DevOps Questions

AWS

**1. AWS services used in DevOps.**

• Source Control: AWS CodeCommit (instead of GitHub)

• Build & Test: AWS CodeBuild (instead of Jenkins + Maven + Selenium)

• Deploy: AWS CodeDeploy or AWS CodePipeline (for full CI/CD orchestration)

• Infrastructure: AWS CloudFormation or Terraform on AWS

• Container Service: Amazon ECS or EKS (instead of manual Docker on EC2)

• Monitoring: Amazon CloudWatch + AWS X-Ray (instead of Prometheus & Grafana)

**2. What Happens If You Forget the Public Key of an EC2 Instance?**

When you launch an EC2 instance, you connect to it using a **private key** (.pem file) that matches the **public key** stored on the instance. If you **lose the key pair** or forget the public key, **you won't be able to SSH into the instance** directly.

But don't worry — there are ways to recover access.

**🔒 First, What’s the Problem?**

* AWS doesn’t allow you to **see or retrieve the private key (.pem)** after the instance is launched.
* If you lose it or forget the public key:
  + You **cannot SSH** into the instance.
  + You **cannot replace the key pair directly** from the EC2 console.

**🛠️ Solutions to Regain Access**

**✅ Option 1: Use EC2 User Data to Replace the Key**

1. **Stop the EC2 instance.**
2. Detach the root volume (EBS).
3. Attach it to another instance as a secondary volume.
4. Log in to that other instance.
5. **Edit the ~/.ssh/authorized\_keys** file on the attached volume.
6. Add a **new public key** (one you have the private key for).
7. Detach the volume, reattach it to the original instance as the root volume.
8. Start the instance and SSH using the **new key pair**.

**✅ Option 2: Create an AMI & Launch New Instance**

1. Create an **image (AMI)** of the instance.
2. Launch a **new EC2** from that AMI.
3. In the launch settings, select a **new key pair**.
4. You'll now be able to SSH into the new instance.

**✅ Option 3: Use Systems Manager (SSM) – if set up**

If your instance:

* Has the **SSM agent installed**,
* Is in a **public subnet** or has **VPC endpoints** for SSM,
* And has the correct **IAM role attached**,

You can use **AWS Systems Manager > Session Manager** to connect **without SSH or a key pair**.

* Always **download and store your .pem file safely**.
* Use **EC2 Instance Connect** (browser-based SSH) if you're using Amazon Linux 2 or Ubuntu and haven’t disabled it.
* Use **SSM** wherever possible — it's secure and avoids key management headaches.

3. **What Are Security Groups in AWS?**

**Security Groups** are like **virtual firewalls** for your **EC2 instances** and other AWS resources. They control **inbound and outbound traffic** — that means they decide **who can connect to your instance and how**.

4. What is RDS?

**RDS** stands for **Relational Database Service** — it’s a **fully managed database service** by AWS that lets you run **relational databases** (like MySQL, PostgreSQL, SQL Server, etc.) **without having to manage the database software or infrastructure**.

**Amazon RDS** is a cloud service that lets you **easily set up, operate, and scale relational databases** — while AWS takes care of all the hard work like **backups, patching, and maintenance**.

5. How to connect EC2 instances to the local machine

To connect an **EC2 instance** to your **local machine**, you typically use **SSH** (Secure Shell) from your terminal or command prompt. Here's a simple step-by-step guide:

6. What is the difference between a public and private subnet?

A **public subnet** can connect to the internet.  
A **private subnet** **cannot** directly connect to the internet.

| **Feature** | **Public Subnet** | **Private Subnet** |
| --- | --- | --- |
| 🌍 **Internet Access** | Yes (via **Internet Gateway**) | No direct access (needs **NAT Gateway** to go out) |

7. S3 bucket globally rename it or not?

**No, you cannot rename an S3 bucket** in AWS.

 **S3 bucket names are globally unique**, meaning no two buckets in the world can have the same name.

 Once a bucket is created, its name is **locked** for that bucket and cannot be changed.

8. How to access the internet in a private subnet?

To **allow internet access from a private subnet**, you need to use a **NAT Gateway** or **NAT Instance**.

**9. Default Settings of Security Groups**

When you create a **new security group**:

* ✅ **Inbound (incoming)**:  
  ❌ **Nothing is allowed** by default.  
  ➤ No one can connect **to** your instance.
* ✅ **Outbound (outgoing)**:  
  ✅ **All traffic is allowed** by default.  
  ➤ Your instance **can connect out** to the internet.

🧠 So by default:

Your instance can go out (e.g., download updates), but nothing can come in unless you allow it.

**10. ELB (Elastic Load Balancer) – How It Works**

ELB **distributes traffic** to multiple servers so no single server gets overloaded.

👣 **Simple Flow:**

1. **User sends a request** (e.g., to your website).
2. **ELB receives it** and checks which server is healthy.
3. It **forwards the request** to one of the healthy EC2 instances.
4. If one instance goes down, ELB automatically **stops sending traffic to it**.

✅ Types:

* **Application Load Balancer (ALB)** – works best for web apps (Layer 7).
* **Network Load Balancer (NLB)** – for high performance (Layer 4).
* **Classic Load Balancer** – older, basic use.

**11. Auto Scaling Group (ASG) – Uses**

ASG automatically **adds or removes EC2 instances** based on traffic.

✅ Why it’s useful:

* 🚀 **Handles more traffic** by launching more instances.
* 💰 **Saves money** by shutting down extra instances when not needed.
* ❤️ **Keeps your app running** if an instance fails (self-healing).
* 📏 Scales **based on CPU, memory, traffic**, or custom metrics.

🧠 Example:

You set a rule:  
If CPU > 80%, add 1 more EC2.  
If CPU < 30%, remove 1 EC2.

**12. EBS – How to Mount a Volume in EC2**

EBS = Elastic Block Store = Storage volume attached to an EC2 instance.

👣 **How to Mount It (Linux)**

1. **Attach EBS** volume to your EC2 instance (from AWS Console).
2. **Connect to EC2** using SSH.
3. Run this to check if it’s detected:

bash

CopyEdit

lsblk

1. **Format** the volume (only if it’s new):

bash

CopyEdit

sudo mkfs -t ext4 /dev/xvdf

1. **Create a mount point**:

bash

CopyEdit

sudo mkdir /data

1. **Mount the volume**:

bash

CopyEdit

sudo mount /dev/xvdf /data

1. To make it permanent after reboot, add it to /etc/fstab.

🧠 Now your EC2 can use the new storage like a regular hard drive!

**13. Types of Storage in AWS**

| **Storage Type** | **Use Case** | **Example Services** |
| --- | --- | --- |
| **Block Storage** | Like a hard disk for EC2 | **EBS** |
| **Object Storage** | Store files, images, videos | **S3 (Simple Storage)** |
| **File Storage** | Shared file system across servers | **EFS (Elastic File System)** |
| **Instance Storage** | Temporary storage for EC2 | **EC2 Instance Store** |
| **Archive Storage** | Low-cost, long-term storage | **S3 Glacier** |

**14. CloudWatch for EC2**

**CloudWatch** helps you **monitor EC2 instances** and other AWS resources.

✅ What you can monitor:

* CPU usage
* Disk I/O
* Network traffic
* Memory (with CloudWatch Agent)

✅ Alerts:

* You can **set alarms** to notify you (e.g., if CPU > 80%).

🧠 Example:  
Get an email if your EC2 CPU is too high → take action!

**15. CloudFront**

**CloudFront** is AWS’s **Content Delivery Network (CDN)**.

✅ It delivers content (images, videos, websites) **faster to users** around the world by storing it in **edge locations**.

🧠 Example:  
User in India loads your site → instead of pulling data from the U.S., it comes from the **nearest server in India**.

**16. Edge Locations**

**Edge Locations** are **servers located around the world** used by **CloudFront**.

* They **cache content** for fast delivery.
* Used for **low latency** and **high performance**.

🧠 Think of them as "mini data centers" closer to your users.

**17. How Do You Monitor Amazon VPC?**

✅ Use these tools:

* **VPC Flow Logs** – See which traffic is coming in/out of VPC.
* **CloudWatch Logs** – Store and analyze flow log data.
* **CloudTrail** – Monitor changes to VPC configuration.
* **Amazon GuardDuty** – Detect threats inside VPC.

🧠 Example:  
Use Flow Logs + CloudWatch to detect blocked traffic or unauthorized access attempts.

**18. What is an Elastic IP?**

An **Elastic IP** is a **static public IP address** you can **attach to EC2**.

✅ Use it when:

* You want a fixed IP even if the instance restarts.
* You need to switch IPs between instances quickly.

⚠️ Note: AWS charges you **if it's not in use**.

**19. What is a Terraform State File?**

terraform.tfstate stores the **current state** of your infrastructure.

✅ Purpose:

* Keeps track of what resources Terraform manages.
* Helps Terraform **know what changed** since the last apply.

⚠️ Don’t edit it manually.  
✅ Use **remote state (e.g., S3)** for team environments.

**20. Security Groups vs NACLs**

| **Feature** | **Security Group** | **NACL (Network ACL)** |
| --- | --- | --- |
| Level | Instance level | Subnet level |
| Direction | Inbound & outbound | Inbound & outbound |
| Default behavior | Deny all inbound, allow all out | Allow all in & out |
| Rule type | **Allow only** | **Allow & Deny** |
| Stateful? | ✅ Yes (automatic return traffic) | ❌ No (return traffic needs a rule) |
| Use case | Control EC2 traffic | Extra layer of security on subnet |

🧠 Think of **security groups** like firewalls **per instance**, and **NACLs** as firewalls **per subnet**.

**21. VPC Components**

* **Subnets:** Segments of your VPC IP range (public/private).
* **Route Tables:** Rules for where network traffic goes.
* **Internet Gateway (IGW):** Connects VPC to the internet.
* **NAT Gateway:** Allows private subnet instances to access the internet.
* **Security Groups:** Virtual firewalls for instances.
* **Network ACLs:** Firewall rules at the subnet level.
* **Elastic IPs:** Static public IPs for instances.
* **Endpoints:** Private connections to AWS services without internet.

**22. How to Launch Instances in AWS EC2?**

1. Go to EC2 Dashboard.
2. Click **Launch Instance**.
3. Choose an **AMI** (OS template).
4. Select **Instance Type** (CPU, RAM).
5. Configure **network, storage, and security group**.
6. Add **key pair** for SSH access.
7. Review and **launch** the instance.

**23. What is EC2?**

**EC2 (Elastic Compute Cloud)** is AWS’s **virtual server** service.  
You can rent and run virtual machines (instances) in the cloud.

It provided a secure and resizable compute capacity in the cloud

**24. How to Manage Users and Permissions in AWS?**

Use **IAM (Identity and Access Management):**

* Create **users, groups, roles**.
* Assign **permissions** via **policies** (JSON rules).
* Control who can access what in your AWS account.

**25. What is Amazon EKS?**

**EKS (Elastic Kubernetes Service)** is a managed service for running **Kubernetes clusters** on AWS.

It simplifies deploying, managing, and scaling containerized applications.

**26. What is IAM?**

**IAM** is AWS’s **Identity and Access Management** system.

It helps you **securely control** access to AWS resources by creating users, groups, roles, and policies.

**27. What is AMI?**

**AMI (Amazon Machine Image)** is a **template** that contains an OS and software configuration to launch EC2 instances.

**28. How Can You Provide Permission in IAM?**

* Create or attach **IAM policies** (JSON docs) to:
  + **Users**
  + **Groups**
  + **Roles**

Policies define what **actions are allowed or denied** on which resources.

**29. What is ELB?**

**ELB (Elastic Load Balancer)** distributes incoming network traffic across multiple EC2 instances to ensure high availability and fault tolerance.

