



DEVOPS INTERVIEW – 150+

1 What is DevOps?

Answer:

DevOps is a **culture + practice + set of tools** that brings **Development** and **Operations** teams together to **deliver software faster, safer, and continuously**.

Before DevOps:

- Developers write code
- Operations deploy manually
- If production fails → blame game
- Releases take weeks or months

With DevOps:

- Same team mindset
- Automated build, test, deploy
- Continuous monitoring & feedback
- Releases can happen **multiple times a day**

Real-time example:

When a developer pushes code to Git:

1. Jenkins automatically builds it
2. Tests are executed
3. Docker image is created
4. Kubernetes deploys it

5. Monitoring checks health

👉 This full automation is DevOps

2 Why DevOps is required?

Answer:

DevOps is required to **solve problems of traditional software delivery.**

Traditional problems:

- Slow deployments
- Manual errors
- Poor collaboration
- Late bug detection

DevOps solves by:

- Automating deployments
- Early testing (CI)
- Continuous feedback
- Infrastructure as Code

Interview line (important):

“DevOps reduces time-to-market while improving reliability.”

3 What are the core principles of DevOps?

Answer:

DevOps works on **5 main principles:**


1. **Collaboration** – Dev & Ops work together

2. **Automation** – CI/CD, Infra automation
 3. **Continuous Improvement** – monitoring & feedback
 4. **Customer-centric action** – faster bug fixes
 5. **Responsibility ownership** – You build it, you run it
-

4 Explain DevOps lifecycle in detail

DevOps Lifecycle Stages:

1. **Plan** – Requirement analysis (Jira)
2. **Code** – Developers write code (Git)
3. **Build** – Compile code (Maven)
4. **Test** – Automated testing (JUnit)
5. **Release** – Versioning
6. **Deploy** – Jenkins + Kubernetes
7. **Operate** – Server management
8. **Monitor** – Prometheus, Grafana

 Continuous feedback loop

5 What is CI (Continuous Integration)?

Answer:

CI means **automatically integrating code changes multiple times a day**.

What happens in CI?

- Developer pushes code
- Jenkins triggers build

- Unit tests run
- Errors detected early

Benefits:

- Early bug detection
 - No “integration hell”
 - Stable codebase
-

6 What is CD (Continuous Delivery vs Deployment)?

Continuous Delivery:

- Code is **ready for production**
- Manual approval needed

Continuous Deployment:

- Code **automatically goes to production**
- No manual intervention

Example:

Banking apps → Delivery
E-commerce apps → Deployment

7 Difference between DevOps & Agile (Detailed)

Agile	DevOps
Focus on development	Focus on deployment
Sprint based	Continuous

Ends at release

Ends at monitoring

👉 Agile + DevOps = Full SDLC automation

8 What is Infrastructure as Code (IaC)?

Answer:

Managing infrastructure using **code instead of manual configuration**.

Tools:

- Terraform
- CloudFormation
- ARM Templates

Benefits:

- Version control
- Repeatable infra
- No human error

Example:

```
resource "aws_instance" "server" {  
  ami = "ami-123"  
  instance_type = "t2.micro"  
}
```

9 What are DevOps tools categories?

1. Version Control – Git
2. CI/CD – Jenkins
3. Containers – Docker

4. Orchestration – Kubernetes
 5. IaC – Terraform
 6. Monitoring – Prometheus
-

10 What is Version Control?

Answer:

System that tracks **who changed what, when, and why**.

Benefits:

- Rollback
 - Collaboration
 - Audit trail
-

11 Git vs GitHub

Git	GitHub
Tool	Platform
Local	Cloud
Version control	Hosting

12 What is Automation in DevOps?

Answer:

Replacing **manual repetitive tasks** with scripts/tools.

Automated tasks:

- Build
- Test

- Deploy
 - Server provisioning
-

13 What is Monitoring in DevOps?

Answer:

Continuous observation of:

- CPU
- Memory
- Disk
- Application health

Tools:

- Prometheus
 - Grafana
 - CloudWatch
-

14 What is Blue-Green Deployment?

Answer:

Two identical environments:

- Blue → Old version
- Green → New version

Switch traffic after validation → **Zero downtime**

15 What is Canary Deployment?

Answer:

Deploy new version to **small % of users first**.

