Ansible

Ansible Scenario based questions: Part 1

In this blog we will go through ansible scenario-based questions from basic to advanced topic including ansible scripts example.

Here are a ten scenario-based questions that could be useful in ansible interview:

-
- You have an ansible playbook that configures a server with a specific package. You want to run the playbook on a group of servers, but you want to skip the task that installs the package on a specific server. How would you do this?

To skip a task for a specific host, you can use the --limit option and specify the hostname or pattern to match the host you want to skip.
ansible-playbook --limit '!server1' playbook.yml
2. You have an ansible playbook that installs and configures a web server. You want to run the playbook, but you want to test the changes before making them. How would you do this?
To test the changes made by an ansible playbook without making any actual changes, you can use the --check flag. This will perform a "dry run" of the playbook and report any changes that would be made, but it will not actually make any changes on the target hosts.
ansible-playbook --check playbook.yml
3. You have an ansible playbook that installs and configures a database server. You want to run the playbook, but you want to make it so that any failures will be ignored, and the playbook will continue to run. How would you do this?
To ignore failures and continue running the playbook, you can use the --force-handlers flag. This will cause ansible to continue running the playbook even if a task fails, and it will run any handlers (tasks that are triggered by other tasks) that were registered for the failed tasks.
ansible-playbook --force-handlers playbook.yml
4. You have an ansible playbook that installs and configures a load balancer. You want to run the playbook, but you want to make it so that any tasks that would change the target hosts will be skipped. How would you do this?
To skip tasks that would make changes to the target hosts, you can use the --diff flag. This will perform a "dry run" of the playbook and report any changes that would be made, but it will skip any tasks that would make actual changes on the target hosts.
ansible-playbook --diff playbook.yml
5. You have an ansible playbook that installs and configures a monitoring system. You want to run the playbook, but you want to make it so that any tasks that are not idempotent (i.e., tasks that cannot be run multiple times without causing harm) will be skipped. How would you do this?
To skip non-idempotent tasks, you can use the --skip-tags flag and specify the tag or tags for the tasks you want to skip.
ansible-playbook --skip-tags non_idempotent playbook.yml

6. You have an ansible playbook that installs and configures a file server. You want to run the playbook, but you want to make it so that any tasks that are not relevant to the target hosts will be skipped. How would you do this?
To skip tasks that are not relevant to the target hosts, you can use the --skip-tags flag and specify the tag or tags for the tasks you want to skip.
ansible-playbook --skip-tags '!file_server' playbook.yml
7. You want to run a shell command on all of the servers in your inventory. How would you do this using ansible?
You can use the command module to run a shell command on all of the servers in your inventory.

```
- name: Run shell command
  hosts: all
  tasks:
    - name: Run command
      command: echo hello
```

8. You want to copy a file from your local machine to all of the servers in your inventory. How would you do this using ansible?
You can use the copy module to copy a file from your local machine to all of the servers in your inventory.

```
- name: Copy file
  hosts: all
  tasks:
    - name: Copy file
      copy:
        src: /path/to/local/file
        dest: /path/on/remote/server
```

9. You want to install a package on all of the servers in your inventory using the package manager for the target operating system. How would you do this using ansible?
You can use the package module to install a package on all of the servers in your inventory using the package manager for the target operating system.

```
- name: Install package
  hosts: all
  tasks:
    - name: Install package
      package:
        name: nginx
```

10. You want to create a user on all of the servers in your inventory. How would you do this using ansible?
You can use the user module to create a user on all of the servers in your inventory.

```
- name: Create user
  hosts: all
  tasks:
    - name: Create user
      user:
        name: newuser
```

```
    password: $6$rounds=656000$somesalt$encryptedpassword
```

———————————————————

Ansible Scenario based questions: Part 2

In this blog we will go through ansible scenario-based questions from basic to advanced topic including ansible scripts examples.

This Part 2 of this Ansible series, if you missed part 1 please go through it first.

- You want to create a group on all of the servers in your inventory and add a user to the group. How would you do this using ansible?

You can use the group module to create a group on all of the servers in your inventory, and the user module to add a user to the group.

```
- name: Create group and add user
  hosts: all
  tasks:
    - name: Create group
      group:
        name: newgroup
        state: present
    - name: Add user to group
      user:
        name: newuser
        groups: newgroup
        append: yes
```

2. You want to create a directory on all of the servers in your inventory and set the owner and group for the directory. How would you do this using ansible?

You can use the file module to create a directory on all of the servers in your inventory and set the owner and group for the directory.

```
- name: Create directory
  hosts: all
  tasks:
    - name: Create directory
      file:
        path: /path/to/new/directory
        state: directory
        owner: newuser
        group: newgroup
```

3. You want to create a configuration file on all of the servers in your inventory and set the contents of the file. How would you do

this using ansible?

You can use the template module to create a configuration file on all of the servers in your inventory, and set the contents of the file using a Jinja2 template.

```
- name: Create configuration file
  hosts: all
  tasks:
    - name: Create configuration file
      template:
        src: /path/to/template
        dest: /path/to/configuration/file
        owner: newuser
        group: newgroup
```

4. You want to start a service on all of the servers in your inventory. How would you do this using ansible?

You can use the service module to start a service on all of the servers in your inventory.

```
- name: Start service
  hosts: all
  tasks:
    - name: Start service
      service:
        name: nginx
        state: started
```

5. You want to reboot all of the servers in your inventory. How would you do this using ansible?

You can use the reboot module to reboot all of the servers in your inventory

```
- name: Reboot servers
  hosts: all
  tasks:
    - name: Reboot servers
      reboot:
        reboot_timeout: 300
```

6. You want to configure a firewall on all of the servers in your inventory. How would you do this using ansible?

You can use the firewall module to configure a firewall on all of the servers in your inventory.

```
- name: Configure firewall
  hosts: all
  tasks:
    - name: Allow HTTP traffic
      firewall:
        name: http
        state: enabled
```

```
      protocol: tcp
      port: 80
```

