**DevOps Interview Preparation Answers**

**1. What are the different ways to trigger Jenkins pipelines?**

**Answer:**

* **Manually:** Click "Build Now" in the Jenkins UI.
* **SCM Polling:** Automatically triggers when there's a change in the Git repository.
* **Webhooks:** GitHub/GitLab notifies Jenkins on a new push.
* **Cron schedule:** Use triggers { cron('H/15 \* \* \* \*') } for periodic builds.

**Example:**

pipeline {

triggers {

cron('H/5 \* \* \* \*') // triggers every 5 minutes

}

}

**2. How to add a new worker node in Jenkins?**

**Answer:**

1. Go to **Manage Jenkins > Manage Nodes**.
2. Click **New Node** > name it > choose "Permanent Agent".
3. Define node properties: number of executors, labels, remote directory.
4. Launch agent using SSH or Java Web Start.

**3. How to add a new plugin in Jenkins?**

**Answer:**

1. Go to **Manage Jenkins > Plugin Manager**.
2. Under **Available**, search for the plugin.
3. Click **Install without restart** or **Download now and install after restart**.

**4. Where do you find errors in Jenkins?**

**Answer:**

* Console output of the failed job.
* Jenkins logs: /var/log/jenkins/jenkins.log.
* System log: **Manage Jenkins > System Log**.

**5. Difference between Docker and Ansible?**

| **Feature** | **Docker** | **Ansible** |
| --- | --- | --- |
| Purpose | Containerization | Configuration Management |
| Runs On | Container Engine | Python (agentless) |
| Use Case | Run apps in isolated units | Automate software provisioning |
| Example | docker run nginx | ansible-playbook deploy.yml |

**6. How to write a Dockerfile?**

**Answer:**

FROM node:18-alpine

WORKDIR /app

COPY . .

RUN npm install

CMD ["npm", "start"]

**7. Explanation about COPY and ADD in Dockerfile?**

**Answer:**

* **COPY:** Copies files/folders from host to container.
* **ADD:** Same as COPY but can also extract TAR files and supports URLs.

**Example:**

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

ADD archive.tar.gz /app/

**8. Explain about ENTRYPOINT in Dockerfile?**

**Answer:**

* ENTRYPOINT defines the **main command** to run when a container starts.
* Works with CMD to pass default arguments.

**Example:**

ENTRYPOINT ["python"]

CMD ["app.py"]

=> Runs python app.py

**9. Explain about Dockerfile structure?**

**Answer:**

1. **FROM** – base image.
2. **WORKDIR** – set working directory.
3. **COPY/ADD** – copy code/files.
4. **RUN** – install packages.
5. **CMD/ENTRYPOINT** – run command on container start.

**10. What is the purpose of the chmod command in Linux?**

**Answer:**  
chmod changes file permissions (read, write, execute) for user/group/others.

**Example:**

chmod 755 script.sh # rwxr-xr-x

**11. How do you change the permissions of a file to 777 using chmod?**

**Answer:**

chmod 777 filename.txt

This gives **read, write, and execute** permissions to everyone.

**12. What is SSH and how do you use it to connect to remote servers securely?**

**Answer:**  
SSH (Secure Shell) is used to **securely connect** to remote machines.

**Example:**

ssh user@192.168.1.10

**13. What is a public key and private key in the context of SSH?**

**Answer:**

* **Private Key:** Kept securely on your machine.
* **Public Key:** Shared with the server.
* Used for **password-less secure authentication**.

**Example:**

* Generate: ssh-keygen
* Copy public key to server: ssh-copy-id user@server

**14. How do you generate SSH key pairs using ssh-keygen?**

**Answer:**

ssh-keygen -t rsa -b 4096 -C "your\_email@example.com"

It generates public/private keys in ~/.ssh/id\_rsa and ~/.ssh/id\_rsa.pub.

**15. Common package managers used in Linux:**

* **Ubuntu/Debian:** apt
* **Amazon Linux/CentOS/RHEL:** yum, dnf

**Examples:**

sudo apt install nginx

sudo yum install nginx

**16. Describe your experience using Terraform to create AWS infrastructure**

**Answer:**  
Used Terraform to create:

* VPCs, subnets, EC2, S3, IAM roles
* Wrote reusable .tf modules
* Used terraform init, plan, apply to manage infra

**17. Key benefits of using Terraform:**

* **Infrastructure as Code (IaC)**
* **Reusable Modules**
* **Version Control**
* **Plan before apply**

**Project Benefit Example:** Rebuilt environments quickly and consistently across dev/test/prod.

**18. Two Ansible tasks example:**

- name: Install Apache

apt:

name: apache2

state: present

- name: Start Apache Service

service:

name: apache2

state: started

enabled: yes

**19. What are Ansible roles and their purpose?**

**Answer:**  
Roles help organize Ansible code into reusable units (defaults, tasks, handlers, etc.).

**Usage Example:** ansible-galaxy init myrole

**20. Functional uses of Ansible in your project:**

* Provisioning servers
* Installing packages
* Deploying applications
* Managing configurations

**Example:** Auto-deploying Spring Boot apps on EC2 using playbooks.

**DevOps Overview**

**1. What do you understand about DevOps?**

**DevOps is a combination of Development (Dev) and Operations (Ops).  
It’s a way of working where developers and operations teams work together to deliver software faster, with fewer bugs.**

**✅ Example:  
Before DevOps, developers wrote code and handed it over to the operations team to deploy. Now, in DevOps, both teams collaborate using automation tools like Git, Jenkins, Docker, etc.**

**2. What are the core principles of DevOps?**

**The main principles of DevOps are:**

1. **Collaboration – Dev and Ops work as one team**
2. **Automation – Automate builds, tests, and deployments**
3. **Continuous Integration and Delivery (CI/CD)**
4. **Monitoring and Feedback – Regularly check app performance**
5. **Culture of improvement – Learn and improve constantly**

**✅ Example:  
Using Jenkins to automatically build and test your code whenever you push to GitHub.**

**3. What is the difference between continuous delivery and deployment?**

* **Continuous Delivery: Code is always ready to be deployed, but needs manual approval to go live.**
* **Continuous Deployment: Code is automatically deployed to production after testing.**

**✅ Example:**

* **Continuous Delivery: The app is tested and waiting – someone has to click “Deploy”.**
* **Continuous Deployment: As soon as tests pass, the app goes live automatically.**