**30. What Are the Main Services in AWS?**

* **Compute:** EC2, Lambda, EKS
* **Storage:** S3, EBS, EFS, Glacier
* **Database:** RDS, DynamoDB, Aurora
* **Networking:** VPC, Route 53, ELB
* **Security:** IAM, KMS, Shield
* **Monitoring:** CloudWatch, CloudTrail
* **DevOps:** CodePipeline, CodeBuild, CodeDeploy

**31. Relational Database in AWS**

* AWS offers **RDS (Relational Database Service)**.
* Supports databases like **MySQL, PostgreSQL, Oracle, SQL Server, MariaDB, Aurora**.
* Managed service: handles backups, scaling, patching, and recovery for you.

**32. EC2 Instance Auto-Scaling**

* **Auto Scaling Group (ASG)** automatically adds or removes EC2 instances based on demand (like CPU or network traffic).
* Helps maintain performance and reduce cost.
* Ensures **high availability** by replacing unhealthy instances.

**33. Explain VPC and Components**

* **VPC (Virtual Private Cloud):** Your own isolated network in AWS.
* **Public Subnet:** Can access internet via **Internet Gateway (IGW)**.
* **Private Subnet:** No direct internet access; uses **NAT Gateway** to access internet.
* **NAT Gateway:** Allows private subnet instances to reach internet securely.
* **IGW (Internet Gateway):** Connects VPC to the internet.
* **Security Groups:** Firewall rules applied to instances controlling inbound/outbound traffic.

**34. EC2 and RDS Backups**

* **EC2 Backups:** Use **EBS snapshots** to back up instance storage.
* **RDS Backups:** Automated backups and manual snapshots of databases.
* Backups help restore data or launch new instances/databases from the saved state.

**35. What is S3?**

* **S3 (Simple Storage Service)** is object storage for files like images, videos, backups.
* Scalable, durable, and accessible from anywhere.
* Pay only for what you use.

**36. S3 Versioning**

* Keeps **multiple versions** of the same object.
* Allows recovery from accidental deletes or overwrites.
* You can enable or suspend versioning on a bucket.

**37. How to Access S3 and RDS from EC2**

* **S3:** Use AWS CLI, SDKs, or REST API (no need for public IP).
* **RDS:** Connect using the database endpoint and credentials.
* **Make sure security groups and network ACLs allow traffic** between EC2 and RDS.
* Optionally use **IAM roles** attached to EC2 for S3 access without storing credentials.

**38. What is EBS?**

* **EBS (Elastic Block Store)** provides persistent block storage volumes for EC2.
* Like a virtual hard disk.
* Can be attached/detached to instances.

**39. What is EFS?**

* **EFS (Elastic File System)** is a scalable file storage service.
* Can be shared across multiple EC2 instances.
* Works like a network file system (NFS).

**40. How to Configure VPC in Terraform**

Basic example snippet:

hcl

CopyEdit

resource "aws\_vpc" "my\_vpc" {

cidr\_block = "10.0.0.0/16"

}

resource "aws\_subnet" "public\_subnet" {

vpc\_id = aws\_vpc.my\_vpc.id

cidr\_block = "10.0.1.0/24"

map\_public\_ip\_on\_launch = true

}

resource "aws\_internet\_gateway" "igw" {

vpc\_id = aws\_vpc.my\_vpc.id

}

resource "aws\_route\_table" "public\_rt" {

vpc\_id = aws\_vpc.my\_vpc.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.igw.id

}

}

resource "aws\_route\_table\_association" "a" {

subnet\_id = aws\_subnet.public\_subnet.id

route\_table\_id = aws\_route\_table.public\_rt.id

}

**41. AWS Services**

* **EC2:** Virtual servers in the cloud.
* **RDS:** Managed relational databases.
* **EFS:** Shared file storage.
* **EBS:** Block storage volumes for EC2.
* **Load Balancer (ELB):** Distributes traffic across servers.
* **Internet Gateway (IGW):** Connects VPC to the internet.
* **NAT Gateway:** Allows private subnet resources to access the internet.

**42. T2 Micro**

* A small, low-cost EC2 instance type.
* Good for light workloads, testing, or small apps.
* Uses “burstable” CPU performance.

**43. Difference Between On-Prem and Cloud**

| **Aspect** | **On-Prem** | **Cloud** |
| --- | --- | --- |
| Ownership | You own/manage hardware | Provider manages hardware |
| Scalability | Limited by your hardware | Easy and instant scaling |
| Cost | High upfront costs | Pay-as-you-go |
| Maintenance | You handle everything | Provider handles infrastructure |
| Flexibility | Less flexible | Highly flexible |

**44. How to Migrate Apps and Databases from On-Prem to Cloud**

* Plan and assess workloads.
* Use tools like **AWS Migration Hub, Database Migration Service (DMS)**.
* Migrate data (e.g., databases) using DMS.
* Re-host or re-architect apps for cloud.
* Test and optimize.

**45. AWS ElasticSearch**

* Managed search and analytics service.
* Based on **Elasticsearch** open-source project.
* Used for log analytics, full-text search, monitoring.

**46. AWS IAM Roles**

* Temporary credentials assigned to AWS resources.
* Allow resources to access other AWS services securely.
* Example: EC2 instance role to access S3 without storing keys.

**47. EBS-backed vs. Instance-store Backed Instances**

| **Feature** | **EBS-backed** | **Instance-store backed** |
| --- | --- | --- |
| Root volume | EBS volume | Instance store (ephemeral) |
| Data persistence | Persistent across stops/reboots | Data lost if instance stops |
| Flexibility | Can stop/start instance | Cannot stop, only terminate |

**48. AWS EC2 Instance Types**

* **General Purpose:** Balanced CPU, memory (e.g., t3, m5).
* **Compute Optimized:** High CPU (e.g., c5).
* **Memory Optimized:** High RAM (e.g., r5).
* **Storage Optimized:** Fast local storage (e.g., i3).
* **GPU Instances:** For graphics and ML (e.g., p3).

**49. Cross-region S3**

* Replicates S3 bucket data to another AWS region.
* Improves disaster recovery and compliance.
* Uses **Cross-Region Replication (CRR)**.

**50. NAT Gateway Purpose**

* Allows instances in private subnets to access the internet for updates/downloads.
* Keeps instances private and unreachable from the internet.

**51. Lost PEM Key & Need SSH Access**

* You **cannot SSH directly** without the PEM key.
* Workaround:
  + Stop the instance (do **not terminate**).
  + Detach its root volume.
  + Attach the volume to another EC2 instance as a secondary volume.
  + Access the attached volume and add a new public key to ~/.ssh/authorized\_keys.
  + Detach and reattach the volume back to the original instance.
  + Start the instance and SSH using the new key.

**52. Connect to Private Subnet via SSH from Local Machine**

* Use a **bastion host** (jump server) in the public subnet.
* SSH first into the bastion host.
* From the bastion, SSH into the private subnet instance.

Example:

bash

CopyEdit

ssh -i bastion.pem ec2-user@bastion-public-ip

ssh -i private-instance.pem ec2-user@private-ip

Or use **SSH agent forwarding** or **Session Manager** for easier access.

**53. Connectivity Between Two VPCs in Private Subnets**

* Use **VPC Peering**: Connect two VPCs privately.
* Or use **VPN connection** or **Transit Gateway** for complex networks.
* Update route tables in both VPCs to allow traffic.

**54. Why Auto Scaling in AWS?**

* Automatically **adjusts number of instances** based on load.
* Improves **availability** and **performance**.
* Saves **costs** by shutting down unused instances.
* Provides **fault tolerance** by replacing unhealthy instances.

**55. What is IAM Server?**

* Actually, there is no "IAM server."
* **IAM (Identity and Access Management)** is a **service** in AWS to manage users, permissions, and roles.
* It controls who can do what inside your AWS account.

**56. What is AMI in a Server?**

* **AMI (Amazon Machine Image)** is a **template** for launching EC2 instances.
* Contains the OS, application server, and applications.
* You can create custom AMIs to launch identical servers easily.

Linux

**1. Linux Basic Commands**

* ls — List files and directories
* cd — Change directory
* pwd — Show current directory path
* mkdir — Create a new directory
* rm — Remove files or directories
* cp — Copy files or directories
* mv — Move or rename files or directories
* cat — Show file content
* touch — Create an empty file
* chmod — Change file permissions
* chown — Change file ownership
* ps — List running processes
* kill — Terminate a process

**2. What is Bash?**

* **Bash (Bourne Again Shell)** is a command-line shell and scripting language in Linux.
* It lets you interact with the system by typing commands.
* Also used to write shell scripts for automating tasks.

**3. Disk Management**

* df — Show disk space usage
* du — Show directory size
* fdisk — Partition disks
* mount — Attach a filesystem
* umount — Detach a filesystem
* lsblk — List block devices (disks)
* blkid — Show disk UUIDs and types

**4. Bash Scripting**

* Writing scripts using Bash language to automate commands.
* Example:

bash

CopyEdit

#!/bin/bash

echo "Hello, World!"

* Save as script.sh.

**5. How to Run Bash Scripts**

* Give execute permission:  
  chmod +x script.sh
* Run it:  
  ./script.sh
* Or run directly with Bash:  
  bash script.sh

**6. Linux Distributions**

* Different versions of Linux with different features.
* Popular distros:
  + Ubuntu
  + CentOS
  + Debian
  + Fedora
  + Red Hat Enterprise Linux (RHEL)

**7. Difference Between top and nice Commands**

* top: Shows real-time system processes and resource usage.
* nice: Runs a process with modified priority (CPU scheduling).

**8. Command to Check Server Load**

* uptime — Shows load averages and uptime.
* top — Shows load and running processes.
* w — Shows users and load averages.

**9. How to Rename a File in Linux?**

* Use mv command:

bash

CopyEdit

mv old\_filename new\_filename

**10. Basic Linux Commands**

(Repeating from #1 for easy reference)

* ls, cd, pwd, mkdir, rm, cp, mv, cat, chmod, ps, kill

**11. Linux Troubleshooting Commands**

* dmesg — View system boot and kernel messages
* journalctl — Check system logs
* top / htop — Monitor system processes and resource use
* ping — Test network connectivity
* traceroute — Trace network path to a host
* netstat / ss — Show network connections and ports
* lsof — List open files and processes
* strace — Trace system calls of a process

**12. Commands for Disk, Memory, and CPU Usage**

* Disk:
  + df -h — Disk free space in human-readable form
  + du -sh /path — Disk usage of directory
* Memory:
  + free -m — Memory usage in MB
  + vmstat — Virtual memory statistics
* CPU:
  + top — Real-time CPU usage
  + mpstat — CPU usage per processor

**13. Linux Patching**

* Updating system packages to fix bugs/security issues.
* On Debian/Ubuntu:

bash

CopyEdit

sudo apt update

sudo apt upgrade

* On CentOS/RHEL:

bash

CopyEdit

sudo yum update

* Can be automated using tools like **cron** or **Ansible**.

**14. How to Add a User to Your System?**

bash

CopyEdit

sudo adduser username

* Creates a new user and prompts to set password.
* Alternatively:

bash

CopyEdit

sudo useradd username

sudo passwd username

**15. Authentication Methods Used by SSH**

* Password-based authentication
* Public key (SSH key) authentication
* Two-factor authentication (optional)
* Host-based authentication (less common)

**16. Operating System-Related Questions**

* OS manages hardware and software resources.
* Linux is a popular open-source OS.
* Kernel is the core part managing hardware.
* Shell is the interface for user commands.

**17. About Linux**

* Linux is an open-source operating system.
* It is Unix-like and widely used on servers.
* Consists of kernel + utilities + shell.
* Distributions customize Linux for different needs.