7. You want to configure a load balancer on all of the servers in your inventory. How would you do this using ansible?

You can use the haproxy module to configure a load balancer on all of the servers in your inventory.

```
- name: Configure load balancer
  hosts: all
  tasks:
    - name: Add backend servers
      haproxy:
        name: backend
        state: present
        servers:
          - name: server1
            address: 1.2.3.4
          - name: server2
            address: 5.6.7.8
```

8. You want to configure a database on all of the servers in your inventory. How would you do this using ansible?

You can use the mysql_user module to configure a database on all of the servers in your inventory.

```
- name: Configure database
  hosts: all
  tasks:
    - name: Create database
      mysql_db:
        name: newdatabase
        state: present
    - name: Create user
      mysql_user:
        name: newuser
        password: newpassword
        priv: 'newdatabase.*:ALL'
        state: present
```

9. You want to configure a network interface on all of the servers in your inventory. How would you do this using ansible?

You can use the interface module to configure a network interface on all of the servers in your inventory.

```
- name: Configure network interface
  hosts: all
  tasks:
    - name: Configure interface
      interface:
        name: eth0
        ipv4:
```

```
    address: 1.2.3.4
    netmask: 255.255.255.0
    gateway: 1.2.3.1
```

10. You want to configure DNS on all of the servers in your inventory. How would you do this using ansible?

You can use the lineinfile module to configure DNS on all of the servers in your inventory.

```
- name: Configure DNS
  hosts: all
  tasks:
    - name: Set nameservers
      lineinfile:
        path: /etc/resolv.conf
        line: 'nameserver 1.2.3.4'
        state: present
```

I hope these examples are helpful! Let me know if you have any other questions. Follow me for more such content and Part 3 will publish soon.

———————————

Ansible Scenario based questions: Part 3

In this blog we will go through ansible scenario-based questions from basic to advanced topic including ansible scripts examples.

This Part 3 of this Ansible series, if you missed part 1 please go through it first.

- You want to configure NTP on all of the servers in your inventory. How would you do this using ansible?

You can use the ntp module to configure NTP on all of the servers in your inventory.

```
- name: Configure NTP
  hosts: all
  tasks:
    - name: Set NTP servers
      ntp:
        servers:
          - 0.pool.ntp.org
          - 1.pool.ntp.org
          - 2.pool.ntp.org
```

2. You want to add a SSH key to the authorized keys file for all users on all of the servers in your inventory. How would you do this using ansible?

You can use the authorized_key module to add a SSH key to the authorized keys file for all users on all of the servers in your

inventory.

```
- name: Add SSH key
  hosts: all
  tasks:
    - name: Add key
      authorized_key:
        user: '{{ item }}'
        key: 'ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCfQGQDYOc1cNGn1xrKrGcJYOoCzNc
XuZDYXV7YcJb1WV7BYxMxr+VrpJyOjkV7+p6b3q6+0pt5l5U6ZxR6+0C2fvG0yL
WVJm9Rg1fQmZv+GJLc8PxoA3brwM1nIbW8M8OTN4s4tNlDtJY+B/8CzGJT/
wLmvT/
1NQ8Kj+9l2R0YgZvb8W98H+zKMeTLq3ixFgv8EfiB4sX9+4K4O4pZ8sWt86JLs
OwT+xyeOJc2W1N/
X9VU6jYU6i9u5vb2Vx5K5J5RV7Zl+nZFVzZiCkKjJ1WWnI1y8E7BjDpf48GnweB/
+zIwZO+N/Y2UVJuu0v1O8I/o/pGJbM0H5O6U+wVUJ3mOj7VX9 newkey'
        state: present
  with_items:
    - root
    - ansible
```

3. You want to configure a web server on all of the servers in your inventory. How would you do this using ansible?

You can use the nginx module to configure a web server on all of the servers in your inventory.

```
- name: Configure web server
  hosts: all
  tasks:
    - name: Install nginx
      package:
        name: nginx
    - name: Create configuration file
      template:
        src: /path/to/template
        dest: /etc/nginx/nginx.conf
        owner: root
        group: root
    - name: Start nginx
      service:
        name: nginx
        state: started
```

4. You want to configure a database server on all of the servers in your inventory. How would you do this using ansible?

You can use the mysql module to configure a database server on all of the servers in your inventory.

```
- name: Configure database server
  hosts: all
```

```
tasks:
  -
```

5. You want to configure a monitoring system on all of the servers in your inventory. How would you do this using ansible?

You can use the zabbix_agent module to configure a monitoring system on all of the servers in your inventory.

```
- name: Configure monitoring
  hosts: all
  tasks:
    - name: Install Zabbix agent
      package:
        name: zabbix-agent
    - name: Create configuration file
      template:
        src: /path/to/template
        dest: /etc/zabbix/zabbix_agentd.conf
        owner: root
        group: root
    - name: Start Zabbix agent
      service:
        name: zabbix-agent
        state: started
```

6. You want to configure a continuous integration (CI) server on all of the servers in your inventory. How would you do this using ansible?

You can use the jenkins module to configure a CI server on all of the servers in your inventory.

```
- name: Configure CI server
  hosts: all
  tasks:
    - name: Install Jenkins
      package:
        name: jenkins
    - name: Start Jenkins
      service:
        name: jenkins
        state: started
```

7. You want to configure a configuration management database (CMDB) on all of the servers in your inventory. How would you do this using ansible?

You can use the mongodb module to configure a CMDB on all of the servers in your inventory.

```
- name: Configure CMDB
  hosts: all
  tasks:
    - name: Install MongoDB
```

```
    package:
      name: mongodb
  - name: Start MongoDB
    service:
      name: mongodb
      state: started
```

8. You want to install and configure a message queue on all of the servers in your inventory. How would you do this using ansible?