**4. What is CI/CD?**

**CI/CD stands for:**

* **Continuous Integration (CI): Automatically testing code when developers push changes.**
* **Continuous Delivery/Deployment (CD): Automatically delivering or deploying the tested code.**

**✅ Example:  
You push code to GitHub → Jenkins runs tests → Docker builds image → App is deployed to AWS.**

**5. How Agile and Waterfall Methodology differ from DevOps?**

| **Methodology** | **Description** | **Example** |
| --- | --- | --- |
| **Waterfall** | **Do everything step-by-step, one after the other (like a waterfall).** | **Finish design → then coding → then testing → then deploy.** |
| **Agile** | **Break work into small parts (sprints), with frequent releases.** | **Build a login page this week, feedback, then next feature.** |
| **DevOps** | **Focuses on automation, collaboration, and fast delivery. Works well with Agile.** | **Automatically test and deploy code after each sprint.** |

**AWS Overview**

**1. What is Cloud Computing?**

**Cloud Computing means using computers, storage, and services over the internet instead of buying and managing physical servers.**

**✅ Example:  
Instead of buying your own server, you can rent one from AWS (Amazon Web Services) to run your website or app.**

**2. What are AZs and Regions in AWS?**

* **Region: A physical area in the world (like US-East, Asia-Pacific).**
* **AZ (Availability Zone): A data center (or group of them) in a Region.  
  Each Region has 2 or more AZs, used for high availability.**

**✅ Example:  
Region = US East (N. Virginia)  
AZs = us-east-1a, us-east-1b, us-east-1c**

**So, if one AZ goes down, others still work.**

**3. What are the various types of EC2 instance types?**

**EC2 instances are virtual machines in AWS. There are different types based on need:**

| **Instance Type** | **Use Case** | **Example** |
| --- | --- | --- |
| **t2/t3 (General Purpose)** | **Small apps, websites** | **t2.micro (free tier)** |
| **m5 (Balanced)** | **Medium applications** | **m5.large** |
| **c5 (Compute optimized)** | **High CPU apps** | **Video processing** |
| **r5 (Memory optimized)** | **Big databases** | **r5.large** |
| **g4/g5 (GPU optimized)** | **Machine learning, graphics** | **AI training** |

**4. What is the importance of IAM in AWS?**

**IAM (Identity and Access Management) is used to securely control access to AWS services.**

**✅ Example:**

* **You can allow a developer to only access S3.**
* **Or give a tester read-only access to EC2.  
  IAM lets you manage users, roles, and permissions.**

**5. What do you understand about Load Balancers? What are their types?**

**A Load Balancer distributes traffic to multiple servers so no single server gets overloaded.**

**✅ Example:  
If 1,000 users visit your website, a load balancer splits the traffic to 2 or 3 EC2 instances.**

**Types of AWS Load Balancers:**

1. **Application Load Balancer (ALB) – Best for web apps (HTTP/HTTPS)**
   * **Example: Route based on URL path like /login to one server and /admin to another.**
2. **Network Load Balancer (NLB) – Best for high-speed, low-latency traffic (TCP)**
   * **Example: Gaming or chat apps**
3. **Classic Load Balancer (CLB) – Old type, used for basic needs (HTTP & TCP)**
   * **Not used much anymore.**

**AWS DevOps**

**1. Which service is DNS service in AWS?**

**Amazon Route 53 is the DNS (Domain Name System) service in AWS.  
It helps you connect your domain name (like example.com) to your AWS resources (like an EC2 or Load Balancer).**

**✅ Example:  
You can use Route 53 to point www.mywebsite.com to an EC2 instance.**

**2. What is AWS ECR?**

**ECR (Elastic Container Registry) is a service to store Docker container images in AWS.**

**✅ Example:  
You build a Docker image of your app → Push it to ECR → Pull and run it on an EC2 or ECS service.**

**3. How can you connect S3 with an EC2 Instance?**

**You can connect EC2 and S3 in two main steps:**

1. **Create an IAM role with S3 access (like read/write).**
2. **Attach the IAM role to your EC2 instance.**

**✅ Example:  
Once connected, your EC2 app can upload files to or download data from your S3 bucket directly using AWS CLI or SDK.**

**4. What is AWS CodeDeploy service?**

**CodeDeploy is a deployment service that automates code deployment to:**

* **EC2 instances**
* **Lambda functions**
* **On-premises servers**

**✅ Example:  
You can push code from GitHub or S3 → CodeDeploy automatically installs it on EC2 instances.**

**5. How can you create infrastructure using only AWS services?**

**You can use AWS CloudFormation or AWS CDK (Cloud Development Kit) to create infrastructure as code.**

**✅ Example (with CloudFormation):  
You write a YAML or JSON template to define resources (like EC2, VPC, S3) → Launch the stack → AWS creates all resources for you automatically.**

**Linux Fundamentals**

**1. Shell script to install Jenkins on Ubuntu**

**bash**

**CopyEdit**

**#!/bin/bash**

**# Update system**

**sudo apt update -y**

**# Install Java (Jenkins prerequisite)**

**sudo apt install openjdk-11-jdk -y**

**# Add Jenkins repo key and source list**

**curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \**

**/usr/share/keyrings/jenkins-keyring.asc > /dev/null**

**echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \**

**https://pkg.jenkins.io/debian-stable binary/ | sudo tee \**

**/etc/apt/sources.list.d/jenkins.list > /dev/null**

**# Update again**

**sudo apt update -y**

**# Install Jenkins**

**sudo apt install jenkins -y**

**# Start Jenkins service**

**sudo systemctl start jenkins**

**# Enable Jenkins to start on boot**

**sudo systemctl enable jenkins**

**# Show Jenkins status**

**sudo systemctl status jenkins**

**To execute:**

1. **Save this as install\_jenkins.sh**
2. **Run chmod +x install\_jenkins.sh**
3. **Run ./install\_jenkins.sh**

**2. How to send a file from local system to EC2 instance?**

**Use scp (secure copy):**

**bash**

**CopyEdit**

**scp -i /path/to/your-key.pem /path/to/local-file.txt ubuntu@ec2-public-ip:/home/ubuntu/**

* **-i specifies your private key**
* **ubuntu is the username for Ubuntu EC2**
* **ec2-public-ip is your instance's public IP**
* **/home/ubuntu/ is destination directory on EC2**