**18. Write and Explain All Linux Commands You Know**

* This is broad; here are some grouped by purpose:

**File and Directory:** ls, cd, pwd, mkdir, rm, mv, cp  
**Process Management:** ps, top, kill, nice  
**Networking:** ping, netstat, ss, traceroute  
**Disk and Storage:** df, du, mount, umount, fdisk  
**User Management:** adduser, userdel, passwd  
**System Info:** uptime, uname, hostname  
**Search:** grep, find

**19. Linux Commands for Disk, Memory, CPU Usage, and grep**

* Disk: df -h, du -sh /path
* Memory: free -m, vmstat
* CPU: top, mpstat
* Search inside files:

bash

CopyEdit

grep "pattern" filename

**20. Change IP Address and Hostname on Linux (Command Line)**

* **Change IP address temporarily:**

bash

CopyEdit

sudo ip addr add 192.168.1.10/24 dev eth0

sudo ip addr del 192.168.1.5/24 dev eth0

* **Change hostname temporarily:**

bash

CopyEdit

sudo hostnamectl set-hostname new-hostname

* For permanent IP and hostname changes, edit config files like /etc/network/interfaces, /etc/sysconfig/network-scripts/ifcfg-eth0, and /etc/hostname depending on distro.

**21. Configuration of DNS on Linux**

To configure DNS (name servers):

* Edit the DNS file:

bash

CopyEdit

sudo nano /etc/resolv.conf

* Add DNS servers like:

nginx

CopyEdit

nameserver 8.8.8.8

nameserver 1.1.1.1

*Note: In some distros, this file is managed by NetworkManager or systemd.*

**22. Public and Private DNS**

* **Public DNS**: DNS servers accessible over the internet.  
  Example: Google DNS (8.8.8.8), Cloudflare (1.1.1.1)
* **Private DNS**: Used within private networks (like AWS VPC), only accessible inside that network.

**23. Command Executed When a User Logs In to Linux**

* On login, the system runs files like:
  + ~/.bash\_profile
  + ~/.profile
  + ~/.bashrc (for non-login shells)

You can add commands in these files to auto-run something at login.

**24. How to Check IP Address in Linux**

* Use:

bash

CopyEdit

ip a

or

bash

CopyEdit

ifconfig # Might need to install: sudo apt install net-tools

**25. How to Log in to an Ubuntu Machine**

* Using SSH:

bash

CopyEdit

ssh -i key.pem ubuntu@<public-ip>

Make sure port 22 is open in the security group.

**26. Steps to Launch an EC2 Instance (Ubuntu)**

1. Go to AWS EC2 console.
2. Click **Launch Instance**.
3. Choose **Ubuntu AMI**.
4. Choose instance type (e.g., t2.micro).
5. Configure network/subnet.
6. Add storage (default is fine).
7. Add tags (optional).
8. Configure security group (allow SSH - port 22).
9. Choose or create a key pair.
10. Launch the instance.

**27. Can We Connect to an EC2 Server Without a Key Pair?**

* **No, not directly with SSH**.
* Alternatives:
  + Use **EC2 Instance Connect (browser)**.
  + Use **SSM (AWS Systems Manager)**.
  + Or, create a key pair, add the public key manually to the instance via console or user-data.

**28. rm -i Command in Linux**

* The -i flag asks for confirmation before deleting:

bash

CopyEdit

rm -i filename

* Prevents accidental file deletion.

**29. How to Save and Exit a File in Vi Editor**

* Press Esc
* Type:
  + :w → Save
  + :q → Quit
  + :wq → Save and quit
  + :q! → Quit without saving

**30. Can You Name Some Commands in Linux?**

Sure! Here’s a quick list:

* File management: ls, cd, cp, mv, rm, touch, mkdir
* Networking: ping, curl, wget, netstat, ip
* System: top, uptime, ps, free, df, du
* User: adduser, passwd, whoami
* Search: grep, find, locate
* Archive: tar, zip, unzip

**31. SCP vs SSH**

| **Feature** | **SCP** | **SSH** |
| --- | --- | --- |
| Purpose | Securely copy files | Secure shell access to systems |
| Command Format | scp file user@host:/path | ssh user@host |
| Transfers | Files only | Commands, files, remote sessions |
| Port | Uses port 22 (same as SSH) | Uses port 22 |

Git

**1. What is the most effective approach to restructure commit history when Robert John has multiple commits for a single task?**

👉 Use **interactive rebase** to squash multiple commits into one:

bash

CopyEdit

git rebase -i HEAD~N

Replace N with the number of commits you want to modify. Then mark the extra commits as squash or s to combine them into a single commit with a clear message.

**2. What happens when you run the following command on the Git repository? git reset --soft HEAD^**

👉 This **removes the latest commit**, but **keeps the changes in the staging area**. You can edit and recommit if needed.

**3. Which Git command can configure the use of vimdiff as the default Git merge tool?**

👉 Run:

bash

CopyEdit

git config --global merge.tool vimdiff

**4. How does Git merge handle whitespace differences between versions?**

👉 Git offers merge options to **ignore whitespace**:

bash

CopyEdit

git merge -Xignore-space-change branch-name

or

bash

CopyEdit

git merge -Xignore-all-space branch-name

This helps avoid conflicts due to formatting-only changes.

**5. Difference between centralized vs. distributed version control systems (VCS)?**

| **Centralized VCS** | **Distributed VCS (e.g., Git)** |
| --- | --- |
| One central server | Each developer has full repo copy |
| Needs internet to commit | Can commit offline |
| Examples: SVN, CVS | Examples: Git, Mercurial |

**6. What is the difference between Git pull and Git fetch?**

| **Git Pull** | **Git Fetch** |
| --- | --- |
| Fetches + merges immediately | Only fetches updates from remote |
| May change your local code | Safe; local changes untouched |
| Shortcut for fetch + merge | Use when you want to review before merging |

**7. What is the difference between Git rebase and merge?**

| **Git Merge** | **Git Rebase** |
| --- | --- |
| Keeps the original branch structure | Rewrites commit history linearly |
| Creates a merge commit | Applies commits one by one |
| Good for shared/public branches | Good for cleaning up local history |

**8. What is the command used to create a branch?**

👉 To create a branch:

bash

CopyEdit

git branch branch-name

👉 To create and switch to it:

bash

CopyEdit

git checkout -b branch-name

**9. What is the difference between Git remote and Git clone?**

| **Git Remote** | **Git Clone** |
| --- | --- |
| Links your local repo to a remote | Creates a local copy of a remote repo |
| Used with git init | Used as the first step to work with GitHub |
| Example: git remote add origin <url> | Example: git clone <url> |

**10. What are common Git branching strategies?**

* **Main/Master + Feature branches**: Keep main clean, develop in branches.
* **Git Flow**: Uses develop, feature, release, hotfix, and main branches.
* **Trunk-Based Development**: Small, frequent commits to main.
* **Forking Workflow**: Each developer works in their forked repo and submits PRs (popular in open source).

**11. Types of Version Control Systems**

* **Local VCS**: Tracks changes on your local machine (e.g., RCS).
* **Centralized VCS**: One central server, all users pull/push from it (e.g., SVN).
* **Distributed VCS**: Everyone has a full copy of the repo (e.g., Git, Mercurial).

**12. Basic Git Commands**

* git init – Initialize a new repository
* git clone <url> – Copy a remote repository
* git status – Show the current status
* git add <file> – Stage file for commit
* git commit -m "msg" – Save changes
* git push – Upload changes to remote
* git pull – Download and merge changes
* git log – View commit history

**13. Command to Rename a Directory or File**

bash

CopyEdit

git mv old\_name new\_name

**14. How to Checkout Code from GitHub**

bash

CopyEdit

git clone https://github.com/username/repo.git

This downloads the code from GitHub to your local system.

**15. Difference Between Git Merge and Git Rebase**

| **Git Merge** | **Git Rebase** |
| --- | --- |
| Keeps commit history | Rewrites commit history |
| Creates a merge commit | Makes history linear |
| Good for collaboration | Good for cleaning local history |

**16. Difference Between Git Pull and Git Fetch**

| **Git Pull** | **Git Fetch** |
| --- | --- |
| Fetches + merges automatically | Only fetches updates |
| May cause merge conflicts | Safe, no code changes locally |

**17. What is a Merge Conflict in Git and How Do You Resolve It?**

* **Merge Conflict**: Happens when two branches change the same line in a file.
* **How to Resolve**:
  1. Git will mark the conflict area.
  2. Open the file, fix the conflict manually.
  3. After fixing:

bash

CopyEdit

git add conflicted\_file

git commit

**18. Command to Create a New Branch in Git**

bash

CopyEdit

git branch new-branch

Or to create and switch at once:

bash

CopyEdit

git checkout -b new-branch

**19. Command to Checkout from One Branch to Another in Git**

bash

CopyEdit

git checkout branch-name

**20. Git vs. GitHub Differences**

| **Git** | **GitHub** |
| --- | --- |
| A version control tool | A cloud platform for hosting Git repos |
| Works locally | Works online (browser or CLI) |
| Created by Linus Torvalds | Owned by Microsoft |
| Command-line tool | Web-based UI with extra tools |

**21. Git Commands**

Common Git commands:

* git init – Initialize a new repo
* git clone <url> – Copy a repo
* git status – Check current changes
* git add <file> – Stage changes
* git commit -m "message" – Save changes
* git push – Upload changes to remote
* git pull – Get changes from remote
* git branch – List branches
* git checkout <branch> – Switch branch
* git merge <branch> – Merge branches
* git rebase <branch> – Reapply commits on top
* git log – View commit history

**22. User Access Management in Git**

If using platforms like **GitHub**, **GitLab**, or **Bitbucket**:

* Use **Teams/Roles** to give permissions (read/write/admin).
* Manage through **Settings > Collaborators or Access Management**.
* Use **SSH keys or Personal Access Tokens** for secure access.

**23. Checkout Command in Git**

* Switches between branches or commits:

bash

CopyEdit

git checkout branch-name

* To create and switch to a new branch:

bash

CopyEdit

git checkout -b new-branch

**24. Purpose of git rebase Command**

* Rewrites your branch’s history to make it **linear and clean**.
* Used to apply your changes on top of another branch.
* Helpful to **avoid merge commits** and keep commit history simple.

Example:

bash

CopyEdit

git rebase main

**25. Difference Between git pull and git fetch**

| **git pull** | **git fetch** |
| --- | --- |
| Fetches and merges immediately | Only fetches, no merge |
| May change local files | Leaves your working directory untouched |
| Good when you want latest changes now | Good when you want to review changes first |

**26. What are Pull Request Strategies?**

* **Feature Branch Strategy** – Each feature in a new branch, reviewed via pull request.
* **Forking Workflow** – Used in open source; users fork and create pull requests.
* **Git Flow Strategy** – Uses develop, feature, release, and main branches.
* **Trunk-Based Development** – Frequent small pull requests to the main branch.

**27. What Are All Git Commands? What Did You Perform in Your Project?**

**Common Git commands:**

* git init, git clone, git status, git add, git commit, git push, git pull, git branch, git checkout, git merge, git rebase, git log, git stash, git reset

**In my project, I:**

* Cloned the repo from GitHub
* Created branches for features
* Committed and pushed changes
* Resolved merge conflicts
* Raised pull requests for code review
* Merged code into the main branch

**28. What is .gitignore File?**

* A file used to **exclude files/folders** from being tracked by Git.
* Example entries:

bash

CopyEdit

node\_modules/

\*.log

.env

**29. How Do You Configure Git in Jenkins?**

1. Install Git plugin in Jenkins.
2. In Jenkins:
   * Go to **Manage Jenkins > Global Tool Configuration**
   * Add Git installation path (auto-detected in most cases)
3. In your Jenkins job:
   * Under **Source Code Management**, choose **Git**
   * Enter repo URL and credentials
   * Set branch (e.g., \*/main)
4. Jenkins pulls the code during the build.

