5 hours

1. Overview of Configuration Automation

- Configuration automation refers to the process of automating the configuration and management of software systems.
- It involves using tools and techniques to automate the deployment, configuration, and maintenance of software applications, infrastructure, and environments.
- Ansible is one such popular configuration automation tool that simplifies the process of managing and automating configuration tasks.
- Ansible is an open-source automation tool that allows you to automate configuration management, application deployment, and orchestration tasks.
- It follows a declarative approach, where you define the desired state of the system, and Ansible takes care of bringing the system to that state.
- Ansible uses a simple YAML-based syntax called Ansible Playbooks to define the automation tasks.

Some key features and benefits of using Ansible for configuration automation are:

A. Agentless: Ansible does not require any agent to be installed on the target systems. It uses SSH or WinRM protocols to connect to the remote systems and perform the configuration tasks, making it easy to manage a large number of systems.

B. Infrastructure as Code: Ansible uses a declarative language to define infrastructure configurations as code. This allows you to treat infrastructure configurations as version-controlled code, enabling reproducibility, collaboration, and scalability.

C. Idempotent: Ansible ensures idempotency, meaning that running the same playbook multiple times will have the same result. It checks the current state of the system and only applies the necessary changes to bring it to the desired state.

D. Extensible: Ansible provides a rich set of modules that can be used to perform various configuration tasks. It also allows you to extend its functionality by writing custom modules or using third-party modules.

E. Orchestration: Ansible allows you to define complex workflows and orchestrate the execution of tasks across multiple systems. This enables you to automate complex configuration scenarios and ensure consistency across your infrastructure.

F. Easy to learn and use: Ansible has a simple and easy-to-understand syntax, making it accessible to both beginners and experienced developers. It has excellent documentation and a large community of users, providing extensive resources and support.

- By using Ansible for configuration automation, you can streamline the process of deploying and

managing software systems, reduce manual errors, and ensure consistency and repeatability in your infrastructure configurations.

2. Introduction to Ansible

- Ansible is an open-source automation tool that simplifies the process of automating IT tasks such as configuration management, application deployment, and orchestration.
- It provides a simple and powerful automation framework that allows you to manage and configure systems, deploy applications, and orchestrate complex workflows.

Key features of Ansible include:

A. Agentless Architecture: Ansible does not require any agents or additional software to be installed on the target systems. It communicates with remote systems using SSH or WinRM protocols, making it lightweight and easy to manage.

B. Declarative Language: Ansible uses a simple YAML-based language called Ansible Playbooks to define automation tasks. Playbooks describe the desired state of the system, and Ansible takes care of making the necessary changes to bring the system to that state.

C. Idempotent Operations: Ansible ensures that running the same playbook multiple times has the same result.

It checks the current state of the system and only applies the necessary changes to achieve the desired state. This allows for consistent and predictable results.

D. Extensible and Modular: Ansible provides a large number of pre-built modules that can be used to perform a wide range of tasks. Additionally, you can create custom modules to extend Ansible's functionality and meet specific automation requirements.

E. Infrastructure as Code: Ansible treats infrastructure configurations as code. Playbooks can be version-controlled and shared, enabling collaboration, reproducibility, and scalability. Infrastructure changes can be reviewed, tested, and deployed in a controlled manner.

F. Easy to Learn and Use: Ansible has a relatively low learning curve compared to other automation tools. Its syntax is human-readable and intuitive, making it accessible to both developers and system administrators. Ansible also provides comprehensive documentation and a large community that actively contributes to its development and support.

- Ansible is widely used in DevOps practices to automate repetitive and complex tasks, improve operational efficiency, and ensure consistency across infrastructure.

- It integrates well with popular cloud platforms, container technologies, and configuration management systems, making it a versatile tool for managing both on-premises and cloud-based environments.

3. Ansible Architecture

- The architecture of Ansible is designed to be simple, efficient, and scalable.
- It consists of the following components:

A. Ansible Control Node: The control node is where Ansible is installed and from where automation tasks are executed. It can be a physical server, a virtual machine, or even a developer's workstation. The control node is responsible for managing inventory, playbooks, and executing tasks on managed hosts.

B. Managed Hosts: Managed hosts are the target systems or devices that are controlled and managed by Ansible. These hosts can be physical servers, virtual machines, network devices, or any system accessible over SSH or WinRM. Ansible communicates with managed hosts using SSH or WinRM connections to execute commands, copy files, and perform various automation tasks.