**3. How to create a user in Linux?**

**bash**

**CopyEdit**

**sudo adduser newusername**

**This command creates the user and asks you to set a password and some info.**

**4. What are package managers in Linux?**

**Package managers help install, update, and remove software easily.**

* **For Ubuntu/Debian: apt or apt-get**
* **For RedHat/CentOS: yum or dnf**
* **For Arch Linux: pacman**

**5. How to resolve permission denied errors while executing a file?**

**Usually, the file is not executable.**

**Run:**

**bash**

**CopyEdit**

**chmod +x filename.sh**

**Then run:**

**bash**

**CopyEdit**

**./filename.sh**

**Application Development**

**1. What do you understand about application development?**

**Application development is the process of planning, building, testing, and launching software apps for users to use on computers, phones, or web browsers.**

**✅ Example:  
Creating a mobile banking app, a food delivery website, or a desktop game is application development.**

**2. What is SDLC?**

**SDLC (Software Development Life Cycle) is a step-by-step process used to develop software.**

**🛠️ Phases in SDLC:**

1. **Requirement gathering**
2. **Design**
3. **Development (coding)**
4. **Testing**
5. **Deployment**
6. **Maintenance**

**✅ Example:  
Before building a ride-sharing app, the team gathers features (login, ride-booking), designs the app, develops it, tests it, deploys it, and keeps it updated.**

**3. What are microservices?**

**Microservices is a way of designing an application as a collection of small, independent services that work together.**

**✅ Example:  
An e-commerce site might have microservices for:**

* **User service**
* **Product service**
* **Order service**
* **Payment service**

**Each can be built, tested, and deployed separately.**

**4. Difference between compiled and interpreted languages?**

| **Type** | **Compiled Language** | **Interpreted Language** |
| --- | --- | --- |
| **How it runs** | **Code is converted to machine code before running** | **Code is run line-by-line by an interpreter** |
| **Speed** | **Faster** | **Slower** |
| **Example** | **C, C++, Java** | **Python, JavaScript, PHP** |

**✅ Example:**

* **C code → Compiled to a .exe file → Then executed**
* **Python → Runs directly using the Python interpreter**

**5. What are platform-independent languages?**

**A platform-independent language can run on any operating system (Windows, Linux, Mac) without changing the code.**

**✅ Example:**

* **Java is platform-independent. You write code once, and it runs on any system with a Java Virtual Machine (JVM).**

**"Write once, run anywhere."**

**Build tools**

**1. Name dependencies used in your project**

**Dependencies are libraries or tools that your project needs to work.  
Here are common ones used in a Java/Spring Boot project:**

**✅ Example dependencies:**

* **spring-boot-starter-web – for building REST APIs**
* **spring-boot-starter-data-jpa – for database access using JPA**
* **spring-boot-starter-security – for securing the app**
* **H2 or MySQL – database dependencies**
* **Lombok – to reduce boilerplate code**
* **JUnit – for testing**
* **Swagger – for API documentation**

**2. What do you understand about application building?**

**Application building is the process of compiling the code, resolving dependencies, and packaging it into a runnable format like .jar, .war, or .zip.**

**✅ Example:  
In a Spring Boot project, when you run mvn clean install, Maven compiles your code and creates a .jar file that can be run using java -jar.**

**3. What is the Maven build lifecycle?**

**Maven has a standard build lifecycle made up of phases:**

| **Phase** | **Purpose** |
| --- | --- |
| **validate** | **Check if the project is correct** |
| **compile** | **Compile the source code** |
| **test** | **Run unit tests** |
| **package** | **Package the code (e.g., into .jar)** |
| **verify** | **Check results of integration tests** |
| **install** | **Install the package to local repo** |
| **deploy** | **Deploy to remote repository (like Nexus)** |

**✅ Example command:  
mvn clean install runs: clean → compile → test → package → install**

**4. What are dependencies?**

**Dependencies are external libraries or tools your project needs to run or build correctly.**

**✅ Example:  
If you want to connect to a database in Java, you add a MySQL driver dependency in pom.xml.**

**xml**

**CopyEdit**

**<dependency>**

**<groupId>mysql</groupId>**

**<artifactId>mysql-connector-java</artifactId>**

**<version>8.0.33</version>**

**</dependency>**

**5. Which AWS service is used for application building?**

**AWS CodeBuild is the main service used for building applications in AWS.**

**✅ Example:  
CodeBuild can compile code, run tests, and package your app. It’s part of the CI/CD pipeline using:**

* **CodeCommit → Git repo**
* **CodeBuild → Build stage**
* **CodeDeploy → Deploy stage**

**Git and Github**

**1. What is SCM?**

**SCM (Source Code Management) is a system used to track changes in code, manage versions, and collaborate with other developers.**

**✅ Example:  
Git is a popular SCM tool. It lets you go back to old versions of your code or work with others without overwriting their changes.**

**2. What is the purpose of the staging area in Git?**

**The staging area (also called the index) is where Git stores files before you commit them.**

**✅ Example:  
You make changes in 3 files but only want to commit 1 — you can git add file1.java to stage that file, then git commit to save it.**

**3. Difference between git pull and git fetch**

| **Command** | **What it does** |
| --- | --- |
| **git fetch** | **Gets the latest changes from remote, but does not merge** |
| **git pull** | **Gets changes and merges them into your current branch** |

**✅ Example:**

* **Use git fetch when you want to check what's new before merging.**
* **Use git pull when you're ready to update your code with remote changes.**

**4. Difference between git reset --soft and git reset --mixed**

| **Command** | **Effect on Staging Area** | **Effect on Working Directory** |
| --- | --- | --- |
| **git reset --soft** | **Keeps files staged** | **Does not change your code** |
| **git reset --mixed** | **Unstages files** | **Code remains the same** |

**✅ Example:**

* **--soft: You keep your changes ready for commit.**
* **--mixed: You keep your code, but have to add files again before committing.**

**5. What is the purpose of commands like git rebase and git remote -v?**

* **git rebase:  
  Used to move or clean up commits by putting your changes on top of another branch.**

**✅ Example:  
If you created feature branch from main, you can run:  
git rebase main  
This puts your changes on top of the latest main.**

* **git remote -v:  
  Shows the URLs of the remote repositories (for fetch and push).**