You can use the rabbitmq_plugin module to install and configure a message queue on all of the servers in your inventory.

```
- name: Configure message queue
  hosts: all
  tasks:
    - name: Install RabbitMQ
      package:
        name: rabbitmq-server
    - name: Start RabbitMQ
      service:
        name: rabbitmq-server
        state: started
    - name: Enable RabbitMQ management plugin
      rabbitmq_plugin:
        name: rabbitmq_management
        state: enabled
```

9. You want to install and configure a cache on all of the servers in your inventory. How would you do this using ansible?

You can use the memcached module to install and configure a cache on all of the servers in your inventory.

```
- name: Configure cache
  hosts: all
  tasks:
    - name: Install memcached
      package:
        name: memcached
    - name: Start memcached
      service:
        name: memcached
        state: started
```

10. You want to install and configure a search engine on all of the servers in your inventory. How would you do this using ansible?

You can use the elasticsearch module to install and configure a search engine on all of the servers in your inventory.

```
- name: Configure search engine
  hosts: all
  tasks:
    - name: Install Elasticsearch
```

```
    package:
      name: elasticsearch
  - name: Start Elasticsearch
    service:
      name: elasticsearch
      state: started
```

==========

Ansible Scenario based questions: Part 4

In this blog we will go through ansible scenario-based questions from basic to advanced topic including ansible scripts examples.

This Part 4 of this Ansible series, if you missed part 1 please go through it first.

- You want to install and configure a continuous delivery (CD) server on all of the servers in your inventory. How would you do this using ansible?

You can use the gocd_agent module to install and configure a CD server on all of the servers in your inventory.

```
- name: Configure CD server
  hosts: all
  tasks:
    - name: Install GoCD agent
      package:
        name: gocd-agent
    - name: Create configuration file
      template:
        src: /path/to/template
        dest: /etc/go/agent-bootstrapper.properties
        owner: root
        group: root
    - name: Start GoCD agent
      service:
        name: gocd-agent
        state: started
```

2. You want to install and configure a log management system on all of the servers in your inventory. How would you do this using ansible?

You can use the filebeat module to install and configure a log management system on all of the servers in your inventory.

```
- name: Configure log management
  hosts: all
  tasks:
    - name: Install Filebeat
      package:
        name: filebeat
```

```
    - name: Create configuration file
      template:3
        src: /path/to/template
        dest: /etc/filebeat/filebeat.yml
        owner: root
        group: root
    - name: Start Filebeat
      service:
        name: filebeat
        state: started
```

3. You want to install and configure a container orchestration system on all of the servers in your inventory. How would you do this using ansible?
You can use the docker module to install and configure a container orchestration system on all of the servers in your inventory.

```
- name: Configure container orchestration
  hosts: all
  tasks:
    - name: Install Docker
      package:
        name: docker-ce
    - name: Start Docker
      service:
        name: docker
        state: started
```

4. You want to install and configure a virtualization platform on all of the servers in your inventory. How would you do this using ansible?
You can use the kvm module to install and configure a virtualization platform on all of the servers in your inventory.

```
- name: Configure virtualization
  hosts: all
  tasks:
    - name: Install KVM
      package:
        name: qemu-kvm
    - name: Start KVM
      service:
        name: libvirtd
        state: started
```

5. You want to install and configure a database cluster on all of the servers in your inventory. How would you do this using ansible?
You can use the galera_cluster module to install and configure a database cluster on all of the servers in your inventory.

```
- name: Configure database cluster
  hosts: all
```

```
  tasks:
    - name: Install Galera cluster
      package:
        name: galera-3
    - name: Create configuration file
      template:
        src: /path/to/template
        dest: /etc/mysql/conf.d/galera.cnf
        owner: root
        group: root
    - name: Start Galera cluster
      service:
        name: mysql
        state: started
```

6. You want to install and configure a network monitoring system on all of the servers in your inventory. How would you do this using ansible?
You can use the telegraf module to install and configure a network monitoring system on all of the servers in your inventory.

```
- name: Configure network monitoring
  hosts: all
  tasks:
    - name: Install Teleg
```

7. You want to install and configure a continuous integration and delivery (CI/CD) platform on all of the servers in your inventory. How would you do this using ansible?
You can use the gitlab_runner module to install and configure a CI/CD platform on all of the servers in your inventory.

```
- name: Configure CI/CD platform
  hosts: all
  tasks:
    - name: Install GitLab Runner
      package:
        name: gitlab-runner
    - name: Create configuration file
      template:
        src: /path/to/template
        dest: /etc/gitlab-runner/config.toml
        owner: root
        group: root
    - name: Start GitLab Runner
      service:
        name: gitlab-runner
        state: started
```

8. You want to install and configure a log analysis platform on all of the servers

in your inventory. How would you do this using ansible?
You can use the logstash module to install and configure a log analysis platform on all of the servers in your inventory.

```
- name: Configure log analysis platform
  hosts: all
  tasks:
    - name: Install Logstash
      package:
        name: logstash
    - name: Create configuration file
      template:
        src: /path/to/template
        dest: /etc/logstash/logstash.yml
        owner: root
```

9. You want to install and configure a load balancer on all of the servers in your inventory. How would you do this using ansible?
You can use the haproxy module to install and configure a load balancer on all of the servers in your inventory.

```
- name: Configure load balancer
  hosts: all
  tasks:
    - name: Install HAProxy
      package:
        name: haproxy
    - name: Create configuration file
      template:
        src: /path/to/template
        dest: /etc/haproxy/haproxy.cfg
        owner: root
        group: root
    - name: Start HAProxy
      service:
        name: haproxy
        state: started
```

