1. \*\*Question: What is Ansible, and how does it differ from other configuration management tools?\*\*

- \*\*Answer:\*\* Ansible is an open-source automation tool used for configuration management, application deployment, and task automation. It differs from other tools by not requiring agent installation on managed nodes.

2. \*\*Question: Explain the concept of idempotence in Ansible.\*\*

- \*\*Answer:\*\* Idempotence means that running a task multiple times has the same result as running it once. In Ansible, this ensures that applying the same playbook multiple times does not cause unnecessary changes if the desired state is already achieved.

### Playbooks and Tasks:

3. \*\*Question: What is an Ansible playbook?\*\*

- \*\*Answer:\*\* An Ansible playbook is a YAML file that defines a set of tasks to be executed on remote hosts. Playbooks are used for automation and orchestration.

4. \*\*Question: How can you parameterize a playbook in Ansible?\*\*

- \*\*Answer:\*\* Playbooks can be parameterized using variables. Variables can be defined in the playbook or passed as external parameters using `-e` during playbook execution.

### Roles and Organizing Playbooks:

5. \*\*Question: What is an Ansible role, and how is it useful in playbooks?\*\*

- \*\*Answer:\*\* An Ansible role is a way to organize tasks, variables, and handlers in a structured directory hierarchy. Roles make playbooks more modular and reusable.

6. \*\*Question: Explain the purpose of the `handlers` section in Ansible playbooks.\*\*

- \*\*Answer:\*\* The `handlers` section contains tasks that are triggered by other tasks in the playbook. Handlers are executed only if notified by a task, and they are often used to restart services or perform other actions in response to changes.

### Modules and Tasks:

7. \*\*Question: What is an Ansible module? Can you give an example?\*\*

- \*\*Answer:\*\* An Ansible module is a reusable, standalone script that Ansible uses to perform tasks on remote nodes. Examples include the `apt` module for package management and the `copy` module for file copying.

8. \*\*Question: How can you run a shell command on remote hosts using Ansible?\*\*

- \*\*Answer:\*\* The `command` or `shell` module can be used. For

- name: Run a shell command

command: ls /path/to/directory

### Inventory and Hosts:

9. \*\*Question: What is an Ansible inventory file?\*\*

- \*\*Answer:\*\* An Ansible inventory file lists the hosts on which Ansible should run tasks. It can be static or dynamic and can contain groups of hosts.

10. \*\*Question: How can you limit the execution of a playbook to specific hosts?\*\*

- \*\*Answer:\*\* The `-l` option can be used to limit the playbook execution to specific hosts or groups. For example:

```bash

ansible-playbook -i inventory.ini -l web\_servers playbook.yml

```

### Best Practices and Troubleshooting:

11. \*\*Question: What are some best practices for writing Ansible playbooks?\*\*

- \*\*Answer:\*\* Best practices include using roles for organization, using idempotent tasks, providing clear documentation, and using Ansible Vault for sensitive data.

12. \*\*Question: How do you troubleshoot a failed Ansible playbook?\*\*

- \*\*Answer:\*\* Troubleshooting steps include checking task output, inspecting logs on the remote hosts, using the `--check` option for dry runs, and increasing verbosity with the `-v` option.

### Integration and Extensibility:

13. \*\*Question: How can Ansible be integrated into a continuous integration (CI) pipeline?\*\*

- \*\*Answer:\*\* Ansible playbooks can be executed as part of CI/CD pipelines using tools like Jenkins or GitLab CI. This enables automated infrastructure provisioning and configuration.

14. \*\*Question: Can you write custom Ansible modules?\*\*

- \*\*Answer:\*\* Yes, Ansible allows the creation of custom modules in various programming languages. Custom modules extend Ansible's functionality for specific use cases.

### Security:

15. \*\*Question: How does Ansible handle security considerations?\*\*

- \*\*Answer:\*\* Ansible follows security best practices by using SSH for communication, supporting encrypted connections, and providing Ansible Vault for securing sensitive data.

16. \*\*Question: What is Ansible Vault, and how is it used?\*\*

- \*\*Answer:\*\* Ansible Vault is a feature that allows the encryption of sensitive data such as passwords and secret keys. It ensures secure storage and transmission of confidential information.

These questions cover a range of Ansible topics and can be adjusted based on the specific skills and knowledge areas relevant to your organization's DevOps practices.