Docker

D is a containirazation tool for building, packaging , deploying and running the application as container.

**1. Commands of Docker**

* docker build – Build an image from a Dockerfile
* docker run – Run a container
* docker ps – List running containers
* docker stop – Stop a container
* docker rm – Remove a container
* docker rmi – Remove an image
* docker images – List images
* docker exec -it <container> bash – Run commands inside a running container
* docker logs <container> – View logs of a container
* docker volume create – Create a volume

**2. What is Docker Hub?**

👉 Docker Hub is a cloud-based registry where you can **store and share Docker images**.  
You can **pull public images** (like Ubuntu, Nginx) or **push your own** images to share with others.

**3. Dockerfile: How Will You Write It?**

Sample Dockerfile:

dockerfile

CopyEdit

FROM ubuntu

RUN apt update && apt install -y nginx

COPY . /app

CMD ["nginx", "-g", "daemon off;"]

This file:

* Uses Ubuntu image
* Installs Nginx
* Copies app files
* Runs Nginx

**4. Difference Between Docker Image and Container**

| **Docker Image** | **Docker Container** |
| --- | --- |
| Blueprint/template | Running instance of image |
| Read-only | Writable + isolated runtime |
| Built using Dockerfile | Created using docker run |

**5. Where to Use RUN and EXEC in Docker?**

* **RUN**: Used in Dockerfile during **image build** to install packages, run scripts, etc.
* **docker exec**: Used on **running containers** to run a command inside the container.

**6. Docker Volume Creation and Mounting**

Create volume:

bash

CopyEdit

docker volume create myvolume

Mount volume when running container:

bash

CopyEdit

docker run -v myvolume:/app ubuntu

**7. Docker Log**

View logs of a container:

bash

CopyEdit

docker logs <container\_name\_or\_id>

**8. Execute a Command in Docker Container**

Use:

bash

CopyEdit

docker exec -it <container\_id\_or\_name> bash

This opens a terminal inside the container.

**9. RUN vs CMD in Dockerfile**

| **RUN** | **CMD** |
| --- | --- |
| Used during build time | Used during container runtime |
| Executes and commits result in image | Specifies default command when container starts |

**10. What is Docker Containerization?**

👉 Docker containerization is a technology that **packages an app and its dependencies** into a lightweight, portable container that runs the same everywhere.

**11. Difference Between CMD and ENTRYPOINT in Dockerfile**

| **CMD** | **ENTRYPOINT** |
| --- | --- |
| Provides default command | Sets fixed executable |
| Can be overridden easily | Harder to override (unless --entrypoint) |
| Example: CMD ["nginx"] | Example: ENTRYPOINT ["nginx"] |

**12. What is Docker?**

👉 Docker is a tool that allows developers to **build, package, and run applications** in containers. Containers are **lightweight, portable, and consistent** across environments.

**13. What is a Dockerfile?**

👉 A **Dockerfile** is a script that contains a list of instructions to build a Docker image (e.g., install software, copy files, set environment variables).

**14. Difference Between CMD and RUN in Dockerfile**

| **RUN** | **CMD** |
| --- | --- |
| Executes during build | Executes when container starts |
| Example: RUN apt install | Example: CMD ["python", "app.py"] |

**15. Difference Between CMD and ENTRYPOINT in Dockerfile**

(Repeated — see #11 above)

**16. What is ARG in Dockerfile?**

ARG is used to define **build-time variables** in a Dockerfile.  
Example:

dockerfile

CopyEdit

ARG VERSION=1.0

RUN echo $VERSION

**17. How to build an image in Docker?**

Use the docker build command:

bash

CopyEdit

docker build -t my-image-name .

The . refers to the location of the Dockerfile.

**18. Docker scenario: Updating an application in a container without disrupting others**

* Use **separate containers** for each service.
* Deploy the updated app version in a **new container**.
* Use a **load balancer** to redirect traffic.
* Replace old container once the new one is verified.

**19. Docker networks**

Docker provides three types of networks:

* bridge (default for containers on the same host)
* host (uses host’s networking)
* none (no networking)  
  Create custom network:

bash

CopyEdit

docker network create mynetwork

**20. Docker Compose**

Tool to **define and run multi-container apps** using a YAML file.  
File: docker-compose.yml  
Command:

bash

CopyEdit

docker-compose up -d

**21. How to stop a Docker container?**

bash

CopyEdit

docker stop <container\_id\_or\_name>

**22. What is Docker?**

Docker is a **containerization platform** that packages apps with all dependencies to run reliably in any environment.

**23. Why use Docker instead of directly providing code to the client?**

* Ensures **consistency** across environments
* Reduces dependency issues
* Easy to deploy and rollback
* Lightweight and **isolated runtime**

**24. Steps to do containerization**

1. Install Docker
2. Write a Dockerfile
3. Build the image with docker build
4. Run the image with docker run
5. Push to Docker Hub (optional)

**25. Docker networking**

Docker networking allows containers to **communicate** with each other and external systems.  
You can define custom networks using:

bash

CopyEdit

docker network create mynetwork

**26. Dockerfile and its use**

A Dockerfile is a **script of instructions** to create a Docker image (e.g., install software, copy files).  
It automates container image creation.

**27. Commands for containers (list, start, stop, remove)**

* List: docker ps -a
* Start: docker start <container>
* Stop: docker stop <container>
* Remove: docker rm <container>

**28. How to run a Docker container?**

bash

CopyEdit

docker run -it --name mycontainer ubuntu

-it: interactive terminal  
--name: custom container name

**29. Docker commands for creating containers and volume mapping**

bash

CopyEdit

docker run -v myvolume:/data --name myapp ubuntu

**30. Deployment file for Docker**

Use docker-compose.yml for deployments:

yaml

CopyEdit

version: '3'

services:

web:

image: nginx

ports:

- "80:80"

**31. Write a Dockerfile for nginx**

dockerfile

CopyEdit

FROM nginx

COPY ./index.html /usr/share/nginx/html/index.html

**32. What is the logging driver in Docker?**

It defines how container logs are collected.  
Examples: json-file (default), syslog, journald, fluentd  
Set it like this:

bash

CopyEdit

docker run --log-driver=json-file nginx

**33. Docker-compose file to build an image**

yaml

CopyEdit

version: '3'

services:

app:

build: .

ports:

- "8080:80"

**34. Docker up, Docker run, Docker start in detail**

* docker run: Creates & starts a new container
* docker start: Starts a **stopped** container
* docker-compose up: Builds and starts all services in docker-compose.yml

**35. List Docker commands?**

* docker build
* docker run
* docker ps
* docker stop
* docker rm
* docker rmi
* docker logs
* docker exec
* docker volume
* docker network
* docker-compose up

**36. What is meant by Dockerfile?**

A **Dockerfile** is a script with instructions to build a Docker image.  
It defines the **base image, dependencies, files, and default commands**.

Kubernetes

**1. Difference between Kubernetes and Docker**

* **Docker**: A containerization tool used to create and run containers.
* **Kubernetes**: An orchestration tool that **manages** containers (like Docker), scaling them, handling networking, etc.

**2. What is a Replication Set?**

A **ReplicaSet** ensures a specific number of **pod replicas** are running at all times. If a pod dies, it creates a new one.

**3. Kubernetes Structure**

* **Master Node**: Controls the cluster. Has API server, scheduler, etc.
* **Worker Nodes**: Run the application (pods).
* **Pods**: Smallest unit in Kubernetes.
* **Controller, Scheduler, Kubelet, etc.**

**4. Networking in Kubernetes**

* Every pod gets a unique **IP address**.
* Pods can talk to each other directly.
* Uses **CNI (Container Network Interface)** plugins (like Flannel, Calico) for setup.

**5. ReplicaSet**

Same as Replication Set – ensures a certain number of **identical pods** are always running.

**6. How to get a pod?**

bash

CopyEdit

kubectl get pods

**7. How to launch a pod in Kubernetes?**

Create a YAML file or use:

bash

CopyEdit

kubectl run mypod --image=nginx

**8. What is Kubernetes Architecture?**

* **Control Plane (Master)**: API Server, Controller Manager, Scheduler, etcd.
* **Node**: kubelet, container runtime (like Docker), kube-proxy.
* **Pods** run on nodes.

**9. Kubernetes Scaling Deployment**

Use:

bash

CopyEdit

kubectl scale deployment myapp --replicas=5

This increases/decreases pod count.

**10. Deployment Strategies in Kubernetes**

* **Rolling Update** (default): Gradually updates pods.
* **Recreate**: Deletes old pods, then creates new ones.
* **Blue/Green** and **Canary** strategies are also used.

**11. What is Node Affinity and Pod Affinity?**

* **Node Affinity**: Schedule pods on specific nodes (based on labels).
* **Pod Affinity**: Schedule pods near/with other specific pods.
* **Anti-Affinity**: Opposite (avoid certain nodes or pods).

**12. What is the default Kubernetes network?**

There is no specific "default network", but Kubernetes expects **all pods to communicate freely**. CNI plugins provide this network.

**13. Commands in Kubernetes**

* kubectl get pods
* kubectl get nodes
* kubectl describe pod <name>
* kubectl apply -f <file>.yaml
* kubectl delete pod <name>

**14. How to kill Kubernetes pods?**

bash

CopyEdit

kubectl delete pod <pod-name>

**15. How to deploy a microservice in Kubernetes?**

1. Write a Deployment YAML for the microservice
2. Expose it using a **Service**
3. Apply using:

bash

CopyEdit

kubectl apply -f deployment.yaml

**16. Kubernetes fmt command**

There is **no kubectl fmt**, but YAML files must follow **proper format and indentation**. Tools like yamllint help.

**17. How to handle replicas and crash issues?**

* Use **ReplicaSets** or **Deployments**
* Set restartPolicy: Always
* Use **Liveness** and **Readiness probes**

**18. StatefulSet with PV**

* **StatefulSet**: Used for apps needing **stable identity and storage** (like DBs).
* Needs **Persistent Volume Claims (PVC)** per pod.

**19. What is Ingress? Which Ingress are you working on?**

* **Ingress**: Manages **external access** (like HTTP) to services.
* Ingress Controller needed (e.g., NGINX Ingress).  
  Used to route traffic based on paths or hostnames.

**20. What is Kubernetes, and why use it?**

Kubernetes is an **open-source container orchestration** platform.  
Used to **automate deployment, scaling, and management** of containerized applications.

**21. Kubernetes architecture (control plane, worker, and master nodes)**

* **Control Plane** (Master Node):
  + **API Server**: Entry point for all commands.
  + **Scheduler**: Decides where pods run.
  + **Controller Manager**: Manages states.
  + **etcd**: Key-value store for cluster data.
* **Worker Nodes**:
  + **Kubelet**: Talks to the master.
  + **kube-proxy**: Handles networking.
  + **Container Runtime** (Docker/Containerd): Runs containers.

**22. Different Kubernetes strategies**

* **Rolling Update** (default): Update pods one by one.
* **Recreate**: Kill old pods, then create new ones.
* **Blue/Green**: Two environments; switch traffic.
* **Canary**: Release to small set before full rollout.

**23. Kubernetes StatefulSet and DaemonSet**

* **StatefulSet**: For apps that need unique network/storage (like databases).
* **DaemonSet**: Runs one pod on **every node** (e.g., log agents, monitoring).

**24. Deployment vs. StatefulSet**

| **Feature** | **Deployment** | **StatefulSet** |
| --- | --- | --- |
| Use case | Stateless apps | Stateful apps (DBs) |
| Pod names | Random | Ordered (e.g., pod-0, pod-1) |
| Storage | Shared | Dedicated (Persistent) |

**25. Commands for Kubernetes (create deployment, pods, services)**

bash

CopyEdit

kubectl create deployment myapp --image=nginx

kubectl run mypod --image=nginx

kubectl expose deployment myapp --type=NodePort --port=80

**26. Service types in Kubernetes**

* **ClusterIP** (default): Internal access only.
* **NodePort**: Access via <NodeIP>:<Port>.
* **LoadBalancer**: Uses cloud provider to expose externally.
* **ExternalName**: Maps to external DNS name.