**✅ Example:**

**bash**

**CopyEdit**

**git remote -v**

**Output:**

**perl**

**CopyEdit**

**origin https://github.com/user/project.git (fetch)**

**origin https://github.com/user/project.git (push)**

**JENKINS**

**1. What is Continuous Integration (CI)?**

**Continuous Integration (CI) is the practice of automatically building and testing code whenever developers push changes to a shared repository (like GitHub).**

**✅ Example:  
When a developer pushes code to GitHub, Jenkins automatically:**

* **Pulls the latest code**
* **Builds the app**
* **Runs tests  
  This helps catch bugs early and avoid "it works on my machine" problems.**

**2. Write a Jenkins pipeline script used in your project**

**Here’s a simple declarative pipeline used in many projects (especially with Maven and Docker):**

**groovy**

**CopyEdit**

**pipeline {**

**agent any**

**stages {**

**stage('Pull Code') {**

**steps {**

**git 'https://github.com/your-repo/project.git'**

**}**

**}**

**stage('Build') {**

**steps {**

**sh 'mvn clean package'**

**}**

**}**

**stage('Test') {**

**steps {**

**sh 'mvn test'**

**}**

**}**

**stage('Build Docker Image') {**

**steps {**

**sh 'docker build -t myapp:latest .'**

**}**

**}**

**stage('Push to Docker Hub') {**

**steps {**

**withCredentials([usernamePassword(credentialsId: 'dockerhub-creds', usernameVariable: 'USER', passwordVariable: 'PASS')]) {**

**sh 'echo $PASS | docker login -u $USER --password-stdin'**

**sh 'docker push myapp:latest'**

**}**

**}**

**}**

**}**

**}**

**3. How can you build Jenkins jobs periodically?**

**You can schedule builds using CRON syntax in the job configuration:**

**✅ Example:  
To run a job every day at midnight:**

**CopyEdit**

**0 0 \* \* \***

**💡 In Jenkins:**

* **Go to the job → Configure**
* **Check "Build periodically"**
* **Enter the CRON schedule there**

**4. What is the purpose of using Jenkins Slaves/Agents?**

**Jenkins Agents (Slaves) are machines that run builds for Jenkins Master. They help distribute the workload.**

**✅ Purpose:**

* **Run jobs on different OS (Windows/Linux)**
* **Parallel job execution**
* **Free up the Jenkins master**
* **Scale builds for large teams**

**✅ Example:  
If you have a heavy build job, you can run it on a high-memory agent machine while keeping the master free.**

**5. Demonstrate Jenkins installation using a shell script**

**bash**

**CopyEdit**

**#!/bin/bash**

**# Update system**

**sudo apt update -y**

**# Install Java (Jenkins needs it)**

**sudo apt install openjdk-11-jdk -y**

**# Add Jenkins repo key and source**

**curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \**

**/usr/share/keyrings/jenkins-keyring.asc > /dev/null**

**echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \**

**https://pkg.jenkins.io/debian-stable binary/ | sudo tee \**

**/etc/apt/sources.list.d/jenkins.list > /dev/null**

**# Update and install Jenkins**

**sudo apt update -y**

**sudo apt install jenkins -y**

**# Start and enable Jenkins**

**sudo systemctl start jenkins**

**sudo systemctl enable jenkins**

**# Show status**

**sudo systemctl status jenkins**

**✅ To run the script:**

**bash**

**CopyEdit**

**chmod +x install\_jenkins.sh**

**./install\_jenkins.sh**

**Docker**

**1. Container vs Virtual Machine (VM)**

| **Feature** | **Container** | **Virtual Machine (VM)** |
| --- | --- | --- |
| **Lightweight** | **Yes (shares host OS)** | **No (runs full OS per VM)** |
| **Boot Time** | **Seconds** | **Minutes** |
| **Resource Usage** | **Low** | **High** |
| **Use Case** | **Fast app deployment** | **Full OS with isolation** |

**✅ Example:**

* **Containers: Run a single app (e.g., a Java microservice).**
* **VMs: Set up an entire OS with multiple apps.**

**2. Docker Architecture**

**Docker has 3 main components:**

* **Docker Client: Runs docker commands like docker run**
* **Docker Daemon (dockerd): Runs in the background, manages images and containers**
* **Docker Registry: Stores Docker images (like Docker Hub)**

**✅ Flow:  
docker build → creates image → pushed to Docker Hub → pulled by daemon → run as a container**

**3. Write a Dockerfile to containerize a Java application**

**dockerfile**

**CopyEdit**

**# Use a base image with Java**

**FROM openjdk:17**

**# Set working directory**

**WORKDIR /app**

**# Copy jar file**

**COPY target/myapp.jar myapp.jar**

**# Run the application**

**CMD ["java", "-jar", "myapp.jar"]**

**✅ Save as Dockerfile, then build and run:**

**bash**

**CopyEdit**

**docker build -t my-java-app .**

**docker run -p 8080:8080 my-java-app**

**4. Command to create a container with port binding**

**bash**

**CopyEdit**

**docker run -d -p 8080:8080 my-java-app**

* **-d: run in background**
* **-p 8080:8080: map container port 8080 to host port 8080**

**5. What are Docker Volumes and Docker Compose?**

* **Docker Volumes: Used to store data outside the container — data remains even if the container is deleted.**

**✅ Example:**

**bash**

**CopyEdit**

**docker run -v myvolume:/app/data myapp**

* **Docker Compose: Tool to run multi-container apps using a docker-compose.yml file.**

**✅ Example docker-compose.yml:**

**yaml**

**CopyEdit**

**version: '3'**

**services:**

**web:**

**image: my-java-app**

**ports:**

**- "8080:8080"**

**db:**

**image: mysql**

**environment:**

**MYSQL\_ROOT\_PASSWORD: root**

**Then run:**

**bash**

**CopyEdit**

**docker-compose up**

**Kubernetes**

**1. What is Container Orchestration?**

**Container orchestration is the process of automatically managing containers—including starting, stopping, scaling, and networking them.**

**✅ Tools: Kubernetes, Docker Swarm**

**✅ Example:  
If your app has 10 containers and one crashes, Kubernetes will automatically restart it.**

**2. Components of K8s Control Plane**

**The Kubernetes Control Plane manages the entire cluster. Key components:**

| **Component** | **Purpose** |
| --- | --- |
| **kube-apiserver** | **Frontend to communicate with the cluster** |
| **etcd** | **Stores cluster data (key-value store)** |
| **kube-scheduler** | **Assigns Pods to Nodes** |
| **kube-controller-manager** | **Ensures desired state is maintained** |
| **cloud-controller-manager** | **Handles cloud-specific logic (e.g., AWS, GCP)** |