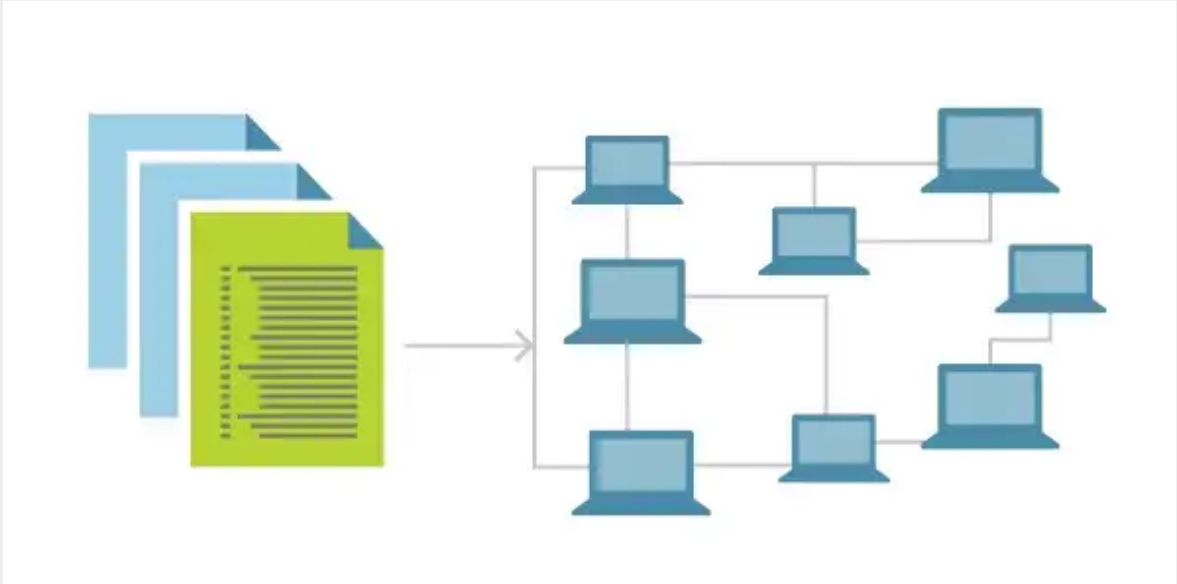
10. You want to install and configure a continuous testing platform on all of the servers in your inventory. How would you do this using ansible?
You can use the jenkins module to install and configure a continuous testing platform on all of the servers in your inventory.

```
- name: Configure continuous testing platform
  hosts: all
  tasks:
    - name: Install Jenkins
      package:
        name: jenkins
    - name: Create configuration file
```

```
    template:
      src: /path/to/template
      dest: /etc/jenkins/jenkins.yml
      owner: root
      group: root
  - name: Start Jenkins
    service:
      name: jenkins
      state: started
```

==========

Ansible DevOps Interview Questions and Answers



IaC source:

What is Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is a methodology for managing and provisioning IT infrastructure through the use of code, rather than through manual processes. This can include things like servers, storage, and networking equipment, as well as software and services that run on top of them.

The idea behind IaC is to treat infrastructure the same way that software developers treat application code: as something that can be versioned, reviewed, and tested before it is deployed to production. By using code to automate the provisioning and management of infrastructure, teams can increase the speed and reliability of their deployments, as well as make it easier to track changes and roll back in case of problems.

There are several tools available for implementing IaC, including:

- Terraform: This is an open-source tool from Hashicorp that allows you to define your infrastructure using HashiCorp Configuration Language (HCL), which is then used to provision and manage resources across multiple cloud providers.
- Ansible : This is an open-source automation tool that can be

used to provision and manage resources in a variety of environments, including cloud providers and on-premises infrastructure. Ansible uses a simple, human-readable language called YAML to define automation tasks, which are then executed on remote systems.

- CloudFormation: This is an AWS service that allows you to define and provision AWS resources using JSON or YAML templates.
- Kubernetes : open-source container orchestration system for automating application deployment, scaling, and management

IaC allows teams to easily provision, manage, and scale infrastructure, and reduces the risk of configuration drift and manual errors that can occur when managing infrastructure manually. It also enables teams to move faster, as they don't have to spend time on manual tasks like spinning up new servers or configuring load balancers.

What are some of the best practices for using IaC?

- Version control: Store your IaC templates in version control so that you can track changes and roll back to previous versions if needed.
- Automate testing: Automate testing of the templates and make sure they are tested before they are deployed.
- Use modules: Break down your templates into reusable modules that can be used across multiple projects. This will make your templates more maintainable and easier to understand.
- Use parameterisation: Make your templates more flexible by parameterising them. This will allow you to easily change values such as server sizes or number of instances without having to modify the template.
- Keep security in mind: Secure infrastructure as code by use of policy-as-code to set up proper access controls and compliance checks
- Continuous Integration and Deployment (CI/CD) : Use a CI/CD pipeline for your infrastructure as code, this will help in automatic validation and deployment of your templates.
- Use established best practices and conventions: Adhere to established best practices and conventions that are commonly used within your organisation or industry.
- Keep documentation up-to-date: Document your templates and make sure they are up-to-date so that others can understand what they do and how they should be used.

Following these best practices, teams can improve the reliability, maintainability, and scalability of their infrastructure, while also reducing the risk of errors and misconfigurations.

What is Idempotency?

This is a a property of certain operations in which applying the operation multiple times has the same effect as applying it just once. In other words, if an operation is idempotent, it can be repeated without changing the result.

In the context of infrastructure as code (IaC), idempotency is an important concept to understand when creating templates or scripts that provision or manage resources. Idempotent scripts or templates will ensure that running them multiple times will not create duplicate resources or cause unintended changes.
Examples of idempotent actions are:

- Creating a new resource if it doesn't exist
- Updating a resource to match a specific configuration if it's different
- Deleting a resource if it exists

An example of non-idempotent action can be:

- Increase the number of instances of a service, because if run multiple times, it will increase the number of instances more than required.