---

- name: Deploy a web application

hosts: web\_servers

roles:

- common

- web\_app

The corresponding directory structure would be:

my\_playbook/

|-- roles/

| |-- common/

| | |-- tasks/

| | | └── main.yml

| | └── ...

| |-- web\_app/

| |-- tasks/

| | └── main.yml

| └── ...

|-- my\_playbook.yml

Playbook for Copying files to remote server \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

---

- name: copy files to remote

copy:

src: /srv/myfiles/foo.conf

dest: /etc/foo.conf

owner: foo

group: foo

mode: u+rw,g-wx,o-rwx

- name: Copy file with owner and permissions

copy:

src: /srv/myfiles/foo.conf

dest: /etc/foo.conf

owner: foo

group: foo

mode: '0644'

**You can execute ansible by below ways**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ansible playbookname.yml

ansible-playbook playbookname.yml --check

ansible-playbook playbookname.yml --start-at-task "task name"

ansible-playbook playbookname.yml -t <tagname>

**\*\*\* List out inventory list**

ansible-inventory --list

OR

ansible all --list-hosts

**Ping localhost**

# ansible localhost –m ping

**Creating a file on all remote clients**

# ansible all –m file –a “path=/home/iafzal/adhoc1 state=touch mode=700”

**Deleting a file on all remote clients**

# ansible all –m file –a “path=/home/iafzal/adhoc1 state=absent”

**Copying a file to remote clients**

# ansible all –m copy –a “src=/tmp/adhoc2 dest=/home/iafzal/adhoc2”

**Installing package (telnet and httpd-manual)**

# ansible all –m yum –a “name=telnet state=present”

# ansible all –m yum –a “name=httpd-manual state=present”.

**Starting httpd package service**

# ansible all –m service –a “name=httpd state=started”

**Start httpd and enable at boot time**

# ansible all –m service –a “name=httpd state=started enabled=yes”

**Checking httpd service status on remote client**

# ansible all –m shell -a “systemctl status httpd”

**Remove httpd package**

# ansible all –m yum –a “name=httpd state=absent”

OR

# ansible all –m shell -a “yum remove httpd”.

**Creating a user on remote clients**

# ansible all –m user –a “name=jsmith home=/home/jsmith shell=/bin/bash state=present”

**To add a user to a different group**

# ansible all –m user –a “name=jsmith group=iafzal”

**Deleting a user on remote clients**

# ansible all –m user –a “name=jsmith home=/home/jsmith shell=/bin/bash state=absent”

OR

# ansible all –m shell –a “userdel jsmith”

**Getting system information from remote clients**

# ansible all –m setup

**You can run commands on the remote host without a shell module e.g. reboot client1**

# ansible client1 –a “/sbin/reboot”

**Pick & Choose steps :**

**start a playbook at a specific task**

#ansible-playbook yamlfile.yml --start-at-task ‘Task Name’

#ansible-playbook http.yml --start-at-task ‘Install telenet’

\*\*\* cron job entry

---

- name: Create Cron Job

hosts: target\_servers

become: true # Run tasks with elevated privileges (sudo)

vars:

cron\_job\_name: "my\_cron\_job"

cron\_schedule: "\*/5 \* \* \* \*" # Run every 5 minutes

command\_to\_run: "/path/to/your/command"

tasks:

- name: Add the cron job

cron:

name: "{{ cron\_job\_name }}"

minute: "{{ cron\_schedule.split(' ')[0] }}"

hour: "{{ cron\_schedule.split(' ')[1] }}"

day: "{{ cron\_schedule.split(' ')[2] }}"

month: "{{ cron\_schedule.split(' ')[3] }}"

weekday: "{{ cron\_schedule.split(' ')[4] }}"

job: "{{ command\_to\_run }}"

\*\*\* Install and start apache

---

- name: Install and start Apache

hosts: web\_servers

tasks:

- name: Update package cache

apt:

update\_cache: yes

- name: Install Apache

apt:

name: apache2

state: present

- name: Start Apache service

service:

name: apache2

state: started

pipeline{

**agent**{label 'p29uatap004'}

**parameters**{

choice(name:'ENV', choices:['UAT','DBG','PRD'], description: 'Environment name of sftp server')

string(name:'HOST', defaultValue:'up29-uat-sftp.fr.world.socgen', description:'Hostname of SFTP server')

string (name:'PORT',defaultValue: '22', description: 'port of sftp server')

credential(name:'SSH\_CREDS', defaultValue:'SFTP\_UAT', credentialType: 'SSH Username with private key', description:'SSH key with user for the SFTP server')

}

stages{

stage('stage -1 '){

steps{

script{

withCrekdentials([sshUserPrivateKey(credentialsId:'SSH\_CREDS','KeyFileVariable:'SSH\_KEY', usernameVariable: 'SSH\_USER')]){

def remote=[:]

remote.name='SFTP server'

remote.user=SSH\_USER

remote.host=HOST

remote.port=PORT as Integer

remote.identityFile=SSH\_KEY

remote.allowAnyHosts=true

sshScript remote:remote, script: 'src/scripts\_sh/jenkins/sftp\_folder.sh'

}

}

}

}

}

}

pipeline {

agent any

environment {

REMOTE\_SERVER = 'your-remote-server'