**27. How to deploy MySQL DB in a Kubernetes cluster**

1. Create a **PVC** for data storage.
2. Create a **Deployment** using mysql image.
3. Set environment variables for DB name/user/password.
4. Create a **Service** to expose MySQL.

**28. PV and PVC in Kubernetes**

* **PV (Persistent Volume)**: Storage provided by admin or cloud.
* **PVC (Persistent Volume Claim)**: Request for storage by pods.  
  PVC is like a "booking" of PV by the pod.

**29. Kubernetes manifest files**

YAML files that define:

* Deployments
* Pods
* Services
* ConfigMaps, PVCs, etc.  
  Used with:

bash

CopyEdit

kubectl apply -f myfile.yaml

**30. How to resolve pod issues**

1. Check status:

bash

CopyEdit

kubectl get pods

1. Describe the pod:

bash

CopyEdit

kubectl describe pod <pod-name>

1. Check logs:

bash

CopyEdit

kubectl logs <pod-name>

1. Use kubectl exec to debug inside the pod.

**31. How to connect two Kubernetes nodes**

They should be part of the **same cluster**. Use kubeadm to join:

bash

CopyEdit

kubeadm join <master-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>

**32. Explain Kubernetes architecture**

(Repeated from #21) — Master node controls the cluster (API server, etcd), worker nodes run the apps (pods) using Kubelet and container runtime. All communication goes through API server.

**33. How do you access the pod through browser?**

1. Expose pod via a **Service** (NodePort or LoadBalancer)
2. Get node IP and port:

bash

CopyEdit

kubectl get svc

1. Open in browser:  
   http://<NodeIP>:<NodePort>

**34. How do you check logs of a pod?**

bash

CopyEdit

kubectl logs <pod-name>

Jenkins

CI tool it manages and control processes such as plan, code, build, test, deploy and monitor in devops environment.

**1. How to manage credentials in Jenkins?**

* Jenkins provides a **Credentials Plugin** to securely store and manage sensitive data such as usernames, passwords, API tokens, SSH keys, and certificates.
* You can add credentials via **Manage Jenkins > Manage Credentials**, organized by domains or scopes (global, folder, or job).
* Credentials are accessed in pipelines using credentials() helper or environment variables, ensuring secrets are not exposed in logs.

**2. What plugins are used in Jenkins?**

* Common Jenkins plugins include:
  + **Git Plugin** — to integrate Git repositories.
  + **Pipeline Plugin** — to define build pipelines as code.
  + **Credentials Plugin** — for secret management.
  + **Docker Plugin** — to build and run Docker containers.
  + **Blue Ocean** — modern pipeline visualization UI.
  + **JUnit Plugin** — to publish test results.
  + **Email Extension Plugin** — to send build notifications.
  + **Kubernetes Plugin** — to run agents dynamically on Kubernetes.
* The plugin list varies based on project needs.

**3. Jenkins Pipeline**

* A **Jenkins Pipeline** is a suite of plugins that supports implementing and integrating continuous delivery pipelines as code.
* Pipelines are defined in a **Jenkinsfile** using **Groovy** DSL.
* Pipelines have **stages** and **steps** for tasks like build, test, and deploy.
* Two types:
  + **Declarative Pipeline** (structured, easier syntax)
  + **Scripted Pipeline** (more flexible, scripted Groovy)

**4. Multi-pipeline in Jenkins**

* Multi-pipeline means managing multiple Jenkins pipelines concurrently.
* You can:
  + Organize pipelines using **folders** or **multibranch pipelines**.
  + Use **Multibranch Pipeline Plugin** to automatically detect branches in a repo and create a pipeline per branch.
  + Trigger downstream pipelines using **pipeline triggers** or **build job steps** for complex workflows.

**5. Working of Jenkins**

* Jenkins runs as a server that listens for build jobs.
* It fetches source code, executes build/test scripts, and reports results.
* It uses **agents (nodes)** to distribute workload.
* Jobs can be triggered by SCM changes, schedules, or manual start.
* Jenkins pipelines automate the entire CI/CD workflow.

**6. Jenkins workspace**

* The **workspace** is the directory on the Jenkins agent machine where the job runs.
* It contains the checked-out source code, build artifacts, and temporary files.
* Each job gets a separate workspace, which is reused unless cleaned up.

**7. Jenkins default port number?**

* Jenkins default port number is **8080**.

**8. Jenkins port number?**

* The Jenkins port can be changed by editing the startup configuration or command line.
* Default is **8080**, but it can be customized (e.g., 9090 or 80).

**9. How do you backup Jenkins?**

* Backup Jenkins by:
  + Backing up the **JENKINS\_HOME** directory (contains configs, jobs, plugins, credentials).
  + Exporting job configurations via UI or script.
  + Using plugins like **ThinBackup** or **SCM Sync Configuration Plugin**.
  + Regularly backing up build artifacts and externalized data.

**10. How can you integrate monitoring tools in Jenkins?**

* Jenkins can be integrated with monitoring tools by:
  + Using the **Prometheus Plugin** to expose Jenkins metrics.
  + Integrating with **Grafana** to visualize Prometheus metrics.
  + Using **Nagios**, **Zabbix**, or **ELK Stack** for logs and alerts.
  + Jenkins also supports webhooks and API calls for external monitoring systems.

**11. How will you configure Kubernetes in Jenkins?**

* Configure Jenkins with Kubernetes by:
  + Installing the **Kubernetes Plugin**.
  + Adding Kubernetes cloud in Jenkins configuration with cluster API URL, credentials, and namespace.
  + Define pod templates for agent containers.
  + Jenkins dynamically spins up pods as agents for builds, providing scalable, isolated environments.

**12. What is a node in Jenkins?**

* A **node** is a machine (physical or virtual) that runs build jobs.
* The default node is called **Master**, which runs Jenkins server.
* Additional nodes (agents/slaves) can be connected for distributed builds, improving scalability.

**13. Write a Groovy script for the Jenkins pipeline and explain the stages.**

agent any // here any = current available ec2 server where jenkins is installed

tools{

maven 'mymaven'

}

stages{

stage('Checkout Code')

{

steps{

git 'https://github.com/Sonal0409/DevOpsCodeDemo.git'

}

}

stage('Compile Code'){

steps{

echo "compiling the code...."

sh 'mvn compile'

}

}

stage('Test code'){

steps{

echo "testing the code using maven surefire"

sh 'mvn test'

}

}

stage('BuildCode'){

steps{

echo "generating artifact..."

sh 'mvn package'

}

}

}

}

**Explanation:**

* **agent any**: Runs the pipeline on any available Jenkins node.
* **stage('Checkout')**: Clones the source code from Git.
* **stage('Build')**: Runs build commands using Maven.
* **stage('Test')**: Executes tests and publishes test results.
* **stage('Deploy')**: Runs deployment script.

Each stage groups related tasks to organize the CI/CD flow clearly.

**14. What are Jenkins environment variables?**

* Jenkins environment variables provide runtime information like build number, job name, workspace path, etc.
* Examples include:
  + BUILD\_NUMBER: Current build number.
  + JOB\_NAME: Name of the job.
  + WORKSPACE: Directory where the job runs.
  + JENKINS\_URL: Jenkins server URL.
* They can be used in pipelines or freestyle jobs to customize build steps dynamically.

**15. What are the types of jobs in Jenkins?**

* Jenkins supports multiple job types, such as:
  + **Freestyle Project**: Simple, configurable build job.
  + **Pipeline**: Job defined with code using Jenkinsfile (Declarative or Scripted).
  + **Multibranch Pipeline**: Automatically creates pipelines per branch in a repo.
  + **Matrix (or Multi-configuration) Project**: Runs the same job across multiple configurations (e.g., OS, JDK versions).
  + **Folder**: Organizes jobs into groups.
  + **GitHub Organization**: Automatically discovers repositories and branches.

**16. How to secure Jenkins?**

* Secure Jenkins by:
  + Enabling **Authentication** using built-in user database or external systems (LDAP, OAuth).
  + Enabling **Authorization** with Role-Based Access Control (RBAC) or Project-based Matrix Authorization.
  + Using **HTTPS** to encrypt Jenkins UI traffic.
  + Restricting access to **JENKINS\_HOME** and server.
  + Keeping Jenkins and plugins updated.
  + Securing credentials with Jenkins credentials plugin.
  + Limiting script approval in pipelines.

**17. Where are Jenkins files and source code stored?**

* Jenkins stores configuration files, job configurations, plugins, credentials, and build artifacts in the **JENKINS\_HOME** directory.
* Source code for builds is checked out inside the **workspace** folder under each job’s directory in JENKINS\_HOME.

**18. Master and slave in Jenkins**

* **Master** (now called Controller) runs Jenkins core, manages jobs, schedules builds, and provides UI.
* **Slave** (Agent) is a separate machine configured to run build jobs delegated by the master.
* Using slaves helps distribute load, parallelize builds, and run jobs on specific environments.

**19. Run deployment on a specific slave node**

* To run deployment on a specific slave, use the node label in the pipeline:

groovy

CopyEdit

pipeline {

agent { label 'deployment-node' }

stages {

stage('Deploy') {

steps {

sh './deploy.sh'

}

}

}

}

* Alternatively, in freestyle jobs, configure the job to restrict where it runs by selecting the specific node.

**20. Jenkins server backup**

* Backup the entire **JENKINS\_HOME** directory, which contains:
  + Job configurations
  + Plugin data
  + Credentials
  + Build history
  + User data
* You can do this manually or use backup plugins.
* Store backups offsite or in version control for disaster recovery.

**21. How to get Jenkins password back if lost?**

* If using the **default Jenkins user**, find the initial admin password at:

swift

CopyEdit

/var/lib/jenkins/secrets/initialAdminPassword

* For other users, reset the password by:
  + Modifying the Jenkins user database file.
  + Using an admin user to reset via UI.
  + Restart Jenkins in safe mode and disable security temporarily.
* For external authentication, reset in the external system.

**22. How to take Jenkins Backup plugin?**

* Install the **ThinBackup Plugin** from Jenkins Plugin Manager.
* Configure ThinBackup via **Manage Jenkins > ThinBackup**:
  + Set backup location.
  + Schedule backup frequency.
  + Backup Jenkins config, jobs, plugins, and builds.
* Use it for automated, scheduled backups.

**23. What will you do when build is failing in Jenkins?**

* Steps to handle a failing build:
  + Check **build logs** to identify the error.
  + Verify recent **code changes** or dependencies.
  + Confirm that the environment or agents are properly configured.
  + Re-run the build to check for intermittent issues.
  + Check if external services (e.g., database, network) are available.
  + Communicate failures to the team via email/slack notifications.
  + Fix code, test scripts, or environment and re-trigger the build.
  + Use **Jenkins pipeline** features like try-catch and post-build actions for failure handling.

CI/CD Pipeline

**1. Describe CI/CD Pipeline and how it works.**

* **CI/CD (Continuous Integration/Continuous Deployment)** pipeline automates the software delivery process.
* **Continuous Integration (CI):** Developers frequently merge code changes into a shared repo. Automated builds and tests verify changes to detect errors early.
* **Continuous Deployment (CD):** Automates release of validated code to production or staging environments.
* **How it works:**
  1. Code commit triggers the pipeline.
  2. Source code is checked out.
  3. Build phase compiles and packages the code.
  4. Automated tests run.
  5. If tests pass, code is deployed automatically or manually approved for deployment.
  6. Monitoring tools verify health after deployment.