Achieving idempotency is often a desired goal in IaC, and many tools provide mechanisms for achieving it. For example, in Ansible, the idempotence of a task is determined by the state specified in the task. In terraform, the state is stored in a local file called the state file, and terraform uses the state information to determine if any changes need to be made.
Using idempotent scripts and templates, teams can ensure that their infrastructure is in the desired state and avoid unintended changes.
Working process of Ansible?
Ansible is an open-source automation tool that can be used to provision and manage resources in a variety of environments, including cloud providers and on-premises infrastructure. It uses a simple, human-readable language called YAML to define automation tasks, which are then executed on remote systems.
The basic process for using Ansible to manage resources is as follows:

- Install ansible on a control machine. This can be a laptop or a server that has access to the resources you want to manage.
- Define the tasks you want to automate in a playbook. A playbook is a YAML file that specifies the tasks you want to run and the order in which you want to run them
- Specify the hosts or groups of hosts that you want to execute the playbook on in an inventory file. An inventory file is a simple text file that lists the hostnames or IP addresses of the systems you want to manage.
- Run the playbook using the ansible-playbook command, specifying the playbook file and the inventory file as arguments.
- Ansible will connect to the hosts specified in the inventory file and execute the tasks defined in the playbook.

Ansible is a powerful tool that can be used to automate a wide range of tasks, including installing software, configuring systems, and deploying applications. It is designed to be easy to use and requires minimal setup, making it a popular choice for teams looking to automate their infrastructure management.
What is architecture of Ansible?
The architecture of Ansible is based on a simple, yet powerful, client-server

model. The basic components of the architecture include the control machine, managed nodes, and the Ansible automation engine.

- Control Machine: This is the machine where Ansible is installed and where playbooks are executed from. It communicates with managed nodes to execute tasks. This can be your local machine or a dedicated machine for Ansible.
- Managed Nodes: These are the machines that Ansible manages, also known as “remote hosts” or “target hosts”. The control machine communicates with managed nodes via SSH (default) or other transport protocols like winrm or telnet.
- Ansible Engine: This is the component that actually executes the automation tasks defined in playbooks and communicates with managed nodes.
- Inventory : The inventory is a list of managed nodes where the playbooks will be executed. Ansible uses this list to target the tasks of the playbooks to the right set of machines. The inventory can be in a simple text file, or you can use dynamic inventory that pulls data from external sources like cloud providers.
- Modules: These are the building blocks of Ansible playbooks. They are small programs that Ansible uses to perform specific tasks on managed nodes. Ansible includes a wide variety of modules for performing different types of tasks such as installing software, configuring systems, and deploying applications.
- Playbooks: Playbooks are YAML files that describe a set of tasks to be executed on managed nodes. They are written in a simple and human-readable language, and they allow you to specify the order in which tasks are executed, as well as the conditions under which they are executed.
- Plugins: Ansible provides a variety of plugins that can be used to extend its functionality. These include connection plugins, which are responsible for establishing connections to managed nodes, and callback plugins, which are used to format the output of Ansible tasks.

Overall, the Ansible architecture is designed to be simple and easy to use, while also being highly extensible and scalable. The client-server model allows Ansible to be used in a wide range of environments, and its simple, human-readable language makes it accessible to users of all skill levels.

What are the differences between Chef and Ansible?

Both Chef and Ansible are configuration management and automation tools that help you to manage and configure your infrastructure. However, there are some key differences between the two.

One of the main differences between Chef and Ansible is the way they approach configuration management. Chef uses a more “declarative” approach, in which you use a domain-specific language (DSL) to describe the desired state of your infrastructure. For example, you might use Chef to describe that a

certain package should be installed, and that a certain service should be running. Chef then takes care of ensuring that your infrastructure is in the desired state.
Ansible, on the other hand, takes a more “procedural” approach. Instead of describing the desired state of your infrastructure, you write “playbooks” (essentially scripts) that describe the steps that need to be taken to configure your infrastructure. For example, an Ansible playbook might contain a series of commands to install a package, start a service, and configure a firewall.
Another difference between Chef and Ansible is the complexity of the setup process. Chef requires more upfront work to get started. This is partly because it is a more comprehensive tool, and also because it requires you to set up a “Chef server” to manage your infrastructure. With Ansible, however, the setup is relatively straightforward, and it is often used as a simple automation tool for tasks like deploying applications.
Ansible also have a lower learning curve than Chef and also works on Agentless mode and thus doesn’t need any special setup on client machines, whereas Chef needs a agent to be installed on all the machines.
Finally, it’s worth noting that Chef is generally considered to be more powerful and feature-rich than Ansible, while Ansible is considered to be more simple and easy to use. Chef is better suited for large, complex infrastructures, while Ansible is better suited for small to medium-sized infrastructures and simple use cases.
You can choose one of the tool depend on your requirement and team’s expertise.

What is playbook in Ansible?

A “playbook” is a collection of instructions, written in the YAML language, that describes the desired state of your infrastructure and the steps needed to achieve that state. Playbooks are used to automate the deployment and configuration of your servers, applications, and other infrastructure components.
A playbook consists of one or more “plays,” which map a group of hosts to a certain set of tasks. Each play contains a list of tasks, and each task is a single instruction to be executed on one or more hosts. For example, a task might tell Ansible to install a package, start a service, or copy a file.
The tasks in a play are executed in the order they are defined, and Ansible provides a wide range of modules that can be used to perform a variety of actions, such as installing packages, editing files, and working with services.
Ansible playbooks can also include variables, which can be used to make your playbooks more flexible and reusable. You can define variables at the top of a playbook and then reference them in tasks and plays. This allows you to easily update the playbook to work with different environments, or to make changes to your infrastructure without modifying the playbook directly.
Additionally, playbook can include conditional statements, looping constructs, and other control flow statements, making it possible to build complex automation scripts.
Ansible playbooks are executed using the ansible-playbook command line tool, which takes the path to a playbook file as its argument. Once the playbook is

executed, Ansible will connect to the specified hosts and execute the tasks defined in the playbook.
Playbooks can be used for many things like configuring servers, deploying applications, and even running ad-hoc commands on remote hosts. Because of this flexibility, playbooks are an essential part of using Ansible effectively for automating your infrastructure.