**3. Write a Deployment YAML file**

**yaml**

**CopyEdit**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: myapp-deployment**

**spec:**

**replicas: 2**

**selector:**

**matchLabels:**

**app: myapp**

**template:**

**metadata:**

**labels:**

**app: myapp**

**spec:**

**containers:**

**- name: myapp**

**image: mydockerhub/myapp:latest**

**ports:**

**- containerPort: 8080**

**✅ Command to apply:**

**bash**

**CopyEdit**

**kubectl apply -f deployment.yaml**

**4. Write a Service YAML to expose the Deployment to the internet**

**yaml**

**CopyEdit**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: myapp-service**

**spec:**

**type: LoadBalancer**

**selector:**

**app: myapp**

**ports:**

**- protocol: TCP**

**port: 80**

**targetPort: 8080**

**✅ Command to apply:**

**bash**

**CopyEdit**

**kubectl apply -f service.yaml**

**✅ Or command directly:**

**bash**

**CopyEdit**

**kubectl expose deployment myapp-deployment --type=LoadBalancer --port=80 --target-port=8080**

**5. Command to list Pods with images inside them**

**bash**

**CopyEdit**

**kubectl get pods -o wide**

**Or, to list Pods with container images:**

**bash**

**CopyEdit**

**kubectl get pods -o=jsonpath="{range .items[\*]}{.metadata.name} - {.spec.containers[\*].image}{'\n'}{end}"**

**✅ Output Example:**

**bash**

**CopyEdit**

**myapp-pod-abc123 - mydockerhub/myapp:latest**

**Ansible**

**1. What is Configuration Management?**

**Configuration management is the process of automating system setup (like installing software, managing files, etc.) to make sure servers are always in the correct and consistent state.**

**✅ Tools: Ansible, Puppet, Chef**

**✅ Example:  
You want to install Java and Docker on 10 servers. Instead of doing it manually, you use Ansible to do it automatically.**

**2. What are Ansible Roles?**

**Roles in Ansible are a way to organize playbooks into reusable parts (like folders).**

**Each role contains:**

* **tasks/ → what to do**
* **handlers/ → restart services**
* **files/ → files to copy**
* **templates/ → config files**
* **vars/ → variables**

**✅ Example:  
You can create a docker role and reuse it in different playbooks for multiple servers.**

**3. How to create a role?**

**Use the command:**

**bash**

**CopyEdit**

**ansible-galaxy init myrole**

**This creates a folder structure like:**

**python**

**CopyEdit**

**myrole/**

**├── tasks/**

**├── handlers/**

**├── templates/**

**├── files/**

**├── vars/**

**├── defaults/**

**└── meta/**

**Then, you can call this role from a playbook like:**

**yaml**

**CopyEdit**

**- hosts: all**

**roles:**

**- myrole**

**4. Write Ansible Playbook to install Java and Docker**

**yaml**

**CopyEdit**

**---**

**- name: Install Java and Docker**

**hosts: all**

**become: yes**

**tasks:**

**- name: Install Java**

**apt:**

**name: openjdk-11-jdk**

**state: present**

**update\_cache: yes**

**- name: Install Docker**

**apt:**

**name: docker.io**

**state: present**

**update\_cache: yes**

**- name: Enable and start Docker**

**systemd:**

**name: docker**

**enabled: yes**

**state: started**

**✅ Run this playbook with:**

**bash**

**CopyEdit**

**ansible-playbook install\_java\_docker.yml -i hosts**

**5. Ansible command to test connection with hosts**

**bash**

**CopyEdit**

**ansible all -m ping -i hosts**

**✅ Output:**

**javascript**

**CopyEdit**

**server1 | SUCCESS => {**

**"changed": false,**

**"ping": "pong"**

**}**

**Terraform**

**1. IaC vs IaaS**

| **Term** | **Full Form** | **Meaning** |
| --- | --- | --- |
| **IaC** | **Infrastructure as Code** | **Automating infrastructure using code (e.g., Terraform, Ansible)** |
| **IaaS** | **Infrastructure as a Service** | **Cloud service model that provides virtual servers (e.g., EC2 in AWS)** |

**✅ Example:**

* **IaC: Terraform script that launches EC2, VPC, S3**
* **IaaS: AWS giving you the ability to launch EC2 manually**

**2. Terraform Script to Provision EC2 on AWS**

**h**

**CopyEdit**

**provider "aws" {**

**region = "us-east-1"**

**}**

**resource "aws\_instance" "myec2" {**

**ami = "ami-0c02fb55956c7d316" # Amazon Linux 2**

**instance\_type = "t2.micro"**

**key\_name = "my-key" # Your EC2 key pair name**

**tags = {**

**Name = "MyFirstInstance"**

**}**

**}**

**✅ To run this script:**

**bash**

**CopyEdit**

**terraform init**

**terraform plan**

**terraform apply**

**3. Terraform plan vs validate**

| **Command** | **Purpose** |
| --- | --- |
| **terraform validate** | **Checks if the code syntax is correct** |
| **terraform plan** | **Shows what changes will be made before applying them** |

**✅ Example:**

* **validate says: “Your script looks okay.”**
* **plan says: “Here’s what I’ll create, change, or destroy.”**

**4. Purpose of Terraform State File (terraform.tfstate)**

* **Keeps track of the real state of your infrastructure**
* **Needed so Terraform knows what already exists**
* **Used to compare with the code and apply only needed changes**

**✅ Example:  
If you rename a resource in your .tf file, Terraform will check the state file to know whether to create a new one or update the existing one.**

**5. Useful Terraform Commands**

| **Command** | **Description** |
| --- | --- |
| **terraform init** | **Initializes the project (downloads providers)** |
| **terraform plan** | **Shows changes before applying** |
| **terraform apply** | **Applies changes to the cloud** |
| **terraform destroy** | **Destroys all resources created** |
| **terraform validate** | **Validates configuration syntax** |
| **terraform fmt** | **Formats .tf files properly** |
| **terraform show** | **Displays current state from .tfstate** |

**Monitoring**

**1. Why do we need to monitor our servers?**

**Monitoring helps to:**

* **📈 Track performance (CPU, memory, disk usage)**
* **🚨 Get alerts when something goes wrong**
* **📊 Visualize health of applications and systems**
* **🔍 Diagnose issues before users complain**