If stable → rollout to all users.

16 What is Rollback?

Answer:

Reverting to **previous stable version** when deployment fails.

17 What is SRE vs DevOps?

DevOps = Culture

SRE = Engineering implementation using SLAs/SLOs

18 What is SLA, SLO, SLI?

- **SLA** – Contract promise
 - **SLO** – Target reliability
 - **SLI** – Actual measurement
-

19 What is Mean Time To Recovery (MTTR)?

Time taken to recover from failure.

Lower MTTR = Better DevOps maturity

20 What is Configuration Management?

Managing system state using tools like:

- Ansible

- Puppet
 - Chef
-

21 DevOps Engineer daily responsibilities?

- CI/CD pipeline
 - Cloud infra
 - Automation scripts
 - Monitoring
 - Incident handling
-

22 What is Immutable Infrastructure?

Servers are **replaced**, not modified.

23 What is Shift-Left testing?

Testing early in development.

24 What is DevSecOps?

Security integrated into DevOps pipeline.

25 Why DevOps engineer is important?

Because modern applications need:

- Speed

- Reliability
- Automation
- Scalability

26 What is Linux? (Deep Explanation)

Answer:

Linux is an **open-source, multi-user, multi-tasking operating system** widely used in **servers, cloud, containers, and DevOps environments**.

Why Linux is preferred in DevOps?

- Stable and secure
- Command-line automation
- Lightweight
- Free & open source
- Strong community support

Real-time example:

AWS EC2 instances mostly run **Amazon Linux / Ubuntu / RHEL**.

27 Difference between Linux and Unix?

Linux	Unix
Open source	Mostly paid
Community supported	Vendor supported
Flexible	Less flexible
Widely used in cloud	Limited

Interview tip:

“Linux is Unix-like but open-source.”

28) What is Kernel?

Answer:

Kernel is the **core of Linux OS**.

Kernel responsibilities:

- Process management
- Memory management
- Hardware communication
- File system control

Types of kernel:

- Monolithic (Linux)
- Microkernel

29) What are Linux distributions?

Answer:

Linux distributions = Kernel + Utilities + Package manager.

Common distros:

- Ubuntu
- RedHat
- CentOS
- Amazon Linux

30) Linux directory structure (VERY IMPORTANT)

Directory	Purpose
<code>/etc</code>	Config files
<code>/var</code>	Logs & variable data
<code>/home</code>	User files
<code>/root</code>	Root user
<code>/bin</code>	Commands
<code>/usr</code>	Software
<code>/tmp</code>	Temporary files

31 What is Root user?

Answer:

Root user is **superuser** with **full system access**.

Why root is dangerous?

- One wrong command can delete OS

Best practice:

Use `sudo` instead of root login.

32 What is sudo?

Answer:

`sudo` allows **temporary admin access**.

```
sudo yum install nginx
```

Benefit:

- Security
- Audit trail

33 Difference between su and sudo?

su	sudo
Switch user	Run single command
Requires root password	User password
Less secure	More secure

34 How to check CPU usage?

top
htop

Production scenario:

If application is slow → check CPU spike.

35 How to check memory usage?

free -m

Important fields:

- total
 - used
 - available
-

36 How to check disk usage?

df -h

Disk space per directory:

```
du -sh /var/*
```

37 What is a process?

Answer:

A process is a **running program**.

Process states:

- Running
 - Sleeping
 - Zombie
-

38 How to list processes?

```
ps -ef
```

39 How to kill a process?

```
kill PID
```

```
kill -9 PID
```

Difference:

- `kill` → graceful
 - `kill -9` → force
-

40 What is systemd?

Answer:

`systemd` is **service manager** in modern Linux.

Why important?

