is dynamic and might change during the playbook run, such as when new hosts are added or removed.

Example:

yaml

Copy code

- name: Refresh inventory

meta: refresh\_inventory

**72. What is the primary purpose of the fetch module in Ansible ?**

**Ans:-**

The primary purpose of the fetch module is to transfer files from remote hosts to the local host. It is commonly used to retrieve logs, configuration files, or any other files from managed nodes.

**73. How do you use the fetch module to transfer a file without preserving the remote directory structure ?**

**Ans:-**

You can use the flat option and set it to yes to transfer a file without preserving the remote directory structure.

yaml

Copy code

- name: Fetch file without preserving directory structure

ansible.builtin.fetch:

src: /var/log/myapp.log

dest: /tmp/logs/

flat: yes

**74. What happens if the file specified in the src parameter does not exist on the remote host and fail\_on\_missing is set to yes ?**

**Ans:-**

If the file specified in the src parameter does not exist on the remote host and fail\_on\_missing is set to yes, the task will fail. This is the default behavior.

**75. How can you configure the fetch module to ignore missing files and continue execution?**

**Ans:-**

You can set the fail\_on\_missing parameter to no to configure the fetch module to ignore missing files and continue execution.

- name: Fetch file and ignore if it is missing

ansible.builtin.fetch:

src: /var/log/nonexistent.log

dest: /tmp/logs/

fail\_on\_missing: no

**76. Explain the use of the dest parameter in the fetch module ?**

**Ans:-**

The dest parameter specifies the local path where the fetched file will be stored. This can be a directory or a file path, depending on whether the flat option is used.

**77. How would you handle fetching multiple files from remote hosts to a specific local directory ?**

**Ans:-**

You can loop over the list of files and use the fetch module within the loop.

- name: Fetch multiple files from remote hosts

ansible.builtin.fetch:

src: "{{ item }}"

dest: /tmp/logs/{{ inventory\_hostname }}/

loop:

- /var/log/myapp.log

- /var/log/myapp.error.log

**78. How can you handle a task that needs to check for the existence of multiple files, retrying until all files are present?**

**Ans:-** You can use the loop directive to iterate over the list of files and the until directive within each iteration to check for the existence of each file.

- name: Check for the existence of multiple files

ansible.builtin.stat:

path: "{{ item }}"

register: file\_stat

until: file\_stat.stat.exists

retries: 5

delay: 10

loop:

- /path/to/file1

- /path/to/file2

- /path/to/file3

**79.** Explain a scenario where you would need to use the until directive in a playbook ?

Ans:-

A common scenario for using the until directive is when waiting for a service to start after installation. For example, after starting a database service, you might use the until directive to repeatedly check the service status until it is running.

**80.** What is the primary purpose of the raw module in Ansible?

**Ans:-**

The primary purpose of the raw module is to execute commands directly on remote machines without using Python on the remote host. It is useful for bootstrapping systems or running commands on devices with limited capabilities.

**81. How does the raw module differ from the command and shell modules ?**

**Ans:-**

The raw module directly executes the command over SSH without requiring Python, whereas the command and shell modules require Python on the remote host. The raw module is less feature-rich but useful for systems without Python.

**82.** Provide an example of using the raw module to install Python on a remote host ?

Ans:-

- name: Ensure Python is installed

ansible.builtin.raw: |

if ! command -v python3 >/dev/null 2>&1; then

apt-get update && apt-get install -y python3

fi

**83. What are the limitations of the raw module compared to other modules ?**

**Ans:-**

The raw module lacks many of the features and safeguards of other modules. It does not handle return codes, environment variables, or remote execution in the same structured way. It is also less portable because it directly executes commands over SSH.

**84. Can you use the raw module with Windows hosts ?**

**Ans:-**

The raw module is primarily designed for Unix-like systems. For Windows hosts, it is better to use modules like win\_command and win\_shell which are specifically designed for Windows environments.

**85. What are the async and poll parameters used for in Ansible ?**

**Ans:-**

The async parameter specifies the maximum amount of time a task can run, while the poll parameter specifies the interval at which Ansible will check the task's status. This is useful for managing long-running tasks.

**86. What is the purpose of the assert module in Ansible ?**

**Ans:-**

The assert module is used to validate conditions and ensure that certain expressions are true. If any of the conditions are false, the task fails, and the playbook stops execution. It is useful for enforcing preconditions and verifying system states.

**87. Provide an example of using the assert module to check multiple conditions ?**

**Ans:-**

Here is an example that checks if the distribution is Ubuntu and the version is at least 18.04:

- name: Assert multiple conditions

ansible.builtin.assert:

that:

- ansible\_distribution == "Ubuntu"

- ansible\_distribution\_version | version\_compare('18.04', '>=')

**88. What happens if an assertion fails in a playbook ?**

**Ans:-**

If an assertion fails, the task fails, and the playbook stops execution. This helps to prevent further tasks from running when the required conditions are not met.

**89.**