**✅ Example:  
If CPU hits 90% or a service goes down, you get an alert immediately.**

**2. What are Prometheus Exporters?**

**Prometheus exporters are small tools that collect and expose system/app metrics in a format Prometheus understands.**

**✅ Popular exporters:**

* **node\_exporter – for server CPU, RAM, disk, etc.**
* **cadvisor – for Docker/container metrics**
* **blackbox\_exporter – for monitoring external endpoints**

**✅ Example:  
node\_exporter exposes data on http://<server>:9100/metrics**

**3. How to Set Up Alerts in Grafana**

**To create alerts in Grafana:**

1. **🧭 Go to the panel you want to monitor.**
2. **Click the "Alert" tab → Create Alert Rule.**
3. **Define:**
   * **Condition (e.g., if CPU > 80%)**
   * **Time interval (how often to check)**
   * **Notification channel (email, Slack, etc.)**
4. **Click "Save and Apply".**

**✅ Make sure Prometheus is the data source.**

**4. Default Port Numbers**

| **Tool** | **Port** |
| --- | --- |
| **Prometheus** | **9090** |
| **Grafana** | **3000** |
| **Node Exporter** | **9100** |
| **cAdvisor** | **8080** |

**5. Installation of Prometheus (on Ubuntu)**

**bash**

**CopyEdit**

**# Step 1: Create user**

**sudo useradd --no-create-home --shell /bin/false prometheus**

**# Step 2: Download Prometheus**

**wget https://github.com/prometheus/prometheus/releases/download/v2.52.0/prometheus-2.52.0.linux-amd64.tar.gz**

**# Step 3: Extract and move files**

**tar xvf prometheus-2.52.0.linux-amd64.tar.gz**

**cd prometheus-2.52.0.linux-amd64**

**sudo mv prometheus /usr/local/bin/**

**sudo mv promtool /usr/local/bin/**

**sudo mkdir /etc/prometheus**

**sudo cp -r consoles/ console\_libraries/ /etc/prometheus/**

**sudo cp prometheus.yml /etc/prometheus/**

**# Step 4: Create systemd service**

**sudo nano /etc/systemd/system/prometheus.service**

**Paste this:**

**ini**

**CopyEdit**

**[Unit]**

**Description=Prometheus**

**After=network.target**

**[Service]**

**User=prometheus**

**ExecStart=/usr/local/bin/prometheus \**

**--config.file=/etc/prometheus/prometheus.yml \**

**--storage.tsdb.path=/var/lib/prometheus/**

**[Install]**

**WantedBy=default.target**

**bash**

**CopyEdit**

**# Step 5: Start Prometheus**

**sudo systemctl daemon-reexec**

**sudo systemctl start prometheus**

**sudo systemctl enable prometheus**

**Then open Prometheus on:  
👉 http://localhost:9090**

**Project**

**1. Flow of the Project and Tools Used**

* **Flow:**
  1. **Code Commit → Developers push code to Git (source control).**
  2. **Build → Jenkins picks the code, runs Maven build, and creates Docker images.**
  3. **Test → Automated tests (unit/UI) run via Selenium.**
  4. **Deploy → Docker containers are deployed to AWS EC2 using Ansible and Terraform.**
  5. **Monitor → Prometheus and Grafana monitor app health and performance.**
* **Tools:**
  1. **Git, GitHub (source control)**
  2. **Jenkins (CI/CD)**
  3. **Maven (build tool)**
  4. **Docker (containerization)**
  5. **Ansible & Terraform (infrastructure automation)**
  6. **Selenium (automated testing)**
  7. **Prometheus & Grafana (monitoring)**

**2. How the Same Project Can Be Done Using Only AWS DevOps Services**

* **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)**

**3. Security Measures to Secure Pipeline and Application**

* **Use IAM roles and policies with least privilege for all AWS services.**
* **Use encrypted secrets and credentials in Jenkins (or AWS Secrets Manager).**
* **Enable multi-factor authentication (MFA) for users.**
* **Use code scanning tools (like SonarQube) to catch vulnerabilities early.**
* **Secure network using VPC, security groups, and firewall rules.**
* **Scan Docker images for vulnerabilities.**
* **Enable audit logging (CloudTrail, Jenkins logs).**
* **Use HTTPS/TLS for all communication.**

**4. Problems Solved by This Project**

* **Manual deployments were error-prone and slow — automated with CI/CD pipeline.**
* **Inconsistent environments — solved with containerization and infrastructure automation.**
* **Lack of monitoring — now can detect and fix issues quickly with Prometheus/Grafana.**
* **Scaling challenges — containers and AWS help scale easily.**
* **Test automation — reduced bugs in production.**

**5. Files and Scripts Candidates Might Write in Projects**

* **Shell script to install Jenkins or other tools**
* **Dockerfile to containerize the app**
* **Jenkinsfile (Pipeline script) for CI/CD pipeline**
* **Terraform scripts to provision infrastructure (EC2, VPC, etc.)**
* **Ansible playbooks/roles to configure servers**
* **Kubernetes YAML files for deployment and service**
* **Shell/Ansible scripts to deploy application**
* **Selenium test scripts for UI testing**

**AWS Cloud**

1. **How is Availability Zone different than a Region in AWS?**

* **Region: A physical location worldwide with multiple data centers.**
* **Availability Zone (AZ): One or more isolated data centers within a Region.**

**✅ *Example:*  
US East (N. Virginia) is a Region, and it has multiple AZs like us-east-1a, us-east-1b.**

1. **Which AWS service lets you set up who can do what with your resources?**

* **AWS IAM (Identity and Access Management) lets you create users, groups, and roles with specific permissions.**

1. **How is S3 Standard different from S3 Glacier?**

| **Feature** | **S3 Standard** | **S3 Glacier** |
| --- | --- | --- |
| **Use case** | **Frequently accessed data** | **Long-term archive, infrequent access** |
| **Access speed** | **Milliseconds** | **Minutes to hours** |
| **Cost** | **Higher** | **Much cheaper** |

1. **If you want to give an EC2 instance access to just one S3 bucket and not others, which AWS service would you use?**

* **Use IAM Role attached to the EC2 instance with a policy that restricts access to only that specific S3 bucket.**

1. **How to give a new hire temporary AWS access that automatically ends?**

* **Create an IAM Role with temporary credentials using AWS STS (Security Token Service) or use IAM user with permissions and set up an expiration policy.**