C. Inventory: The inventory is a list of managed hosts that Ansible can manage. It can be defined as a static file or a dynamic inventory script that generates the list of hosts dynamically based on certain criteria. The

inventory file contains information about the hosts, such as IP addresses, hostnames, connection details, and host groups.

D. Playbooks: Playbooks are YAML files that define the desired state of the system and the tasks to be executed on the managed hosts. Playbooks consist of a series of plays, which are collections of tasks. Each task in a playbook describes an action to be taken on a specific host or a group of hosts. Playbooks can be used to configure systems, deploy applications, and orchestrate complex workflows.

E. Modules: Ansible modules are small pieces of code that perform specific automation tasks on managed hosts. Modules are executed on the managed hosts and report back to the control node. Ansible provides a large number of built-in modules for common tasks such as package installation, file manipulation, service management, and more. Additionally, custom modules can be created to extend Ansible's functionality.

F. Ansible Vault: Ansible Vault is a feature that allows you to encrypt sensitive data such as passwords, private keys, and other credentials used in playbooks. It ensures that sensitive information is securely stored and only accessible to authorized users.

- Ansible follows a push-based model, where the control node pushes configurations and tasks to the managed hosts.

- It does not require any agents to be installed on the managed hosts, making it lightweight and easy to manage.
- The Ansible architecture promotes simplicity, scalability, and flexibility, making it a popular choice for automation in DevOps practices.

4. Components of Ansible

- Ansible is composed of several key components that work together to enable automation and configuration management.

The main components of Ansible are:

A. Ansible Control Node: The control node is where Ansible is installed and from where automation tasks are managed and executed. It serves as the central management point for Ansible and is responsible for orchestrating the automation processes.

B. Inventory: The inventory is a file or collection of files that define the managed hosts or systems that Ansible will control. It contains information such as hostnames or IP addresses, connection details, and host groups. The inventory can be static or dynamic, allowing for flexibility in managing hosts.

C. Playbooks: Playbooks are written in YAML format and define the desired state of the systems and the tasks to be executed on the managed hosts. Playbooks consist of one or more plays, each containing a set of

tasks. Playbooks can be used to perform various automation tasks, such as configuration management, application deployment, and infrastructure provisioning.

D. Modules: Modules are small units of code that Ansible uses to perform specific tasks on managed hosts. Ansible ships with a large number of built-in modules that cover a wide range of operations, including file manipulation, package installation, service management, and more. Modules can be executed individually or as part of a playbook to achieve desired configurations.

E. Roles: Roles are a way to organize and structure playbooks in a reusable manner. A role encapsulates a specific functionality or set of tasks and can be shared across multiple playbooks. Roles provide a modular approach to organizing and managing automation code, making it easier to reuse and maintain.

F. Templates: Templates are files that contain a mix of static text and dynamic variables. Ansible uses template files to generate configuration files or other text-based files on managed hosts. Templates allow for parameterization and customization of configurations based on variables defined in playbooks or inventory.

G. Handlers: Handlers are tasks that are triggered by specific events or conditions in a playbook. They are typically used to restart services or perform other

actions that need to be triggered only when certain changes occur. Handlers are defined in playbooks and executed at the end of a play or when explicitly notified.

H. Ansible Vault: Ansible Vault is a feature that provides encryption and secure storage for sensitive data such as passwords, encryption keys, and other credentials. Vault allows you to encrypt sensitive information within playbooks or variables files, ensuring that it is protected and accessible only to authorized users.

- These components work together to provide a comprehensive automation framework that simplifies the management and configuration of systems and infrastructure.
- Ansible's modular and declarative approach allows for easy scalability and flexibility in handling complex automation tasks.

5. Installation and Configuration of Ansible

A. To install and configure Ansible, you can follow these general steps:

B. Choose a Control Node: Select a machine that will serve as the Ansible control node. This can be a physical or virtual machine running a Linux distribution.

C. Install Ansible: On the control node, install Ansible using the package manager of your Linux distribution.

For example, on Ubuntu, you can use the following command:

```
sudo apt-get install ansible
```

D. Configure SSH: Ansible uses SSH to connect to and manage remote hosts. Ensure that SSH is properly configured on both the control node and the managed hosts. This typically involves generating SSH keys and distributing the public key to the managed hosts.