- Faster boot
 - Dependency handling
-

41 How to manage services?

```
systemctl start nginx
systemctl stop nginx
systemctl restart nginx
systemctl status nginx
```

42 What is a daemon?

Answer:

Background service (nginx, sshd).

43 What is Cron Job?

Answer:

Cron is used to **schedule tasks automatically**.

Example:

Backup script daily at 2 AM.

44 Cron syntax (IMPORTANT)

```
* * * * * command
```

Field	Meaning
-------	---------

- 1 Minute
 - 2 Hour
 - 3 Day
 - 4 Month
 - 5 Weekday
-

45 Run cron every 2 minutes?

`*/2 * * * * command`

46 What is log file?

Answer:

Logs record system/application activity.

Common logs:

- `/var/log/messages`
 - `/var/log/secure`
 - `/var/log/syslog`
-

47 What is swap memory?

Answer:

Swap is disk space used as RAM backup.

When used?

When RAM is full.

48 What is inode?

Answer:

Inode stores **file metadata**, not content.

Includes:

- Size
 - Permissions
 - Owner
-

49 Difference between soft link and hard link?

Soft Link	Hard Link
Pointer	Same inode
Can cross FS	Same FS
Breaks if file deleted	Still exists

50 What is SELinux?

Answer:

Security layer that **restricts access**.

Modes:

- Enforcing
- Permissive
- Disabled

51 What is Git? (Deep Explanation)

Answer:

Git is a **distributed version control system** used to **track changes in source code**, collaborate with teams, and maintain history.

Why Git is critical for DevOps?

- CI/CD pipelines depend on Git
- Every commit can trigger automation
- Rollback and audit are easy

Real-world example:

Developer pushes code → Jenkins pipeline starts automatically.

52 Centralized vs Distributed Version Control

Centralized (SVN)	Distributed (Git)
Single server	Every user has full repo
Offline work not possible	Offline work possible
Slower	Faster

53 Git Architecture (IMPORTANT)

Git has **3 main areas**:

1. **Working Directory** – where you edit files
2. **Staging Area (Index)** – selected changes
3. **Repository** – committed history

Working → Staging → Repository

54 What is a Repository?

Answer:

A repository is a **storage area for project files and Git history**.

- Local repo → on your machine

- Remote repo → GitHub/GitLab
-

55 Git Workflow (End-to-End)

1. `git clone`
 2. Modify files
 3. `git status`
 4. `git add`
 5. `git commit`
 6. `git push`
-

56 What is git clone?

Answer:

Downloads a remote repository to local machine.

```
git clone https://github.com/user/repo.git
```

57 What is git status?

Answer:

Shows:

- Modified files
 - Staged files
 - Untracked files
-

58 Difference between git add and git commit?

git add	git commit
Stage changes	Save changes permanently
Temporary	Permanent

59 What is a Commit?

Answer:

A commit is a **snapshot of code** at a specific time.

Best practice:

- Small commits
 - Meaningful messages
-

60 What is git push?

Answer:

Uploads commits from local repo to remote repo.

61 What is git pull vs git fetch? (INTERVIEW FAVORITE)

git fetch	git pull
Only downloads	Downloads + merges
Safe	Can cause conflict
Review first	Immediate change

Interview tip:

“I prefer fetch before pull to avoid surprises.”

62 What is a Branch?

Answer:

A branch is an **independent line of development**.

Default branch:

- `main` or `master`
-

63 Why branches are important?

- Parallel development
 - Safe experimentation
 - CI pipelines per branch
-

64 How to create and switch branch?

```
git branch dev  
git checkout dev
```

or

```
git checkout -b dev
```

65 What is git merge?

Answer:

Combines changes from one branch into another.

```
git merge dev
```

Merge types:

- Fast-forward
 - 3-way merge
-

66 What is git rebase? (VERY IMPORTANT)

Answer:

Rebase moves commits **on top of another branch**, creating clean history.

```
git rebase main
```

Difference:

- Merge keeps history
 - Rebase rewrites history
-

67 Merge vs Rebase (INTERVIEW TABLE)

Merge	Rebase
Keeps history	Clean history
Extra commit	Linear commits
Safer	Risky for shared branches

👉 **Never rebase public branches**

68 What is a Git Conflict?

Answer:

Occurs when **same file & same lines** are modified in different branches.