1. **What is an Elastic IP Address?**

* **A static, public IPv4 address in AWS that you can assign to EC2 instances — remains the same even if you stop/start the instance.**

**DevOps Overview**

1. **What is Waterfall methodology and its disadvantages?**

* **Waterfall is a linear, sequential software development process (Requirements → Design → Implementation → Testing → Deployment).**
* **Disadvantages:**
  + **No flexibility to change requirements once started**
  + **Late testing means bugs found late**
  + **Slow feedback cycle**
  + **Difficult to adapt to change**

1. **What is Feedback Loop in DevOps?**

* **A continuous cycle of collecting and acting on feedback from development, testing, deployment, and users to improve software quickly and reliably.**

1. **How Agile fixed Waterfall’s issues?**

* **Agile uses iterative development with small cycles (sprints), allowing continuous feedback and changes.**
* **Encourages collaboration and flexibility, so changes can be made any time.**

1. **Why do we need Continuous Monitoring?**

* **To detect and fix issues early**
* **Track performance, security, and availability**
* **Provide feedback for continuous improvement**

1. **Difference between Continuous Delivery and Continuous Deployment**

| **Term** | **Meaning** |
| --- | --- |
| **Continuous Delivery** | **Code is automatically tested and ready for deployment; requires manual approval to deploy.** |
| **Continuous Deployment** | **Code changes are automatically deployed to production without manual intervention.** |

**1. Which command in Linux would you use to change the permissions of a file?**

* **chmod  
  Example:**

**bash**

**CopyEdit**

**chmod 755 filename**

**2. How do you create a new user in Linux?**

* **useradd or adduser  
  Example:**

**bash**

**CopyEdit**

**sudo useradd newusername**

**or**

**bash**

**CopyEdit**

**sudo adduser newusername**

**3. If you want to connect securely to a remote Linux server, which command would you use?**

* **ssh (Secure Shell)  
  Example:**

**bash**

**CopyEdit**

**ssh user@remote-server-ip**

**4. How would you edit a file in the terminal using a text editor that comes pre-installed on most Linux distributions?**

* **Use vi or vim  
  Example:**

**bash**

**CopyEdit**

**vi filename**

**5. If you need to automate a repetitive task in Linux, what would you write to achieve that?**

* **Write a shell script (a .sh file with commands)  
  Example:**

**bash**

**CopyEdit**

**#!/bin/bash**

**# script.sh**

**echo "Hello, this is a repetitive task"**

**Then run it with:**

**bash**

**CopyEdit**

**bash script.sh**

**6. What is the use of grep command?**

* **grep is used to search for specific text or patterns inside files or command output.  
  Example:**

**bash**

**CopyEdit**

**grep "error" logfile.txt**

**This finds all lines containing "error" in logfile.txt.**

1. **Which AWS service is equivalent to an SCM tool?**

* **AWS CodeCommit — It is a managed Git-based source control service to store your code.**

1. **What is CodeBuild?**

* **AWS CodeBuild is a fully managed build service that compiles your source code, runs tests, and produces ready-to-deploy software packages.**

1. **What is CodeDeploy?**

* **AWS CodeDeploy automates application deployment to EC2 instances, Lambda functions, or on-premises servers, helping avoid downtime during updates.**

1. **What is CodePipeline?**

* **AWS CodePipeline is a CI/CD service that automates the steps of building, testing, and deploying your application whenever code changes.**

1. **What is CloudFormation?**

* **AWS CloudFormation lets you create and manage AWS resources using code (templates), so you can automate infrastructure setup (Infrastructure as Code).**

**Git**

**1. Main difference between Central and Distributed Version Control Systems**

* **Centralized VCS (like SVN):  
  All users work with a single central repository. You need to be connected to the server to commit or update code.**
* **Distributed VCS (like Git):  
  Every user has a full copy of the entire repository history on their local machine. You can commit and work offline, then sync later.**

**2. Command to check the status of files in Git repository**

**bash**

**CopyEdit**

**git status**

**This shows which files are modified, staged, or untracked.**

**3. How to set up Git on your local machine for the first time**

**Set your name and email (used for commit info):**

**bash**

**CopyEdit**

**git config --global user.name "Your Name"**

**git config --global user.email "you@example.com"**

**4. Command to create a new branch in Git**

**bash**

**CopyEdit**

**git branch new-branch-name**

**To create and switch to the new branch at the same time:**

**bash**

**CopyEdit**

**git checkout -b new-branch-name**

**5. How to combine changes from one branch into another**

* **First switch to the branch where you want changes (e.g., main):**

**bash**

**CopyEdit**

**git checkout main**

* **Then merge the other branch (e.g., feature):**

**bash**

**CopyEdit**

**git merge feature**

**6. Purpose of git rebase and how it differs from git merge**

* **git merge combines histories and creates a merge commit, keeping the full branch history.**
* **git rebase moves your commits to the tip of another branch, creating a clean, linear history without merge commits.**

**Example:**

**bash**

**CopyEdit**

**git checkout feature**

**git rebase main**

**This applies your feature branch commits on top of main.**

**7. How to temporarily save changes you’re not ready to commit**

**bash**

**CopyEdit**

**git stash**

**It saves your current changes and cleans your working directory. Later you can apply them back with:**

**bash**

**CopyEdit**

**git stash apply**

**8. Difference between git reset and git revert**

* **git reset moves the HEAD pointer and can remove commits or unstage files. It changes history and is not safe if changes are shared.**
* **git revert creates a new commit that undoes the changes of a previous commit, preserving history and safe for shared repos.**

**JENKINS**

1. **What is a Jenkinsfile?**
   * **A text file that defines your build pipeline as code, written in Groovy syntax. It tells Jenkins how to build, test, and deploy your project.**
2. **Explain Jenkins master-slave architecture and purpose**
   * **Master controls the Jenkins server, UI, scheduling jobs.**
   * **Slaves (agents) execute build jobs. This setup allows distributing workload and scaling builds across machines.**
3. **How do you manage credentials in Jenkins?**
   * **Use Jenkins Credentials Plugin to securely store secrets (passwords, API tokens) and inject them into jobs without exposing them.**
4. **Purpose of post action in Jenkins pipeline?**
   * **Actions that run after the main pipeline (e.g., cleanup, sending notifications on success or failure).**
5. **Difference between scripted and declarative pipelines?**
   * **Scripted: Groovy code, more flexible but complex.**
   * **Declarative: Simple, structured, easier to write and maintain.**
6. **How is Jenkins executor different from Jenkins node?**
   * **Node: a machine (master or agent) where jobs run.**
   * **Executor: a slot on that node for running one job at a time.**
7. **How to trigger builds in Jenkins?**
   * **Manually, on schedule (cron), via SCM changes (polling or webhooks), or after other builds.**
8. **Setting up email notifications in Jenkins**
   * **Install Email Extension Plugin, configure SMTP settings, then add a post step in Jenkinsfile or job to send emails on build events.**