**2. How will you integrate with Maven?**

* Jenkins integrates with Maven by:
  + Installing the **Maven plugin** in Jenkins.
  + Configuring Maven installations in **Manage Jenkins > Global Tool Configuration**.
  + In pipeline or freestyle jobs, specify Maven goals like clean install, package, or test.
  + Use Maven commands in the build steps to build and test projects.
  + Jenkins can also archive Maven build artifacts and test reports.

**3. Maven commands**

* Common Maven commands:
  + mvn clean: Cleans the target directory.
  + mvn compile: Compiles source code.
  + mvn test: Runs unit tests.
  + mvn package: Packages compiled code into a JAR/WAR.
  + mvn install: Installs the package into the local repository.
  + mvn deploy: Copies the package to a remote repository.
  + mvn validate: Validates the project is correct.
  + mvn site: Generates project documentation.

**4. What is a CI/CD deployment flow?**

* Typical CI/CD deployment flow:
  1. Developer pushes code to version control.
  2. CI server triggers build and runs tests.
  3. On success, the build artifact is created.
  4. Artifact is stored in artifact repository.
  5. CD tool deploys the artifact to staging.
  6. Automated integration and acceptance tests run.
  7. Upon approval, deployment to production occurs.
  8. Monitoring and alerting ensure the application is stable.

**5. Tools integrated into Jenkins**

* Jenkins commonly integrates with:
  + **Source Control:** Git, SVN.
  + **Build Tools:** Maven, Gradle, Ant.
  + **Containerization:** Docker.
  + **Configuration Management:** Ansible, Puppet, Chef.
  + **Testing:** Selenium, JUnit.
  + **Notification:** Email, Slack.
  + **Artifact Repository:** Nexus, Artifactory.
  + **Cloud Providers:** AWS, Azure, GCP.
  + **Monitoring:** Prometheus, Grafana.

**6. Have you written any Jenkins pipelines? Explain each step.**

* Example pipeline steps:
  1. **Checkout**: Pull source code from Git repository.
  2. **Build**: Compile and package the code (e.g., with Maven).
  3. **Test**: Run unit or integration tests.
  4. **Static Analysis**: Run code quality tools (optional).
  5. **Archive**: Save build artifacts.
  6. **Deploy**: Push to dev or staging environment.
  7. **Notify**: Send build status to team via email or Slack.
* Each step ensures smooth, automated software delivery.

**7. Jenkins job migration between servers**

* To migrate jobs:
  1. Stop Jenkins on source and target servers.
  2. Copy job directories from JENKINS\_HOME/jobs/ on source to target.
  3. Copy any custom configurations, plugins, and credentials.
  4. Start Jenkins on the target server.
  5. Verify jobs appear and run correctly.
* Alternatively, use plugins like **Job Import Plugin** or export/import Jenkins jobs via UI.

**8. Dockerfile and Jenkins pipeline integration**

* In Jenkins pipeline, you can build Docker images using a Dockerfile:

groovy

CopyEdit

pipeline {

agent any

stages {

stage('Build Docker Image') {

steps {

script {

docker.build('myapp:latest')

}

}

}

stage('Run Container') {

steps {

script {

docker.image('myapp:latest').run()

}

}

}

}

}

* This builds a Docker image from the Dockerfile and runs the container as part of the pipeline.
* Allows packaging and deployment as containers seamlessly.

**9. Observability/monitoring in production (e.g., CloudWatch)**

* Observability includes monitoring metrics, logs, and traces to understand system behavior.
* **AWS CloudWatch** collects logs, metrics, and events from AWS resources.
* Jenkins or applications can push logs and metrics to CloudWatch.
* Alerts can be configured on metrics thresholds.
* Integration helps detect and troubleshoot production issues quickly.

Terraform

**1. What is Terraform, and what does it provide?**

* **Terraform** is an open-source Infrastructure as Code (IaC) tool by HashiCorp.
* It allows you to define, provision, and manage infrastructure across multiple cloud providers (AWS, Azure, GCP) and services using a declarative configuration language called **HCL (HashiCorp Configuration Language)**.
* Terraform provides **automation, consistency, version control, and infrastructure lifecycle management**.

**2. Terraform components**

* Key components of Terraform:
  + **Providers:** Plugins that interact with cloud platforms or services (e.g., AWS, Azure).
  + **Resources:** Infrastructure objects to create/manage (e.g., EC2 instance, S3 bucket).
  + **Modules:** Reusable, composable groups of resources.
  + **State File:** Tracks infrastructure state locally or remotely.
  + **Configuration files (.tf):** Define desired infrastructure in HCL.
  + **Variables:** Input parameters to make configurations flexible.
  + **Outputs:** Export values from Terraform for other tools or human consumption.

**3. Terraform plan vs Terraform apply**

* **terraform plan**:
  + Creates an execution plan without making changes.
  + Shows what actions Terraform will take to achieve the desired state.
  + Useful for review and validation before actual changes.
* **terraform apply**:
  + Executes the planned changes to provision or modify resources.
  + Updates infrastructure to match the desired state defined in config files.

**4. Stages in Terraform**

* Typical Terraform workflow stages:
  1. **Write**: Define infrastructure in .tf files.
  2. **Initialize (terraform init)**: Download providers and initialize the working directory.
  3. **Plan (terraform plan)**: Preview changes.
  4. **Apply (terraform apply)**: Provision or update resources.
  5. **Destroy (terraform destroy)**: Tear down infrastructure.

**5. How to configure a server using Terraform?**

* Example: To configure an AWS EC2 server, you write a resource block in Terraform:

hcl

CopyEdit

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "web" {

ami = "ami-0c55b159cbfafe1f0"

instance\_type = "t2.micro"

tags = {

Name = "MyWebServer"

}

}

* Run terraform init, terraform plan, and terraform apply to create the server.

**6. How to rename resources in Terraform?**

* Terraform does not support renaming resources directly.
* To rename a resource:
  1. Use terraform state mv <old\_resource> <new\_resource> to update the state file.
  2. Rename the resource block in your .tf files.
* This keeps the resource intact but updates Terraform’s tracking.

**7. How does Terraform trigger changes after editing or updating resources?**

* Terraform compares the **current state** (from the state file) with the **desired state** (from configuration files).
* On running terraform plan or apply, Terraform detects differences and plans changes (create, update, delete).
* It uses **resource providers’ APIs** to implement these changes.

**8. What is a Terraform state file?**

* The **state file (terraform.tfstate)** stores metadata and mappings of real infrastructure to your configuration.
* It tracks resource IDs, dependencies, and outputs.
* Used by Terraform to know what exists and what changes are needed.
* Can be stored locally or remotely (e.g., S3, Terraform Cloud) for team collaboration.

**9. What is Terraform?**

* Terraform is an **open-source Infrastructure as Code (IaC)** tool by HashiCorp.
* It enables users to define, provision, and manage infrastructure across cloud providers and services using declarative configuration files.
* Terraform automates infrastructure deployment, ensuring consistency, repeatability, and version control.

**10. Use of the provider in Terraform**

* **Providers** are plugins that allow Terraform to interact with APIs of cloud platforms or services.
* They define the resources Terraform can manage (e.g., AWS provider manages EC2, S3).
* Each provider requires configuration, typically including authentication and region details.
* Providers act as the bridge between Terraform and the target infrastructure.

**11. Difference between Ansible and Terraform**

| **Aspect** | **Terraform** | **Ansible** |
| --- | --- | --- |
| Type | Infrastructure provisioning tool (IaC) | Configuration management & automation tool |
| Purpose | Build and manage infrastructure resources | Configure software, deploy applications |
| State Management | Maintains state file to track infrastructure | No persistent state, idempotent operations |
| Language | HashiCorp Configuration Language (HCL) | YAML playbooks |
| Execution | Declarative, plans changes before applying | Procedural, executes tasks in order |
| Use Case | Provision servers, networking, cloud infra | Install packages, configure services on servers |

**12. Terraform commands**

* terraform init — Initialize working directory, download providers.
* terraform plan — Preview changes before applying.
* terraform apply — Apply changes to infrastructure.
* terraform destroy — Delete all resources managed by Terraform.
* terraform fmt — Format configuration files.
* terraform validate — Validate config syntax.
* terraform state — Inspect and modify the state file.
* terraform output — Show outputs defined in config.

**13. Terraform modules**

* Modules are **reusable, self-contained Terraform configurations**.
* They help organize and encapsulate resource definitions.
* Can be **local** (within your repo) or **remote** (from Terraform Registry, GitHub).
* Promote code reuse, reduce duplication, and simplify management of complex infrastructure.

**14. Terraform state file**

* The **state file (terraform.tfstate)** keeps track of real-world infrastructure created by Terraform.
* Maps resource definitions in config to actual resources.
* Helps Terraform detect changes, plan updates, and avoid duplicate resource creation.
* Can be stored locally or remotely for collaboration and locking.

**15. Terraform code for an S3 bucket**

hcl

CopyEdit

provider "aws" {

region = "us-east-1"

}

resource "aws\_s3\_bucket" "my\_bucket" {

bucket = "my-unique-bucket-name-12345"

acl = "private"

tags = {

Environment = "Dev"

Name = "MyTerraformBucket"

}

}

**16. Integration of Jenkins with Terraform for managing AWS infrastructure**

* **Steps:**
  1. Install Terraform CLI on Jenkins server or agents.
  2. Configure AWS credentials (using Jenkins Credentials Plugin) securely in Jenkins.
  3. Create a Jenkins pipeline or freestyle job with steps:
     + terraform init to initialize.
     + terraform plan to preview infrastructure changes.
     + terraform apply to provision/update AWS resources.
  4. Use Jenkins environment variables or parameters to customize Terraform variables.
  5. Automate infrastructure provisioning as part of CI/CD workflows.
* This integration allows Infrastructure as Code to be versioned, tested, and deployed automatically.

DevOps

**1. Why DevOps? How does it help in IT?**

* **DevOps** is a cultural and technical approach combining development (Dev) and operations (Ops) teams.
* It helps deliver software faster, with higher quality and reliability by enabling continuous integration, continuous delivery, automation, and collaboration.
* Benefits include:
  + Faster time-to-market.
  + Improved collaboration between teams.
  + Reduced deployment failures and faster recovery.
  + Continuous feedback for improvements.
  + Efficient resource utilization.

**2. What is Prometheus and Grafana?**

* **Prometheus** is an open-source monitoring and alerting toolkit focused on collecting and storing metrics data.
* It scrapes metrics from configured targets at regular intervals and stores them in a time-series database.
* **Grafana** is an open-source visualization and analytics platform.
* It connects to Prometheus (and other data sources) to create interactive dashboards and graphs for monitoring.

**3. Prometheus and Grafana log case study**

* In real-world use cases, Prometheus collects key metrics (CPU, memory, response times) from applications and infrastructure.
* Grafana visualizes these metrics in dashboards, enabling quick identification of performance bottlenecks or failures.
* Example: A microservices architecture monitored by Prometheus helps detect slow services; Grafana dashboards alert the DevOps team to fix before customers are impacted.
* Together, they provide full observability: Prometheus for data collection and alerting, Grafana for visualization and analysis.

**4. Default port number of Prometheus and Grafana**

* **Prometheus** default port: 9090
* **Grafana** default port: 3000

**5. How Prometheus and Grafana work**

* **Prometheus** scrapes metrics endpoints of services/applications periodically.
* It stores the collected data as time-series data.
* Users can query data using PromQL (Prometheus Query Language).
* **Grafana** connects to Prometheus and queries this data.
* It displays data via customizable dashboards with charts, graphs, and alerts.
* Alerts can be configured in Prometheus and visualized in Grafana.

**6. What is ECS?**

* **ECS (Elastic Container Service)** is AWS's fully managed container orchestration service.
* It allows running and scaling Docker containers on AWS infrastructure.
* Supports deploying containerized applications without managing underlying servers.
* Integrates with AWS services like IAM, ELB, CloudWatch.