Mention some list of sections that we mention in ansible playbooks?

- hosts: This section specifies the hosts or group of hosts that the playbook will be applied to. You can specify individual hostnames, IP addresses, or groups of hosts defined in your Ansible inventory.
- vars: This section is used to define variables that can be used throughout the playbook. Variables can be used to make playbooks more reusable and to easily update the playbook for different environments.
- vars_files: This section is used to include variables from external files. These files usually contain sensitive information and are usually not checked into version control.
- tasks: This is the most important section of a playbook. It contains a list of tasks that Ansible will execute on the specified hosts. Each task is defined using a module, and the list of available modules is quite extensive. This section can also include sub-sections like pre_tasks and post_tasks
- handlers: This section contains a list of tasks that are only executed when notified by a task. For example, a task might restart a service, and a handler might be notified to check that the service is running after it has been restarted.
- templates: This section used to include files from the Ansible control machine that are used to populate the content of a file on remote host.
- files: This section is used to include files from the Ansible control machine that are copied over to the remote host.
- meta: This section is used to include additional information about the playbook like dependencies and tags, which can be used to control the order in which playbooks are executed.

These are some of the main sections that you might include in an Ansible playbook, but depending on the complexity and requirement of the task, additional sections or sub-sections could also be added. It’s also important to mention that Ansible playbooks should be written in YAML format and it is case sensitive.

What is a DRY run in playbook?

“dry run” in Ansible refers to the ability to check the changes that a playbook will make to your infrastructure before actually executing the playbook. This can be useful in situations where you want to preview the changes that a playbook will make, or to check if a playbook has any errors before running it. When you run a playbook in “dry run” mode, Ansible will simulate the execution of the playbook and will report the changes that it would have made to your

infrastructure without actually making those changes. This can be useful to check for any potential mistakes, unintended consequences or missing requirements before actually running the playbook on the infrastructure.
You can run a playbook in “dry run” mode by using the --check or --dry-run option with the ansible-playbook command. For example, the following command would run a playbook called example.yml in dry-run mode:
ansible-playbook example.yml --check
When you run a playbook with dry run, Ansible will execute all tasks in the playbook, except for those that are marked as “always” or “never” to be executed during a dry run. Tasks that will run in dry run can be identified by the ‘changed’ attribute in the output, which will indicate if the task would change anything or not.
It is a good practice to always run a playbook in dry run mode before actually running it on your infrastructure, especially when making changes to production environments, to ensure that the playbook will have the intended effect and to identify any mistakes before they can cause any problems.

Why are we using loops concept in Ansible?

The ability to use loops in Ansible playbooks is an important feature that allows you to perform repetitive tasks and simplify your playbooks. Loops can be used to perform actions on multiple items, such as multiple servers, users, or files, with a single task or set of tasks.
Here are a few examples of why you might use loops in Ansible:

- Installing multiple packages: You can use a loop to install multiple packages on multiple servers. Instead of writing a separate task for each package, you can use a loop to install all of the packages in a single task.
- Creating multiple users: You can use a loop to create multiple users on a server. Instead of writing a separate task for each user, you can use a loop to create all of the users in a single task.
- Configuring multiple services: You can use a loop to configure multiple services on a server. Instead of writing a separate task for each service, you can use a loop to configure all of the services in a single task.
- Running commands on multiple hosts: You can use a loop to run commands on multiple hosts. This can be useful when you want to run the same command on many hosts at once.

Ansible provides several ways of working with loops:

- with_items : this loop is used to loop over a list of items, and for each iteration, the current item is made available to the task.
- with_dict : this loop is used to loop over a dictionary of key-value pairs, and for each iteration, the current key and value are made available to the task.
- with_fileglob : this loop is used to loop over a set of files, selected by a fileglob pattern.
- with_sequence : this loop is used to loop over a sequence of integers.

Using loops allows you to write less code and reduce duplication, making your playbooks more readable, maintainable and efficient. Furthermore, it allows you to manage multiple resources with a single Ansible task, making it easy to repeat the same action over multiple resources and also improve the performance of your playbooks.

Where do we use conditionals in Playbooks?

Conditionals in Ansible playbooks allow you to execute tasks or sets of tasks only when certain conditions are met. This allows you to create more dynamic and flexible playbooks that can adapt to different environments and situations. Here are a few examples of when you might use conditionals in Ansible playbooks:

- Installing a package only if it's not already installed: You can use a conditional statement to check if a package is already installed on a server, and only install it if it's not.
- Starting a service only if it's not already running: You can use a conditional statement to check if a service is already running on a server, and only start it if it's not.
- Creating a user only if it doesn't already exist: You can use a conditional statement to check if a user already exists on a server, and only create it if it doesn't.
- Taking action based on the result of a command: You can use a conditional statement to check the result of a command and take action based on the output.
- Running different sets of tasks based on the OS: You can use a conditional statement to check the OS version and take different actions depending on it.

Ansible provides several types of conditional statements that can be used in playbooks, such as:

- when : It is used to only execute a task when certain conditions are met
- changed_when, failed_when, success_when: It can be used to conditionally mark a task as changed, failed or success based on the output of the task.
- assert : It can be used to check for a certain condition to be true and fail the playbook execution if not

Conditionals in playbooks make them more dynamic and allow them to respond to different situations, depending on the state of the environment they are working on. It also allows you to handle errors, exceptional scenarios and make more intelligent decisions based on the results of certain actions. Conditionals can be used in conjunction with loops and variables, which can make your playbooks even more powerful and efficient.

What is Ansible vault?

Ansible Vault is a feature of Ansible that allows you to protect sensitive data, such as passwords, API keys, and encryption keys, by encrypting them using a password. The encrypted data is stored in files called "vault files," which are just regular text files that have been encrypted using AES256 encryption.

