



Ansible

Core Concepts

Ansible: An open-source automation tool used for configuration management, application deployment, task automation, and IT orchestration.

Playbooks: YAML files that define the tasks to be executed on managed nodes.

Inventory: A list of managed nodes (hosts) that Ansible manages, typically defined in a file.

Modules: Reusable, standalone scripts that Ansible uses to perform tasks on managed nodes.

Roles: A way to organize and reuse Ansible playbooks. Roles group together related tasks, variables, and handlers.

Common Commands

<code>ansible --version</code>	Check the Ansible version.
<code>ansible all -m ping -i inventory</code>	Ping all hosts in the inventory file.
<code>ansible-playbook playbook.yml -i inventory</code>	Run an Ansible playbook against the inventory.
<code>ansible-galaxy install role_name</code>	Install a role from Ansible Galaxy.
<code>ansible-vault encrypt file.yml</code>	Encrypt a file using Ansible Vault.

Example Playbook

```
---
- hosts: webservers
  become: true
  tasks:
    - name: Ensure Apache is installed
      apt:
        name: apache2
        state: present
    - name: Ensure Apache is running
      service:
        name: apache2
        state: started
```

Terraform

Key Concepts

Terraform: An infrastructure as code (IaC) tool that enables you to define and provision infrastructure using a declarative configuration language.

Providers: Plugins that allow Terraform to interact with different infrastructure platforms (e.g., AWS, Azure, GCP).

Resources: Components of your infrastructure, such as virtual machines, networks, and databases.

Modules: Reusable and composable units of Terraform configuration, similar to functions in programming.

State: Terraform uses a state file to track the current configuration of your infrastructure.

Common Commands

<code>terraform init</code>	Initialize a Terraform working directory.
<code>terraform plan</code>	Show changes required by the current configuration.
<code>terraform apply</code>	Apply the changes to the infrastructure.
<code>terraform destroy</code>	Destroy the infrastructure managed by Terraform.
<code>terraform show</code>	Inspect the current Terraform state.

Example Configuration

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = ">= 4.0"
    }
  }
}

provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "example" {
  ami           = "ami-0c55b9479a3c8c88c" # Example AMI
  instance_type = "t2.micro"

  tags = {
    Name = "ExampleInstance"
  }
}
```

Kubernetes Deployments

Core Components

Deployment: Manages the desired state of your application by ensuring the specified number of replicas are running.
Pod: The smallest deployable unit in Kubernetes, representing a single instance of a running process.
Service: An abstraction that defines a logical set of Pods and a policy by which to access them.
Namespace: A way to divide cluster resources between multiple users or teams.
Ingress: Manages external access to the services in a cluster, typically via HTTP.

Common kubectl Commands

<code>kubectl apply -f deployment.yaml</code>	Apply a configuration file to create or update resources.
<code>kubectl get deployments</code>	List all deployments in the current namespace.
<code>kubectl describe deployment <deployment-name></code>	Show detailed information about a deployment.
<code>kubectl scale deployment <deployment-name> - --replicas=<number></code>	Scale the number of replicas in a deployment.
<code>kubectl delete deployment <deployment-name></code>	Delete a deployment.

Example Deployment YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

Jenkins

Key Features

Jenkins: An open-source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery.
Pipelines: Jenkins pipelines allow you to define your entire build, test, and deployment process as code.
Plugins: Jenkins has a wide variety of plugins available to extend its functionality, such as integrations with source control systems, build tools, and deployment platforms.
Jobs: Automated tasks or series of tasks defined within Jenkins to perform specific actions such as building or deploying applications.
Nodes/Agents: Machines or containers that Jenkins uses to execute build jobs.

Pipeline Syntax

<code>pipeline { ... }</code>	Defines the overall pipeline structure.
<code>agent { ... }</code>	Specifies where the pipeline will execute (e.g., any node, a specific label).
<code>stages { ... }</code>	Defines the different stages of the pipeline.
<code>steps { ... }</code>	Contains the actual commands to execute in each stage.
<code>post { ... }</code>	Defines actions to be performed after the pipeline, regardless of the outcome.

Example Jenkinsfile

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        echo 'Building...'
      }
    }
    stage('Test') {
      steps {
        echo 'Testing...'
      }
    }
    stage('Deploy') {
      steps {
        echo 'Deploying...'
      }
    }
  }
}
```