Example:

Two developers edit `config.yml` same line.

69 How to resolve Git conflict?

1. Open conflicted file
 2. Choose correct changes
 3. Remove conflict markers
 4. `git add`
 5. `git commit`
-

70 What is git stash?

Answer:

Temporarily saves uncommitted changes.

```
git stash
git stash pop
```

Use case:

Switch branch without committing incomplete work.

71 What is git reset?

Answer:

Moves HEAD pointer backward.

Types:

- `--soft`
- `--mixed`
- `--hard`

```
git reset --hard HEAD~1
```

⚠ Dangerous if misused.

72 What is git revert?

Answer:

Creates a **new commit** that undoes changes.

Safe for production:

```
git revert commit_id
```

73 Difference between reset and revert?

Reset	Revert
Deletes history	Keeps history
Dangerous	Safe
Local use	Production use

74 What is HEAD in Git?

Answer:

HEAD points to **current branch/commit**.

75 What is git cherry-pick?

Answer:

Apply a **specific commit** from one branch to another.

```
git cherry-pick commit_id
```


76 What is CI/CD? (Deep Explanation)

Answer:

CI/CD is an **automation process** that helps teams **build, test, and deploy software continuously**.

CI (Continuous Integration)

- Developers push code frequently
- Automatic build & test
- Detect bugs early

CD (Continuous Delivery / Deployment)

- Code is always production-ready
- Automated deployment

Real-time flow:

Git push → Jenkins → Build → Test → Deploy

77 Why CI/CD is important in DevOps?

Answer:

CI/CD:

- Reduces manual work
 - Improves software quality
 - Enables fast releases
 - Prevents production failures
-

78 What is Jenkins?

Answer:

Jenkins is an **open-source CI/CD automation tool** written in Java.

Why Jenkins is popular?

- Plugin ecosystem
 - Pipeline as code
 - Works with any tech stack
-

79 Jenkins Architecture (IMPORTANT)

Jenkins Components:

1. **Jenkins Master (Controller)**
 - Schedules jobs
 - Manages UI & plugins
2. **Jenkins Agent (Worker)**
 - Executes builds

Why agents?

- Scalability
 - Isolation
-

80 What is a Jenkins Job?

Answer:

A job is a **task Jenkins executes**.

Types:

- Freestyle

- Pipeline
 - Multibranch
 - Folder
-

81 What is Jenkins Pipeline?

Answer:

A pipeline is **CI/CD workflow defined as code** using Groovy.

Benefits:

- Version controlled
 - Reusable
 - Automated
-

82 Declarative vs Scripted Pipeline

Declarative	Scripted
Simple syntax	Complex
Easy to read	More control
Recommended	Advanced use

83 What is Jenkinsfile?

Answer:

A **Jenkinsfile** defines **pipeline stages**.

Example:

```
pipeline {  
  agent any  
  stages {
```

```
stage('Build') {  
    steps {  
        sh 'mvn clean install'  
    }  
}  
}
```

84 Jenkins Pipeline Stages Explained

1. **Build** – Compile code
 2. **Test** – Unit/integration tests
 3. **Package** – Create artifact
 4. **Deploy** – Release application
-

85 What is Jenkins Agent?

Answer:

Agent is a **machine that runs pipeline steps**.

Agent types:

- Static
 - Dynamic (Docker, Kubernetes)
-

86 Jenkins Master vs Agent

Master	Agent
Controls Jenkins	Executes jobs
Lightweight	Heavy workloads

87 What is Jenkins Plugin?

Answer:

Plugins extend Jenkins functionality.

Examples:

- Git
 - Docker
 - Kubernetes
 - Blue Ocean
-

88 How does Jenkins integrate with Git?

Answer:

Using:

- Git plugin
- Webhooks

Flow:

Git push → Webhook → Jenkins job triggered

89 What is Webhook?

Answer:

Webhook is an **HTTP callback** that triggers Jenkins automatically.