Ansible vault files can be used to protect sensitive data that is included in playbooks and roles, such as variables, files, and templates. For example, you might use Ansible Vault to encrypt a password that is used to connect to a database or a API key for a cloud service.
When you run a playbook that includes an Ansible vault file, you will be prompted for the password to decrypt the file. You can also use the --ask-vault-pass option to provide the password as a command-line argument, or use --vault-password-file to provide the password from file.
Ansible vault can also be used to encrypt whole files and even whole directories. This can be useful if you want to encrypt a whole configuration file or a set of configuration files.
Ansible vault is intended to protect sensitive data at rest, which means it is protecting the data when it's stored, not in transmission. So, it's important to take the necessary security measures when transmitting the data, such as using HTTPS or other encryption methods.
It's also worth noting that you should also protect the vault password and the files themselves, as they will be required to decrypt and access the data when necessary.
Ansible Vault is a powerful tool that allows you to secure sensitive data, making it safer to include in playbooks and roles. It's a good practice to use Ansible vault for any sensitive information you need to include in your playbooks, to protect it from unwanted access.

Write a sample playbook to install any package?

A simple Ansible playbook that installs the httpd package (Apache web server) and starts the service on RedHat/CentOS systems:

```
---
- name: Install and start Apache web server
  hosts: all
  become: yes
  tasks:
  - name: Install the httpd package
    package:
      name: httpd
      state: present
  - name: Start the httpd service
    service:
      name: httpd
      state: started
```

This playbook uses two tasks:

- The first task installs the httpd package using the package module and the state: present option, which will ensure that the package is installed if it is not already installed.
- The second task starts the httpd service using the service module and the state: started option, which ensures that the service is running after the playbook is executed.

The become: yes tells Ansible to run the tasks as the superuser (root) so that it has the necessary permissions to install the package and start the service.

The hosts: all line tells Ansible that this playbook should be run on all of the hosts listed in your inventory file. If you want to run the playbook on specific hosts or groups of hosts, you can replace all with the hostname or group name. You can run this playbook using the command ansible-playbook playbook.yml and it will install and start the Apache web server on all the servers specified in the hosts field.
Here's an example of a simple Ansible playbook that installs the httpd package (Apache web server) and starts the service on RedHat/CentOS systems by using variables instead of hard coding:

```
---
- name: Install and start Apache web server
  hosts: all
  become: yes
  vars:
    package_name: httpd
    service_name: httpd
    package_state: present
    service_state: started
  tasks:
  - name: Install the {{ package_name }} package
    package:
      name: "{{ package_name }}"
      state: "{{ package_state }}"
  - name: Start the {{ service_name }} service
    service:
      name: "{{ service_name }}"
      state: "{{ service_state }}"
```

This is just a simple example of how to install Apache web server on RedHat/CentOS, you can include additional tasks as per your requirement, such as configuring virtual host, firewall configuration, etc.
Infrastructure As Code
Dev Ops
Ansible

==========

Networking tasks in production using Ansible
In this blog, we will go through Networking tasks in production environment like monitoring, troubleshooting, debugging and many more.
As
a devops engineer or system administrator working in a production environment, you may need to perform a wide range of tasks related to networking. Some examples of these tasks might include monitoring network performance, troubleshooting

issues that arise on the network, configuring network devices, managing network security, and automating routine tasks.

Using ansible, you can automate many of these tasks to improve efficiency and reduce the risk of errors. Ansible is a powerful tool that allows you to create playbooks that automate tasks by executing commands and making configuration changes on your devices and servers. With its wide range of modules and tools, ansible can help you manage and maintain your network more effectively and make it easier to respond to changing needs and requirements.

❓

There are many common tasks that a DevOps engineer or system administrator may need to perform in production environments

related to networking.

- Monitoring network performance: Devops engineers and system administrators may need to monitor the performance of the network to ensure that it is functioning properly and meeting the needs of users. This might involve using tools like ping and traceroute to test connectivity, or monitoring performance metrics like bandwidth and latency.
- Troubleshooting network issues: When problems arise on the network, devops engineers and system administrators may need to troubleshoot the issue and identify the cause. This might involve reviewing log files, examining the configuration of network devices, or running diagnostic tests.
- Configuring network devices: Devops engineers and system administrators may need to configure network devices such as routers, switches, and firewalls to meet the needs of the organization. This might involve setting up VLANs, configuring firewall rules, or enabling features like Quality of Service (QoS).
- Managing network security: Devops engineers and system administrators may need to implement and maintain security measures to protect the network from external threats. This might involve installing security patches, configuring firewall rules, or implementing security protocols like IPSec.

Here are more detailed explanations of each of the common networking tasks that a DevOps engineer or system administrator might need to perform in a production environment, along with example code using ansible:

- Monitoring network performance:

Using ansible, you can automate the process of monitoring network performance by creating playbooks that run diagnostic tests and gather performance data. For example, you could use the ansible command module to run a ping test to a specific device on the network:

```
- name: Test connectivity to device
  command: ping -c 4 {{ device_ip }}
```

You can also use the ansible sar module to gather performance data from servers on the network. For example, you could use a playbook like this to gather CPU usage data:

```
- name: Gather CPU usage data
  sar:
    options: "-u"
```

2. Troubleshooting network issues:

Using ansible, you can automate the process of troubleshooting network issues by creating playbooks that gather information about the network and perform diagnostics. For example, you

could use the ansible command module to run a traceroute test to identify the path that packets are taking through the network:

```
- name: Identify packet path
  command: traceroute {{ destination_ip }}
```

You can also use the ansible ios_command module to run commands on Cisco devices and gather information about their configuration and status. For example, you could use a playbook like this to check the status of an interface on a Cisco device:

```
- name: Check interface status
  ios_command:
    commands:
      - show interface {{ interface }} status
```