**7. What do you mean by DevOps?**

* DevOps is a **set of practices, culture, and tools** that increase an organization's ability to deliver applications and services at high velocity.
* It bridges the gap between software development and IT operations, fostering collaboration and automation throughout the software lifecycle.

**8. What are the tools in DevOps?**

* DevOps uses various tools categorized by function:
  + **Version Control:** Git, GitHub, GitLab.
  + **CI/CD:** Jenkins, GitLab CI, CircleCI.
  + **Configuration Management:** Ansible, Chef, Puppet.
  + **Containerization:** Docker, Kubernetes.
  + **Infrastructure as Code:** Terraform, CloudFormation.
  + **Monitoring:** Prometheus, Grafana, Nagios.
  + **Collaboration:** Jira, Slack.
  + **Artifact Management:** Nexus, Artifactory.

**9. What do you mean by CI/CD?**

* **CI/CD (Continuous Integration and Continuous Deployment/Delivery)** is an automated software development practice.
* **Continuous Integration (CI):** Developers frequently merge code changes to a shared repo, triggering automated builds and tests.
* **Continuous Deployment (CD):** Automated deployment of validated code to production without manual intervention.
* This approach ensures faster releases, higher quality, and immediate feedback.

Testing

**1. What is TestNG and its annotations?**

* **TestNG** is a testing framework inspired by JUnit and NUnit, designed for Java.
* It supports **annotations**, flexible test configuration, parallel execution, and data-driven testing.
* Common TestNG annotations:
  + @BeforeSuite — Runs once before all tests.
  + @BeforeTest — Runs before any test belonging to <test> tag in testng.xml.
  + @BeforeClass — Runs once before the first method in the current class.
  + @BeforeMethod — Runs before each test method.
  + @Test — Marks a method as a test case.
  + @AfterMethod — Runs after each test method.
  + @AfterClass — Runs once after all test methods in the class.
  + @AfterTest — Runs after all tests in a <test> tag.
  + @AfterSuite — Runs once after all tests.

**2. New feature added: What type of testing will you do?**

* When a **new feature is added**, the common testing types are:
  + **Unit Testing:** Verify the new feature's individual components.
  + **Integration Testing:** Check how the new feature interacts with existing modules.
  + **Functional Testing:** Validate that the feature works according to requirements.
  + **Regression Testing:** Ensure existing functionality is not broken by the new feature.
  + **User Acceptance Testing (UAT):** Confirm the feature meets end-user needs.
  + **Performance Testing (if applicable):** Ensure new feature does not degrade performance.

**3. Difference between verification and validation**

| **Aspect** | **Verification** | **Validation** |
| --- | --- | --- |
| Definition | Process of evaluating work-products to ensure they meet specified requirements | Process of evaluating the final product to check if it meets user needs |
| Focus | Are we building the product right? | Are we building the right product? |
| When | Early stages (reviews, walkthroughs, inspections) | Later stages (testing phases) |
| Nature | Static process | Dynamic process |
| Example | Code reviews, design reviews | System testing, user acceptance testing |

**4. What is the defect lifecycle?**

* The **Defect Lifecycle** defines the states a defect goes through from identification to closure.
* Typical stages:
  1. **New:** Defect is logged.
  2. **Assigned:** Developer is assigned to fix it.
  3. **Open:** Developer starts working on the defect.
  4. **Fixed:** Developer fixes the defect.
  5. **Retest:** Tester verifies the fix.
  6. **Closed:** Defect is confirmed fixed.
  7. **Reopen:** If the defect is not fixed properly, it can be reopened.
  8. **Deferred:** Defect will be fixed in future releases.
  9. **Rejected:** Defect is invalid or not a bug.

**5. Examples of high severity and low priority bugs**

* **High Severity, Low Priority:**
  + A bug causing data corruption but only occurring in a rarely used feature.
  + A critical security vulnerability affecting a non-critical module with no immediate threat.
* **Low Severity, High Priority:**
  + A typo on the homepage of a widely used website.
  + UI misalignment on a frequently visited page that affects user experience but doesn’t break functionality.

Miscellaneous

**1. What is Ansible?**

* **Ansible** is an open-source automation tool used for configuration management, application deployment, and task automation.
* It uses **agentless architecture**, connecting to nodes via SSH.
* Configurations are written in simple, human-readable YAML files called **playbooks**.
* Ansible is known for its simplicity, scalability, and minimal learning curve.

**2. Ansible roles and playbooks**

* **Playbook:** A YAML file that defines a series of tasks to be executed on managed nodes, describing the desired state of systems.
* **Role:** A reusable, modular unit in Ansible that organizes tasks, variables, files, and templates into a standard directory structure.
* Roles promote code reusability and simplify complex playbooks by encapsulating functionality.

**3. Inventory file in Ansible**

* The **Inventory file** lists managed nodes (hosts) grouped logically.
* It can be a static file (INI or YAML format) or dynamic (script-based).
* Example:

ini

CopyEdit

[webservers]

web1.example.com

web2.example.com

[dbservers]

db1.example.com

* Ansible uses the inventory to know which machines to manage.

**4. Define Chef**

* **Chef** is an automation platform for infrastructure configuration management.
* It uses **recipes** and **cookbooks** written in Ruby DSL to define system configurations.
* Chef follows a client-server architecture where nodes pull configurations from a central Chef server.
* It helps automate provisioning, configuration, and management of servers.

**5. What is the Terraform state file?**

* The **Terraform state file (terraform.tfstate)** records the current state of infrastructure managed by Terraform.
* It maps Terraform configuration to real-world resources.
* Helps Terraform track resources, detect changes, and manage updates safely.
* Can be stored locally or remotely for collaboration.

**6. How to change the username to another username in AWS?**

* If you mean **changing IAM username**:
  + AWS does not allow directly renaming IAM users.
  + To change a username, you must **create a new IAM user** with the desired username.
  + Then, migrate permissions, access keys, and policies from the old user to the new one.
  + Finally, delete the old user once migration is complete.
* Alternatively, use the AWS CLI or Console’s **“Edit user”** option to manage details but not rename.

General DevOps

**What is Terraform?**

* Terraform is an open-source **Infrastructure as Code (IaC)** tool by HashiCorp.
* It allows you to define and provision infrastructure across multiple cloud providers declaratively using configuration files.
* Automates infrastructure deployment and management, enabling version control, repeatability, and scalability.

**What do you know about DevOps?**

* DevOps is a **culture, practice, and set of tools** aimed at uniting software development (Dev) and IT operations (Ops).
* It emphasizes **collaboration, automation, continuous integration, continuous delivery (CI/CD), and monitoring** to deliver software faster and more reliably.
* DevOps helps reduce deployment failures, improve recovery times, and enhance overall software quality.

**What are the popular tools in DevOps?**

* **Version Control:** Git, GitHub, GitLab
* **CI/CD:** Jenkins, GitLab CI, CircleCI, Travis CI
* **Configuration Management:** Ansible, Puppet, Chef
* **Containerization:** Docker, Kubernetes
* **IaC:** Terraform, AWS CloudFormation
* **Monitoring:** Prometheus, Grafana, Nagios
* **Collaboration:** Jira, Slack

**Explain the approach of DevOps**

* DevOps approach integrates **people, processes, and technology**.
* Core principles include:
  + **Continuous Integration:** Frequent code integration with automated testing.
  + **Continuous Delivery/Deployment:** Automated release pipelines to production.
  + **Infrastructure as Code:** Automate infrastructure provisioning.
  + **Monitoring & Feedback:** Continuous monitoring of applications and infrastructure to improve.
  + Encourages **collaboration** between development and operations teams.

**What challenges you faced in DevOps and how did you overcome them?**

* **Common challenges:**
  + Cultural resistance between Dev and Ops teams.
  + Complexity in automating legacy systems.
  + Managing secrets and sensitive data securely.
  + Handling environment inconsistencies.
* **How to overcome:**
  + Promote collaboration and shared goals.
  + Gradual adoption with pilot projects.
  + Use tools like Vault or Jenkins Credentials Plugin for secrets management.
  + Use containerization and IaC for consistent environments.

**What are CI/CD tools?**

* Tools that automate **Continuous Integration (CI)** and **Continuous Delivery/Deployment (CD)** pipelines.
* Examples: Jenkins, GitLab CI, CircleCI, Travis CI, Azure DevOps.
* They build, test, and deploy code automatically on code commit, improving software quality and speed.

**How to handle secrets?**

* Use **secret management tools** to store and manage sensitive data securely:
  + HashiCorp Vault, AWS Secrets Manager, Azure Key Vault.
* Jenkins Credentials Plugin for managing secrets in pipelines.
* Avoid hardcoding secrets in code or config files.
* Use environment variables or encrypted files for secret injection at runtime.

**Steps in Jenkins pipeline to trigger Kubernetes deployment**

1. **Checkout** source code from version control.
2. **Build** Docker image of the application.
3. **Push** Docker image to a container registry (Docker Hub, ECR).
4. Use **kubectl** (or Helm) commands within the pipeline to **deploy/update** the Kubernetes cluster.
5. Optionally, run **tests** and perform **rollback** on failure.

**What are the stages of Jenkins?**

* Jenkins pipeline stages typically include:
  + **Checkout:** Pull code from repository.
  + **Build:** Compile/build application.
  + **Test:** Run unit/integration tests.
  + **Package:** Create deployable artifacts (e.g., JAR, Docker image).
  + **Deploy:** Deploy to staging/production.
  + **Post:** Cleanup or notifications.

**Define Maven**

* Maven is a **build automation and dependency management tool** for Java projects.
* It uses a declarative XML file (pom.xml) to manage project build, dependencies, plugins, and lifecycle.
* Simplifies building, testing, and packaging Java applications consistently.

**Explain troubleshooting performance for production systems**

* Monitor key metrics: CPU, memory, disk I/O, network latency.
* Analyze logs and error messages to identify bottlenecks.
* Use profiling tools (e.g., JProfiler, New Relic).
* Check database queries for optimization.
* Scale resources or tune configurations as needed.
* Perform root cause analysis after incidents and improve system robustness.

Monitoring and Logging

**Use of Prometheus**

* Prometheus is an open-source **monitoring and alerting toolkit**.
* It **collects metrics** from applications and infrastructure using a pull model.
* Stores metrics as **time-series data**.
* Supports powerful queries with **PromQL**.
* Commonly used for monitoring system health, performance metrics, and setting up alerts.

**How to connect Grafana with Prometheus?**

1. Open Grafana UI.
2. Go to **Configuration → Data Sources**.
3. Click **Add data source**.
4. Select **Prometheus** from the list.
5. Enter Prometheus server URL (e.g., http://localhost:9090).
6. Click **Save & Test** to verify the connection.
7. Once connected, you can query Prometheus data in Grafana dashboards.

**Creation of dashboards in Grafana**

1. After connecting data source, click **+ → Dashboard**.
2. Click **Add new panel**.
3. Write a PromQL query to fetch the desired metric (e.g., node\_cpu\_seconds\_total).
4. Choose visualization type (graph, gauge, table, etc.).
5. Customize panel settings: legend, axes, thresholds.
6. Repeat to add more panels.
7. Save the dashboard with a meaningful name.

**SonarQube integration with Git and Jenkins**

* **Git:** Configure SonarQube project to analyze code from Git repositories.
* **Jenkins:**
  1. Install **SonarQube Scanner plugin** in Jenkins.
  2. Configure SonarQube server details in Jenkins global settings.
  3. In Jenkins pipeline/job, add **SonarQube Scanner** build step.
  4. Use commands like sonar-scanner or mvn sonar:sonar to trigger code analysis.
  5. Jenkins runs SonarQube analysis on every build, reporting code quality metrics.