90 What are Jenkins Credentials?

Answer:

Secure storage for:

- Passwords
- Tokens
- SSH keys

Best practice:

Never hardcode secrets in pipeline.

91 How to handle secrets in Jenkins?

- Jenkins credentials store
 - Environment variables
 - Vault integration
-

92 What is an Artifact?

Answer:

Artifact is the **output of build**.

Examples:

- JAR
 - WAR
 - Docker image
-

93 How to archive artifacts in Jenkins?

```
archiveArtifacts artifacts: '**/*.jar'
```

94 What is Blue Ocean?

Answer:

Modern Jenkins UI for pipelines.

95 Jenkins Failure Scenarios (VERY IMPORTANT)

Common failures:

- Build failure
- Test failure
- Dependency failure
- Permission issue

How to debug?

- Check console output
 - Verify logs
 - Re-run locally
-

96 How to retry failed stages?

```
retry(3) {  
    sh 'mvn test'  
}
```

97 What is Jenkins Parameterized Build?

Answer:

Allows dynamic input.

```
parameters {  
    string(name: 'ENV', defaultValue: 'dev')  
}
```

98 What is Jenkins Shared Library?

Answer:

Reusable pipeline code stored centrally.

Benefit:

- DRY principle
 - Standard pipelines
-

99 Jenkins Security Best Practices

- Role-based access
 - Disable anonymous access
 - Secure credentials
 - Use HTTPS
-

1000 Jenkins Best Practices (INTERVIEW GOLD)

- Pipeline as code
- Minimal plugins
- Separate build agents
- Automated testing
- Version Jenkinsfile

1001 What is Docker? (Deep Explanation)

Answer:

Docker is a **containerization platform** that packages an application along with its **dependencies, libraries, and configuration** into a single unit called a **container**.

Why Docker is important in DevOps?

- Same app runs everywhere
- Faster deployments
- Lightweight compared to VMs
- Perfect for CI/CD & Kubernetes

Real-world example:

Developer laptop → QA → Production → Same Docker image

1002 Difference between Virtual Machine and Container (DEEP)

Virtual Machine	Container
Full OS	Shares host kernel
Heavy	Lightweight
Slow startup	Fast startup
High resource usage	Low usage

👉 Docker containers **do not need a guest OS**.

1003 How Docker Works Internally?

Docker uses Linux features:

- **Namespaces** – isolation
 - **Cgroups** – resource limits
 - **UnionFS** – layered filesystem
-

104 What is Docker Image?

Answer:

A Docker image is a **read-only template** used to create containers.

Image contains:

- Base OS
 - Application
 - Dependencies
-

105 What is Docker Container?

Answer:

A container is a **running instance of a Docker image**.

👉 Image = blueprint

👉 Container = running app

106 What is Dockerfile? (VERY IMPORTANT)

Answer:

A Dockerfile is a **text file** that contains instructions to build a Docker image.

Example:

```
FROM ubuntu:22.04
RUN apt update && apt install -y nginx
CMD ["nginx", "-g", "daemon off;"]
```

107 Dockerfile Instructions Explained

Instruction	Purpose
FROM	Base image
RUN	Execute command
COPY	Copy files
ADD	Copy + extract
CMD	Default command
ENTRYPOINT	Fixed command
EXPOSE	Port
ENV	Environment variable

108 CMD vs ENTRYPOINT (INTERVIEW FAVORITE)

CMD	ENTRYPOINT
Can be overridden	Cannot be overridden
Default args	Fixed command

👉 Best practice: Use **ENTRYPOINT + CMD**

109 What is Docker Volume?

Answer:

Docker volume is used for **persistent storage**.

Why needed?

Containers are ephemeral.

```
docker volume create data_vol
```

110 Types of Docker Storage

1. Volumes (recommended)
 2. Bind mounts
 3. tmpfs
-

111 What is Docker Networking?

Answer:

Docker networking allows containers to **communicate with each other**.