3. Configuring network devices:

Using ansible, you can automate the process of configuring network devices by creating playbooks that apply changes to the configuration of the devices. For example, you could use the ansible ios_config module to configure a Cisco device:

```
- name: Configure Cisco device
  ios_config:
    lines:
      - hostname: {{ hostname }}
      - interface GigabitEthernet1/0/1
        ip address {{ ip_address }} 255.255.255.0
```

You can also use the ansible eos_config module to configure Arista EOS devices:

```
- name: Configure Arista device
  eos_config:
    lines:
      - hostname {{ hostname }}
      - interface Ethernet1
        no switchport
        ip address {{ ip_address }}/24
```

4. Managing network security:

Using ansible, you can automate the process of managing network security by creating playbooks that apply security measures to your devices. For example, you could use the ansible ios_command module to install security patches on Cisco devices:

```
- name: Install security patch
  ios_command:
    commands:
      - install activate source tftp://{{ patch_server }}/{{ patch_name }}
```

You can also use the ansible iptables module to configure firewall rules on Linux servers:

```
- name: Configure firewall rules
```

```
  iptables:
    chain: INPUT
    rule: "-p tcp --dport 22 -j ACCEPT"
    state: present
```

You can use the ansible openssh_keypair module to manage SSH keys, which can help to secure access to your servers:

```
- name: Generate SSH keypair
  openssh_keypair:
    path: /root/.ssh/id_rsa
    state: present
```

Overall, these are just a few examples of the types of tasks that devops engineers and system administrators might need to perform in production environments related to networking, and how they can use ansible to automate these tasks.

==========

Ansible:

==========

==========**********==========

Learning Ansible [Day 03 - B]

Let's learn about Ansible Playbook - Loops, Condition and Ansible Vault🚀

Condition:

In a playbook, you may want to execute different tasks or have other goals, depending on the value of a fact (data about the remote system), a variable, or the result of a previous task. You may want the value of some variables to depend on the value of other variables. Or you can create additional groups of hosts based on whether the hosts match other criteria. You can do all of these things with conditionals.

For Example:

```
--- # My Condition Playbook
- hosts: demo
  user: ansible
```

```
  become: yes
  connection: ssh
  gather_facts: yes

  tasks:
  - name: Install httpd server on Debian family
    command: apt-get -y install httpd
    when: ansible_os_family == "Debian"

  - name: Install httpd server on RedHat family
    command: yum -y install httpd
    when: ansible_os_family == "RedHat"
```

Loop:

Ansible loop is used to repeat any task or a part of code multiple times in an Ansible playbook. It includes the creation of multiple users using the user module, installing multiple packages using the apt or yum module or changing permissions on several files or folders using the file module.

For Example:

```
--- # My Loop Playbook
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  gather_facts: yes

  tasks:
  - name: add a list of users
    user: name="{{item}}" state=present
    with_items:
     - Node1
     - Node2
     - Node3
     - Node4
```

Ansible Vault:

Ansible Vault is an Ansible feature that helps encrypt confidential information without compromising security.

Ansible Vault Commands:

□ Create an encrypted file:
The ansible-vault create command is used to create the encrypted file.
➡ ansible-vault create vault.yml

□ Editing the encrypted file
If the file is encrypted and changes are required, use the edit command.
➡ ansible-vault edit secure.yml

□ Decrypting a file
The ansible-vault decrypt command is used to decrypt the encrypted file.
➡ ansible-vault decrypt secure.yml

□ Decrypt a running playbook
To decrypt the playbook while it is running, you usually ask for its password.
➡ ansible-playbook --ask-vault-pass email.yml

□ Reset the file password
➡ Use the ansible-vault rekey command to reset the encrypted file password.

Learning Reference:-

☑ https://lnkd.in/gb6gghQz tutorial

==========

Learning Ansible [Day 04]

Let's learn about Ansible Role🚀

Roles let you automatically load related vars, files, tasks, handlers, and other Ansible artefacts based on a known file structure. After you group your content into roles, you can easily reuse them and share them with other users.

Creating a Role:
The directory structure for roles is essential to creating a new role, such as:

Role Directory Structure:

⬜ By default Ansible will look in each directory within a role for a main.yml file for relevant content (also main.yaml and main):

⬜ tasks/main.yml - the main list of tasks that the role executes.

⬜ handlers/main.yml - handlers, which may be used within or outside this role.

⬜ library/my_module.py - modules, which may be used within this role

(see Embedding modules and plugins in roles for more information).

- defaults/main.yml - default variables for the role (see Using Variables for more information).
These variables have the lowest priority of any variables available and can be easily overridden by any other variable, including inventory variables.

- vars/main.yml - other variables for the role (see Using Variables for more information).

- files/main.yml - files that the role deploys.

- templates/main.yml - templates that the role deploys.

- meta/main.yml - metadata for the role, including role dependencies and optional Galaxy metadata such as platforms supported.

For example:

let us stick to the default structure of the roles. Each role is a directory tree in itself. So the role name is the directory name within the /roles directory.

➡ [root@ansible-server]$ mkdir -p playbook/roles/webserver/task
➡ [root@ansible-server]$ tree

➡ playbook
|
---> roles
|
---> webserver
|
---> tasks

➡ [root@ansible-server]$ cd playbook
➡ [root@ansible-server playbook]$ tree

➡ .
|
---> roles
|
---> webserver
|
---> tasks

➡ [root@ansible-server playbook]$ touch role/webserver/task/main.yml
➡ main.yml ---> - name: install git server on RedHat
yum: pkg=git state=latest

➡ [root@ansible-server playbook]$ vi master.yml
➡ [root@ansible-server playbook]$ ls
master.yml roles

➡--- # My Role Playbook
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  gather_facts: yes

roles:
- webserver

Learning Reference:-

☑ Saidemy Online Training Technical Guftgu Praveen Singampalli
☑ https://lnkd.in/d7Yei6KS

==========
https://github.com/sadebare/AWS_DEVOPS_PROJECTS/tree/main/PROJECT_9