**DOCKER**

1. **How to bind ports in Docker for external access?**

**bash**

**CopyEdit**

**docker run -p 8080:80 image-name**

**Maps container’s port 80 to host’s port 8080.**

1. **Purpose of Dockerfile?**
   * **Defines instructions to build a Docker image (e.g., base image, copying files, running commands).**
2. **How to clean unused Docker images?**

**bash**

**CopyEdit**

**docker image prune -a**

**Removes dangling and unused images to free disk space.**

1. **Benefits of Docker Compose?**
   * **Manage multiple containers as a single app, define services, networks, volumes in one YAML file, easy startup and scaling.**
2. **Container not starting due to missing config file — how to fix?**
   * **Check logs: docker logs container\_id**
   * **Verify volumes and environment variables are correctly set to provide config file.**
3. **Difference between ARG and ENV in Dockerfile?**
   * **ARG: Build-time variable (used only during image build).**
   * **ENV: Environment variable available during build and runtime.**
4. **How to get shell access to a running container?**

**bash**

**CopyEdit**

**docker exec -it container\_id /bin/bash**

**or /bin/sh if bash is unavailable.**

**KUBERNETES (K8s)**

1. **Main components of Kubernetes architecture and roles**
   * **Master (Control Plane): API Server, Scheduler, Controller Manager, etcd (stores cluster state).**
   * **Nodes: Run pods with containerized apps. Each has Kubelet and kube-proxy.**
2. **What is a Pod?**
   * **Smallest deployable unit in K8s; one or more containers running together sharing network/storage.**
3. **Difference between ClusterIP, NodePort, and LoadBalancer services**
   * **ClusterIP: Internal-only IP within the cluster.**
   * **NodePort: Exposes service on a static port on each node’s IP.**
   * **LoadBalancer: Provisions cloud load balancer to expose service externally.**
4. **Methods to scale applications in Kubernetes**
   * **Manually update replica count with:**

**bash**

**CopyEdit**

**kubectl scale deployment my-app --replicas=5**

* + **Use Horizontal Pod Autoscaler (HPA) to scale based on CPU/memory.**

1. **Troubleshooting pod not starting due to resource errors**
   * **Check pod events:**

**bash**

**CopyEdit**

**kubectl describe pod pod-name**

* + **Adjust resource requests/limits in pod spec if insufficient.**

1. **How to inspect logs of a pod?**

**bash**

**CopyEdit**

**kubectl logs pod-name**

**Add -c container-name if multiple containers exist in pod.**

**Terraform**

1. **What is a state file?**
   * **A file (terraform.tfstate) that keeps track of the resources Terraform has created in the cloud. It helps Terraform know what’s already deployed and what needs to change.**
2. **Difference between terraform plan and terraform apply:**
   * **terraform plan: Shows what changes will happen (no actual changes).**
   * **terraform apply: Executes the planned changes and creates/updates resources.**
3. **How to delete a resource in Terraform?**
   * **Remove the resource block from your .tf files and run:**

**bash**

**CopyEdit**

**terraform apply**

1. **Terraform will destroy that resource.**
2. **What is a Terraform provider?**
   * **A plugin that lets Terraform interact with cloud services (like AWS, Azure). It knows how to create, update, delete resources in that platform.**
3. **Terraform modules you have worked with?**
   * **Modules are reusable sets of Terraform code. For example, you can have a module for EC2 instances or VPC setup, so you can reuse it across projects.**

**Ansible**

1. **Steps to install and configure Ansible on a new server:**
   * **Install Ansible using package manager:**

**bash**

**CopyEdit**

**sudo apt install ansible # on Ubuntu**

**sudo yum install ansible # on CentOS**

* + **Set up an inventory file listing your servers.**
  + **Test connection with ansible all -m ping.**

1. **Examples of Ansible modules you have used:**
   * **yum or apt to install packages.**
   * **copy to copy files.**
   * **service to start/stop services.**
   * **command or shell to run shell commands.**
2. **What is an inventory file?**
   * **A file listing hosts (servers) that Ansible manages. It can group servers for targeted playbooks.**
3. **Troubleshooting playbook syntax errors and ensuring task order:**
   * **Run playbook with ansible-playbook --syntax-check to find syntax errors.**
   * **Use indentation carefully.**
   * **Tasks are executed in the order written by default. Use when clauses to run tasks conditionally only if previous tasks succeed.**
4. **What are Ansible roles and how do they help?**
   * **Roles organize tasks, files, handlers, variables into reusable units. They make playbooks clean, modular, and easier to maintain.**

**Prometheus & Grafana**

1. **What is node exporter in Prometheus?**
   * **It is an agent that runs on Linux servers to collect hardware and OS metrics (CPU, memory, disk) for Prometheus to scrape.**
2. **Why do we need Alertmanager in Prometheus?**
   * **It manages alerts sent by Prometheus, groups them, silences notifications, and sends alerts to email, Slack, etc.**
3. **Which query language is used by Prometheus?**
   * **PromQL (Prometheus Query Language).**
4. **Troubleshooting Prometheus not collecting data:**
   * **Check Prometheus config file (prometheus.yml) for scrape configs and targets.**
   * **Use promtool check config prometheus.yml to validate config.**
   * **Check Prometheus logs for errors.**
   * **Make sure target exporters (e.g., node exporter) are running and accessible.**
5. **How to install and configure Grafana with Prometheus data source:**
   * **Install Grafana using package manager or Docker.**
   * **In Grafana UI, add a new data source, select Prometheus, and provide Prometheus server URL.**
   * **Save and test connection.**
6. **How to create and customize dashboards in Grafana:**
   * **Use Grafana UI to create a new dashboard.**
   * **Add panels with queries written in PromQL to visualize metrics.**
   * **Customize graphs, colors, time range, alerts.**