Network types:

- Bridge (default)
 - Host
 - None
 - Overlay
-

112 Expose vs Publish Port

EXPOSE	-p
Documentation	Maps port
Internal	External access

113 What is Docker Compose?

Answer:

Docker Compose is used to **run multi-container applications**.

Example:

App + DB + Cache

114 docker-compose.yml Example

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
```

115 docker build vs docker run

build	run
Creates image	Runs container
Once	Many times

116 docker ps vs docker ps -a

docker ps	docker ps -a
Running containers	All containers

117 docker exec

Answer:

Used to access running container.

```
docker exec -it container_id bash
```

118 docker logs

Answer:

View container logs.

```
docker logs container_id
```

119 What is Docker Registry?

Answer:

Registry stores Docker images.

Examples:

- Docker Hub
- ECR
- GCR

120 What is Multi-stage Build?

Answer:

Build image in **multiple stages** to reduce size.

Benefit:

- Smaller image
- More secure

121 Docker Image Optimization Best Practices

- Use small base images
- Combine RUN commands
- Remove cache
- Use .dockerignore

122 Docker Security Best Practices

- Non-root user
 - Scan images
 - Minimal images
 - Secrets management
-

123 What is Docker Swarm?

Answer:

Native Docker orchestration tool.

👉 Mostly replaced by Kubernetes.

124 Docker vs Kubernetes

Docker	Kubernetes
Container runtime	Orchestrator
Single host	Multi-node
Simple	Advanced

125 Common Docker Interview Traps

- ✗ Running containers as root
- ✗ Huge image sizes
- ✗ Hardcoding secrets
- ✗ No health checks

126 What is Kubernetes? (Deep Explanation)

Answer:

Kubernetes (K8s) is an **open-source container orchestration platform** that automates:

- Deployment
- Scaling
- Load balancing
- Self-healing
- Rollouts & rollbacks

👉 Docker runs containers

👉 Kubernetes **manages containers at scale**

127 Why Kubernetes is needed?

Without Kubernetes:

- Manual container management
- No auto-healing
- No scaling
- Downtime during deployment

With Kubernetes:

- Pods restart automatically
 - Traffic is load-balanced
 - Zero-downtime deployments
 - Auto-scaling
-

128 Kubernetes Architecture (VERY IMPORTANT)

Main Components:

Control Plane (Master):

1. **API Server**
2. **Scheduler**
3. **Controller Manager**
4. **ETCD**

Worker Node:

1. **Kubelet**
2. **Container Runtime**
3. **Kube-proxy**

129 What is API Server?

Answer:

API Server is the **entry point** to Kubernetes.

- All `kubectl` commands go through it
- Communicates with ETCD
- Validates requests

👉 If API server is down → cluster is unusable

130 What is ETCD?

Answer:

ETCD is a **key-value database** that stores:

- Cluster state
- Configurations
- Secrets metadata

⚠ ETCD backup is **CRITICAL**

131 What is Scheduler?

Answer:

Scheduler decides **which pod runs on which node** based on:

- Resource availability
 - Node labels
 - Taints & tolerations
-

132 What is Controller Manager?

Answer:

Ensures **desired state == actual state**.

Example:

If replicas = 3 and 1 pod dies → controller creates new pod.

133 What is a Node?

Answer:

A node is a **worker machine** that runs pods.

Each node has:

- Kubelet
 - Container runtime
 - Kube-proxy
-

134 What is Kubelet?

Answer:

Kubelet is an **agent** running on nodes.

- Talks to API server
 - Ensures pods are running
-

135 What is a Pod? (VERY IMPORTANT)

Answer:

Pod is the **smallest deployable unit** in Kubernetes.

- One or more containers
- Shared network & storage

👉 Containers never run directly, always inside a Pod.

136 Pod vs Container

Pod	Container
Kubernetes concept	Docker concept
Can have many containers	Single process
Shared resources	Isolated

137 What is a Deployment?

Answer:

Deployment manages **replicas and updates** of pods.