E. Create an Inventory: Create an inventory file that lists the IP addresses or hostnames of the managed hosts. The inventory file can be stored in /etc/ansible/hosts or in a custom location specified using the -i flag when running Ansible commands.

F. Configure SSH Connection: In the inventory file, specify the SSH connection details for the managed hosts, such as the SSH username and private key file location. You can also define host groups to organize the hosts.

G. Test Connectivity: Test the connectivity between the control node and the managed hosts using the ansible command with the ping module. For example:

```
ansible all -m ping
```

H. Create Playbooks: Create YAML-based playbooks that define the desired configurations and tasks to be executed on the managed hosts. Playbooks consist of plays, which contain tasks and other elements.

I. Run Playbooks: Use the ansible-playbook command to run the playbooks and apply the configurations to the managed hosts. For example:

```
myplaybook
```

J. Verify Results: After running the playbooks, verify that the desired configurations have been applied to the managed hosts. Use Ansible modules and commands to check the state of the systems.

K. Additional Configuration: Depending on your specific requirements, you may need to configure additional settings in Ansible, such as variables, roles, templates, and handlers.

- These steps provide a basic overview of the installation and configuration process for Ansible.
- You may need to refer to the official Ansible documentation or consult specific guides for your operating system to ensure accurate installation and configuration.

6. Ansible ad-hoc commands

Ansible ad-hoc commands are one-line commands that allow you to perform quick tasks or execute simple modules on remote hosts without the need for writing a playbook. Here are some examples of commonly used ad-hoc commands in Ansible:

A. Ping all hosts:

```
ansible all -m ping
```

B. Execute a shell command on all hosts:

```
ansible all -a "command"
```

C. Gather facts from all hosts:

```
ansible all -m setup
```

D. Install a package on all hosts:

```
ansible all -m apt -a "name=<package_name> state=present"
```

E. Restart a service on all hosts:

```
ansible all -m service -a "name=<service_name> state=restarted"
```

F. Copy a file to all hosts:

```
ansible all -m copy -a "src=<local_file_path>
dest=<remote_file_path>"
```

G. Create a directory on all hosts:

```
ansible all -m file -a "path=<directory_path> state=directory"
```

H. Set up a user on all hosts:

```
ansible all -m user -a "name=<username> password=<password>
state=present"
```

I. Execute a playbook on specific hosts:

```
ansible-playbook playbook.yml --limit=<host_pattern>
```

J. Check available disk space on all hosts:

```
ansible all -m shell -a "df -h"
```

- These are just a few examples of the ad-hoc commands you can use in Ansible.
- Ad-hoc commands are particularly useful for performing quick tasks or troubleshooting specific issues on remote hosts.

7. Ansible Playbooks

- Ansible playbooks are YAML files that define a set of tasks, configurations, and policies to be applied on remote hosts.
- Playbooks are the heart of Ansible and provide a declarative approach to automate complex tasks and system configurations.

Here are some key points about Ansible playbooks:

A. Structure: Playbooks are written in YAML format, which makes them easy to read, write, and understand. A playbook consists of a list of plays, where each play defines a set of tasks to be executed on a group of hosts.

B. Hosts and Groups: Playbooks can target specific hosts or groups of hosts based on their inventory. The inventory file specifies the hosts and their associated variables that can be used in the playbook.

C. Tasks: Each play in a playbook consists of one or more tasks. A task is a unit of work to be performed on the remote host, such as installing packages, configuring services, or executing commands.

D. Modules: Tasks in playbooks use modules to perform specific actions on the remote hosts. Ansible provides a wide range of built-in modules that cover various system administration tasks, such as file

management, package installation, service management, and more.

E. Variables: Playbooks can define variables to store data that can be used across tasks or playbooks. Variables can be defined at different levels, including playbook level, group level, and host level, allowing for flexible and dynamic configurations.

F. Conditionals and Loops: Playbooks support conditional statements and loops, which allow you to control the flow of execution and perform repetitive tasks based on certain conditions.

G. Handlers: Handlers are special tasks in playbooks that are triggered by specific events. They are typically used to restart services or perform other actions only when necessary.

H. Roles: Playbooks can be organized into roles, which are self-contained units of configuration that can be reused across multiple playbooks. Roles provide a modular and scalable way to organize and share Ansible code.