REMOTE\_USER = 'your-remote-user'

SSH\_KEY\_FILE = '/path/to/your/private/key'

}

stages {

**stage('Stop Service') {**

steps {

sh "ssh -o StrictHostKeyChecking=no -i ${env.SSH\_KEY\_FILE} ${env.REMOTE\_USER}@${env.REMOTE\_SERVER} 'chmod +x stop.sh && ./stop.sh'"

}

}

**stage('Start Service') {**

steps {

sh "ssh -o StrictHostKeyChecking=no -i ${env.SSH\_KEY\_FILE} ${env.REMOTE\_USER}@${env.REMOTE\_SERVER} 'chmod +x start.sh && ./start.sh'"

}

}

**stage('Check Service Status') {**

steps {

sh "ssh -o StrictHostKeyChecking=no -i ${env.SSH\_KEY\_FILE} ${env.REMOTE\_USER}@${env.REMOTE\_SERVER} 'chmod +x status.sh && ./status.sh'"

}

}

}

}

pipeline {

agent any

stages {

**stage('Build'**) {

steps {

// Build steps

}

}

**stage('Test')** {

steps {

// Test steps

}

}

// Add more stages as needed

}

}

**What is CI/CD ?**

#CI/CD stands for continues integration and continues delivery. CI is a software development practice in which all developers merge code changes in a central repo multiple times a day.

Continuous Delivery: This is a software development process where changes that are continuously integrated (CI) undergo testing and are deployed into a specific environment. Typically, this deployment is done through a manual release process after all the quality checks have been successfully completed.

Continuous Deployment: Continuous Deployment is a software development practice where the changes continuously integrated (CI) are automatically deployed into the target environment after successful completion of all quality checks. In this approach, the deployment is fully automated, eliminating the need for manual intervention in the release process.

**How to setup credentials in Jenkins?**

# you can add credentials by using a plugin credentials. You can add as ma y credentials and can use them in Jenkins file or configuration settings.

Jenkins>manage Jenkins>manage credentials > add credentials

#**webhook vs POLLSCM**

POLLSCM: In polling, Jenkins periodically checks the version control system for changes. It compares the current state of the repository with the previous state to determine if there are any new changes.

A webhook is a mechanism where the version control system (e.g., Git, GitHub, Bitbucket) notifies Jenkins automatically whenever there is a change in the repository. This notification is sent directly from the version control system to Jenkins.

Most popular 5 plugins of Jenkins:

1. \*\*Pipeline:\*\*The Pipeline plugin is essential for defining and managing Jenkins pipelines. It allows users to define complex build and deployment workflows as code using either the Declarative or Scripted Pipeline syntax.

2. \*\*Git Plugin:\*\* The Git plugin integrates Jenkins with Git version control systems. It enables Jenkins to clone repositories, fetch code changes, and trigger builds based on changes in Git branches.

3. \*\*GitHub Integration Plugin:\*\*This plugin provides seamless integration between Jenkins and GitHub. It allows Jenkins to automatically build and report the status of GitHub pull requests. It also supports webhooks for triggering builds on GitHub events.

4. \*\*Credentials Plugin:\*\* The Credentials plugin provides a centralized and secure way to manage various types of credentials, such as usernames and passwords, secret text, and SSH private keys. It allows Jenkins jobs to use these credentials without exposing sensitive information in the job configuration.

5. \*\*Docker Pipeline:\*\* The Docker Pipeline plugin facilitates the integration of Docker with Jenkins pipelines. It allows users to define and execute Docker-related operations within Jenkins pipelines, such as building and pushing Docker images or running containers.

Please note that the popularity of plugins can change over time, and new plugins may have gained traction since my last update in January 2022. It's recommended to check the Jenkins Plugin Index for the latest and most popular plugins based on user downloads and community activity:

[Jenkins Plugin Index](https://plugins.jenkins.io/)

deployment strategies:

Blue-Green Deployment:\*\* - In a Blue-Green Deployment, two identical production environments (Blue and Green) exist. At any given time, only one environment serves live production traffic. The deployment process involves switching the router or load balancer to direct traffic to the inactive environment, allowing for zero-downtime releases.

Canary Deployment:\*\*- Canary Deployment is a phased release strategy where the new version is initially deployed to a small subset (a "canary group") of users or servers. If the release is successful and no critical issues are detected, the deployment is expanded to the entire user base or infrastructure.

Rolling Deployment:\*\* - In a Rolling Deployment, new versions are gradually deployed across the production infrastructure, one subset at a time. This deployment continues until all instances are updated. This strategy minimizes downtime and allows for easy rollback if issues are detected.

Feature Toggles (Feature Flags):\*\* Feature Toggles involve releasing new functionality in a disabled state. The feature can be activated or deactivated at runtime without deploying new code. This allows for incremental rollouts and easy rollback in case of issues.

.