**How can you check application logs, memory, and CPU utilization?**

* **Application logs:**
  + Access log files on the server (e.g., /var/log/app.log).
  + Use centralized logging tools like **ELK stack (Elasticsearch, Logstash, Kibana)** or **Splunk**.
* **Memory and CPU utilization:**
  + Use system commands: top, htop, free, vmstat on Linux.
  + Use monitoring tools like **Prometheus node exporter**, **Grafana dashboards**, **CloudWatch (AWS)**.
  + Containerized apps: Use kubectl top pods or metrics-server in Kubernetes.

Scripting and Configuration Management

**Playbook format (YAML or JSON)**

* Ansible **playbooks** are written primarily in **YAML** format because it is human-readable and easy to write.
* While Ansible can accept JSON syntax, YAML is preferred due to its simplicity and better support.
* Example YAML playbook snippet:

yaml

CopyEdit

- name: Install Apache

hosts: webservers

tasks:

- name: Ensure Apache is installed

apt:

name: apache2

state: present

**What is Ansible?**

* Ansible is an open-source **automation tool** used for configuration management, application deployment, and orchestration.
* It is **agentless**, using SSH to connect and execute commands on remote machines.
* Uses **playbooks** written in YAML to define desired system states.
* Known for simplicity, ease of use, and scalability.

**How to use Ansible Vault?**

* **Ansible Vault** is used to encrypt sensitive data such as passwords, keys, or secrets within playbooks.
* To create an encrypted file:

bash

CopyEdit

ansible-vault create secrets.yml

* To edit an existing vault file:

bash

CopyEdit

ansible-vault edit secrets.yml

* To run a playbook with encrypted files:

bash

CopyEdit

ansible-playbook site.yml --ask-vault-pass

* This ensures sensitive data is protected within your codebase.

**Integration process in Ansible**

* Ansible integrates with various systems using:
  + **Inventory files** to specify hosts.
  + **Modules** to perform tasks (e.g., file management, package installation).
  + **Plugins** for extending functionality.
  + **API integrations** to interact with cloud providers like AWS, Azure, or Docker.
* Integration often involves:
  + Defining connection parameters.
  + Writing playbooks that call modules or roles.
  + Using callbacks or handlers for advanced workflows.

**Explain bash scripting for automation**

* **Bash scripting** is writing scripts using the Bash shell to automate repetitive tasks.
* Common uses:
  + File and directory management.
  + Automating deployments and backups.
  + Scheduling jobs with cron.
* Example to automate a backup:

bash

CopyEdit

#!/bin/bash

tar -czf /backup/myapp\_$(date +%F).tar.gz /var/www/myapp

echo "Backup completed on $(date)" >> /var/log/backup.log

* Bash scripts can be integrated into CI/CD pipelines or called by Ansible for custom automation.

**Case Scenarios**

**Deploy microservices with RDS as a dependency**

* Deploy your microservices in containers or VMs.
* Configure microservices to connect to the RDS instance by providing the database endpoint, username, and password as environment variables or via secrets management.
* Ensure network connectivity/security groups allow microservices to communicate with RDS.
* Use infrastructure-as-code tools (Terraform/CloudFormation) to provision RDS and microservices infrastructure together to maintain consistency.

**Handle downtime in applications**

* Use **load balancers** to route traffic away from failing instances.
* Implement **health checks** to detect unhealthy services and restart or replace them automatically.
* Use **rolling deployments** or **blue-green deployments** to minimize downtime during updates.
* Monitor with tools like Prometheus and set up alerting.
* Maintain backups and disaster recovery plans.

**How to deploy a microservice with port as an environment variable?**

* Define the port in an environment variable in the container orchestration tool (e.g., Kubernetes deployment YAML):

yaml

CopyEdit

env:

- name: SERVICE\_PORT

value: "8080"

* Configure the microservice to read the port from the environment variable SERVICE\_PORT.
* Map the container port to the host or service accordingly.

**How to resolve a situation with two failed nodes out of five in Kubernetes?**

* Check node status using kubectl get nodes.
* Investigate node logs and system metrics to identify root cause.
* Drain the failed nodes: kubectl drain <node-name> to safely evict pods.
* Restart or replace the nodes depending on the cause (e.g., hardware failure, network issues).
* Use auto-scaling groups or node auto-repair if supported.
* Ensure pods are rescheduled to healthy nodes.

**If a pod restarts, how do you handle the issue?**

* Check pod logs with kubectl logs <pod-name> to identify errors.
* Inspect events and pod description: kubectl describe pod <pod-name>.
* Investigate readiness and liveness probes; misconfiguration can cause restarts.
* Check resource limits to avoid OOM kills or CPU throttling.
* Fix application bugs or configuration issues causing crashes.
* Consider increasing replica count for high availability.

**Ansible**

**About Ansible**

* Ansible is an agentless automation tool for configuration management, deployment, and orchestration.
* Uses SSH to connect to remote nodes.
* Uses YAML playbooks to define tasks declaratively.

**Ansible theory questions (how instances are attached to each other)**

* Ansible manages instances using an **inventory file**, grouping hosts logically.
* Tasks run on these hosts via SSH connections.
* Communication is one-way from control node to managed nodes.
* For orchestration, roles and playbooks define task sequences across hosts.

**Difference between Ansible and Terraform**

| **Aspect** | **Ansible** | **Terraform** |
| --- | --- | --- |
| Purpose | Configuration management, deployment automation | Infrastructure provisioning and management |
| Operation Mode | Push-based (control node initiates SSH connection) | Declarative; uses providers to interact with cloud APIs |
| State Management | Does not maintain persistent state (idempotent but stateless) | Maintains state file tracking resources |
| Language | YAML playbooks | HashiCorp Configuration Language (HCL) |
| Use Case | Configuring software, services | Provisioning servers, networks, cloud resources |

**Python/Shell Scripting**

**Write a shell script/python script to send an alert via email when disk usage exceeds 75%**

**Bash script example:**

bash

CopyEdit

#!/bin/bash

THRESHOLD=75

USAGE=$(df / | grep / | awk '{print $5}' | sed 's/%//')

if [ "$USAGE" -gt "$THRESHOLD" ]; then

echo "Disk usage is above $THRESHOLD%. Current usage: $USAGE%" | mail -s "Disk Usage Alert" your\_email@example.com

fi

**Python script example:**

python

CopyEdit

import shutil

import smtplib

from email.mime.text import MIMEText

threshold = 75

partition = "/"

usage = shutil.disk\_usage(partition)

percent\_used = usage.used / usage.total \* 100

if percent\_used > threshold:

msg = MIMEText(f"Disk usage is above {threshold}%. Current usage: {percent\_used:.2f}%")

msg['Subject'] = 'Disk Usage Alert'

msg['From'] = 'alert@example.com'

msg['To'] = 'your\_email@example.com'

with smtplib.SMTP('localhost') as server:

server.send\_message(msg)

**Write a bash script**

* Bash scripts automate repetitive Linux/Unix commands.
* Example: Backup a directory daily.

bash

CopyEdit

#!/bin/bash

tar -czf /backup/myapp\_$(date +%F).tar.gz /var/www/myapp

echo "Backup completed on $(date)" >> /var/log/backup.log

**Python features**

* Easy-to-learn, readable syntax.
* Extensive standard library and ecosystem.
* Supports multiple paradigms (procedural, OOP, functional).
* Dynamic typing and automatic memory management.
* Great for scripting, automation, data science, web development.

**What is a shell script?**

* A **shell script** is a text file containing a sequence of commands for a Unix shell (like Bash).
* It automates tasks, manages system operations, or runs batch jobs.
* Shell scripts can include conditionals, loops, functions, and variables.

**Maven**

**Maven lifecycle**

* Maven lifecycle defines a sequence of phases to build and deploy your project.
* Common lifecycles:
  + **default** (handles project deployment)
  + **clean** (cleans the project)
  + **site** (generates project documentation)
* Default lifecycle phases include:
  + validate → compile → test → package → verify → install → deploy

**What happens in mvn clean package command?**

* **clean**: Deletes the target/ directory, removing previous builds.
* **package**: Compiles the source code, runs tests, and packages the code into a distributable format (JAR, WAR).

**What is a Maven build profile?**

* Profiles allow customization of builds for different environments (dev, test, prod).
* Activate specific profiles via command line or POM to modify dependencies, plugins, or properties dynamically.

**POM stands for?**

* **Project Object Model** — the fundamental unit of work in Maven, defined in a pom.xml file that describes the project, dependencies, plugins, and build configurations.

**Networking**

**Process of searching something on the internet**

* User enters URL → browser sends DNS query → DNS resolves domain to IP → browser sends HTTP/HTTPS request → server responds with data → browser renders webpage.

**What is DNS and how does it work?**

* DNS (Domain Name System) translates human-readable domain names to IP addresses.
* When you enter a URL, your computer queries DNS servers to find the IP, enabling communication with the target server.

**Difference between DNS and CNAME**

* **DNS**: The entire system for resolving names to IPs.
* **CNAME**: A DNS record type that maps an alias name to the canonical (true) domain name.

**Difference between proxy, reverse proxy, and load balancer**

| **Term** | **Function** |
| --- | --- |
| Proxy | Acts as an intermediary for client requests to the internet |
| Reverse Proxy | Sits in front of servers and forwards client requests to them |
| Load Balancer | Distributes incoming network traffic across multiple servers |

**Port numbers**

* HTTP: 80
* HTTPS: 443
* DNS: 53
* MySQL: 3306
* FTP: 21

**Public vs. Private IP address**

* **Public IP**: Globally unique, reachable over the internet.
* **Private IP**: Used within local networks, not routable on the internet.

**How private IP is mapped to a public IP**

* Using **NAT (Network Address Translation)** on routers/firewalls, private IPs are translated to a public IP to communicate externally.

**Subnets (e.g., /16, /24, /32 notation)**

* CIDR notation defines the size of a subnet:
  + /16 → 65,536 IPs
  + /24 → 256 IPs
  + /32 → Single IP address

**Monitoring Tools**

**What are Prometheus and Grafana? How do they work?**

* **Prometheus** is an open-source monitoring and alerting toolkit that scrapes metrics from targets at intervals and stores them as time-series data.
* **Grafana** is a visualization tool that connects to Prometheus and other data sources to create dashboards and graphs for monitoring.

**Other Questions**

**Explain project details**

* Briefly describe the project’s objective, technologies used, your role, key challenges, and outcomes.

**Explain a complex problem or difficulty and its resolution**

* Share a specific incident, your analysis approach, steps taken to resolve, and results or lessons learned.

**Responsibilities in your current organization**

* List your daily tasks, tools you use, responsibilities in deployment, monitoring, troubleshooting, and collaboration.

**Knowledge of web applications**

* Web applications consist of frontend (UI), backend (server logic), and databases.
* Use HTTP/HTTPS protocols for communication, RESTful APIs, and frameworks like Spring Boot, React.

**HTTP vs. HTTPS**

* HTTPS is HTTP over TLS/SSL, providing encryption, authentication, and integrity, while HTTP is unsecured.

**HTTP request types (GET, POST, PUT)**

* **GET**: Retrieve data.
* **POST**: Submit data to create resources.
* **PUT**: Update existing resources.

**What is auto-scaling?**

* Auto-scaling automatically adjusts the number of active servers or containers based on load, ensuring availability and cost-efficiency.

**What is crontab?**

* A Linux utility to schedule jobs/scripts to run automatically at specified times or intervals.

**How to migrate app and DB from on-prem to cloud**

* Assess application and DB dependencies.
* Choose cloud provider and services.
* Backup DB and app data.
* Transfer data using tools like AWS DMS, or direct upload.
* Deploy app in cloud environment.
* Test and validate.
* Cutover traffic from on-prem to cloud gradually.