I. Idempotence: Ansible playbooks are idempotent, meaning that running the same playbook multiple times should result in the same desired state, regardless of the initial state of the hosts. This allows for safe and repeatable execution.

J. Playbook Execution: Playbooks are executed using the ansible-playbook command, specifying the playbook file to run. Ansible manages the execution of tasks, communicates with the remote hosts, and provides feedback on the execution status.

- Overall, Ansible playbooks provide a powerful and flexible way to automate infrastructure management, application deployments, and other IT operations tasks.
- They enable you to define and maintain desired configurations in a declarative manner, simplifying the management of complex systems and improving overall efficiency.

8. Ansible Variables

- Ansible variables allow you to store and reuse values in your playbooks, making them dynamic and flexible.

Here are some key points about Ansible variables:

A. Variable Types: Ansible supports different types of variables, including global variables, playbook variables, role variables, group variables, and host variables. Each variable type has a different scope and precedence, allowing you to define variables at different levels and override them as needed.

B. Variable Declaration: Variables can be declared directly in playbooks or stored in separate variable files. Playbook variables are defined using the vars

keyword, while variable files can be in YAML or JSON format and are typically stored in the vars/ directory.

C. Variable Usage: Variables can be used in various places within a playbook, including task definitions, module parameters, conditionals, and templates. To reference a variable, you can use the {{ variable_name }} syntax.

D. Variable Precedence: Ansible follows a specific order when resolving variable values. Variables defined at a higher precedence level, such as host variables, override variables defined at a lower level, such as playbook variables.

E. Variable Substitution: Ansible allows you to perform variable substitution within strings using the {{ variable_name }} syntax. This enables you to dynamically generate configuration files, command strings, and other dynamic content based on the values of variables.

F. Variable Defaults: You can set default values for variables using the default filter. This ensures that a variable has a value even if it's not defined or passed from other sources.

G. Variable Expansion: Ansible provides various filters and functions that allow you to manipulate variable values. For example, you can use filters to transform strings, perform mathematical operations, manipulate lists, and more.

H. External Variable Sources: Ansible allows you to source variables from external files or systems. Common examples include using YAML or JSON files, environment variables, or even querying external systems through modules like lookup or ec2.

I. Variable Encryption: Ansible provides mechanisms to encrypt sensitive variable values, ensuring they are stored and transmitted securely. This is particularly useful for storing sensitive information like passwords or API keys.

J. Variable Precedence Rules: Understanding the variable precedence rules is important in Ansible. By knowing which variables take precedence over others, you can control how values are inherited and overridden, allowing for fine-grained control over your configurations.

- Overall, Ansible variables give you the flexibility to customize and tailor your playbooks to specific scenarios.
- They enable you to create reusable and dynamic configurations, making your automation tasks more robust and adaptable to different environments and requirements.

9. Ansible Handlers

- In Ansible, handlers are a way to define tasks that will be triggered only when specific events occur.
- They are typically used to manage services and perform actions that need to be executed as a result of a change in the system.

Here are some key points about Ansible handlers:

A. Definition: Handlers are defined in the playbook under the handlers section. Each handler is a task that specifies what action should be taken when triggered.

B. Triggering: Handlers are triggered by using the notify keyword within tasks. When a task notifies a handler, the handler is added to a list of tasks to be executed at the end of the play or when explicitly triggered.

C. Execution: Handlers are executed sequentially after all tasks have been completed. They are run once, even if multiple tasks notify the same handler. If a handler is notified multiple times, it will still be executed only once.

D. Trigger Conditions: Handlers can be triggered based on various conditions, such as when a configuration file changes, a service is restarted, or a package is installed. This allows for fine-grained control over when certain actions should be performed.

E. Idempotence: Handlers are designed to be idempotent, meaning that they can be run multiple times without causing issues. Ansible takes care of

ensuring that handlers are triggered only when necessary and that they won't cause unnecessary changes if run repeatedly.

F. Task Dependencies: Handlers can have dependencies on specific tasks, ensuring that they are executed only when those tasks have completed successfully. This can be useful when certain actions should be performed after a specific task has finished.

G. Separate Section: While handlers are defined within the playbook, they are typically placed in a separate section after the main tasks. This helps keep the playbook organized and makes it easier to locate and modify handlers when needed.

Example: Here's an example of how handlers can be used in a playbook:

```
tasks:
  - name: Install package
    yum:
      name: mypackage
      state: present
    notify: Restart service

handlers:
  - name: Restart service
    service:
      name: myservice
      state: restarted
```

- In this example, when the package mypackage is installed, it triggers the handler Restart service, which then restarts the myservice service.
- Handlers are a powerful feature in Ansible that allow for efficient and controlled management of actions triggered by specific events.
- They enhance the flexibility and automation capabilities of playbooks, making them a valuable tool in infrastructure management and configuration.

10. Ansible Role using Ansible Galaxy

- Ansible Roles are a way to organize and package Ansible tasks, variables, and files into reusable components.
- They provide a structured and modular approach to managing configuration and automation tasks in Ansible.
- Ansible Galaxy is a repository for sharing and discovering Ansible Roles created by the community.

Here's an overview of how to create and use an Ansible Role using Ansible Galaxy:

A. Role Structure: An Ansible Role has a predefined directory structure. It typically includes directories such as tasks, vars, files, and templates. These directories hold the respective Ansible files for tasks, variables, files, and templates used by the Role.

B. Creating a Role: To create a new Ansible Role, you can use the ansible-galaxy command-line tool. For example, to create a Role called "webserver", you can run the following command:

```
ansible-galaxy init webserver
```

This will create a new directory named "webserver" with the Role's directory structure and some initial files.

C. Writing Tasks: Inside the tasks directory of the Role, you can define the tasks that need to be executed. Tasks are written in YAML format and specify the actions to be performed on the target hosts.

D. Defining Variables: The vars directory is used to define variables specific to the Role. Variables can be used to customize the behavior of the Role and make it more flexible.

E. Including Files and Templates: The files and templates directories are used to include static files and template files, respectively. These files can be copied to the target hosts as part of the Role's execution.

F. Role Dependencies: Roles can have dependencies on other Roles. Dependencies are specified in a meta/main.yml file using the dependencies section. This allows Roles to be composed and reused in a modular way.

G. Sharing Roles on Ansible Galaxy: Once you have created a Role, you can share it on Ansible Galaxy for others to discover and use. You can create a metadata file called meta/main.yml with information about the Role, such as its name, description, author, and any required dependencies.

- Using Ansible Galaxy, you can easily search for and download Roles created by others, or publish your own Roles for others to benefit from.
- This helps promote reusability and collaboration in Ansible automation.
- Overall, Ansible Roles and Ansible Galaxy provide a convenient way to organize, share, and reuse Ansible configurations and automation tasks.
- They promote best practices in infrastructure as code and simplify the management of complex Ansible projects.

K. Practical Includes :

A. Installation and Configuration Ansible
B. Running Ansible ad-hoc commands.
C. Writing Ansible Playbooks to Configure Servers
D. Creating Ansible Roles

Practical: Ansible Configuration and Playbook Creation

a. Installation and Configuration of Ansible:

- Install Ansible on your control machine by following the official installation guide for your operating system.
- Configure the Ansible inventory file (/etc/ansible/hosts) to include the IP addresses or hostnames of your target servers.

b. Running Ansible ad-hoc commands:

- Use Ansible ad-hoc commands to perform simple tasks on your target servers.
- For example, run the following command to check the connectivity to your servers:

```
ansible all -m ping
```

c. Writing Ansible Playbooks to Configure Servers:

- Create a directory for your Ansible project and navigate to it.
- Create a YAML file named playbook.yml to define your playbook.
- Define hosts, tasks, and other necessary parameters in the playbook file.
- Write tasks to perform specific configurations on your target servers using Ansible modules.
- For example, you can write tasks to install packages, create users, configure services, etc.

d. Creating Ansible Roles:

- Inside your Ansible project directory, create a directory for your role. For example, roles/webserver.
- Within the role directory, create the necessary subdirectories like tasks, vars, files, and templates.
- Write tasks in the tasks/main.yml file to define the actions to be performed.
- Define variables in the vars/main.yml file to make your role more flexible.
- Place any required files or templates in their respective directories (files and templates).

e. Testing the Playbook and Role:

- Run the playbook using the ansible-playbook command and specify the path to your playbook file.
- Verify that the configurations specified in your playbook are applied to the target servers.
- Test the functionality of your role by running the playbook that includes the role.

Note: Ensure that you have appropriate SSH access and permissions to the target servers before running the practical.