Features:

- Rolling updates
- Rollback

- Scaling
-

138 What is ReplicaSet?

Answer:

Ensures **specified number of pods are always running**.

👉 Deployment uses ReplicaSet internally.

139 What is a Service?

Answer:

Service exposes pods and provides **stable networking**.

Because pod IPs change.

140 Types of Services (INTERVIEW FAVORITE)

Type	Use Case
ClusterIP	Internal access
NodePort	External testing
LoadBalancer	Production
ExternalName	External service

141 What is Ingress?

Answer:

Ingress manages **HTTP/HTTPS routing** to services.

Benefits:

- One load balancer
- SSL termination

- Path-based routing
-

142 What is ConfigMap?

Answer:

Stores **non-sensitive** configuration data.

Example:

- App config
 - Environment variables
-

143 What is Secret?

Answer:

Stores **sensitive** data:

- Passwords
- Tokens
- Keys

⚠ Secrets are base64 encoded, not encrypted by default.

144 What are Liveness & Readiness Probes?

Liveness Probe:

Checks if app is alive
→ Restarts container

Readiness Probe:

Checks if app can accept traffic
→ Stops traffic temporarily

145 What is Rolling Update?

Answer:

Updates pods **gradually** without downtime.

Example:

Old pod → New pod → Traffic switch

146 What is HPA (Horizontal Pod Autoscaler)?

Answer:

Automatically scales pods based on:

- CPU
 - Memory
 - Custom metrics
-

147 What is Namespace?

Answer:

Logical separation inside cluster.

Examples:

- dev
 - qa
 - prod
-

148 What is DaemonSet?

Answer:

Runs **one pod on every node**.

Use cases:

- Monitoring agents
 - Log collectors
-

149 What is StatefulSet?

Answer:

Used for **stateful applications**.

Features:

- Stable network ID
- Persistent storage

Example:

Databases

150 Kubernetes Troubleshooting (INTERVIEW GOLD)

Step-by-step:

1. `kubectl get pods`
2. `kubectl describe pod`
3. `kubectl logs`
4. Check events
5. Check node status

151 What is Cloud Computing? (Deep Explanation)

Answer:

Cloud computing means **using computing resources (servers, storage, networking) over the internet on-demand** instead of managing physical hardware.

Key benefits:

- Pay-as-you-go
 - Scalability
 - High availability
 - No hardware maintenance
-

152 Types of Cloud Models

Service models:

1. **IaaS** – EC2, VM
 2. **PaaS** – Elastic Beanstalk
 3. **SaaS** – Gmail
-

153 Public vs Private vs Hybrid Cloud

Type	Description
Public	AWS, Azure
Private	Company owned
Hybrid	Both

154 What is AWS?

Answer:

AWS is a **public cloud provider** offering compute, storage, networking, security, and DevOps services.

155 What is EC2?

Answer:

EC2 (Elastic Compute Cloud) is a **virtual server** in AWS.

Key concepts:

- Instance type
- AMI
- Security Group
- Key pair

156 What is AMI?

Answer:

AMI is a **template** used to launch EC2 instances.

157 What is Security Group?

Answer:

A **virtual firewall** controlling inbound & outbound traffic.

👉 Stateful firewall.

158 What is VPC?

Answer:

VPC is a **logically isolated virtual network** in AWS.

159 Public vs Private Subnet

Public	Private
Has internet gateway	No direct internet
Web servers	DB servers

160 What is IAM?

Answer:

IAM manages **users, roles, and permissions** in AWS.

Best practice:

Use **roles instead of users** for EC2.

161 What is S3?

Answer:

S3 is **object storage** for:

- Backups
 - Logs
 - Static websites
-

162 What is Load Balancer?

Answer:

Distributes traffic across multiple servers.

Types:

- ALB
- NLB
- CLB

163 What is Auto Scaling?

Answer:

Automatically adjusts EC2 count based on load.

164 What is CloudWatch?

Answer:

Monitoring service for AWS resources.

165 What is Terraform? (Deep Explanation)

Answer:

Terraform is an **Infrastructure as Code (IaC)** tool used to create, update, and manage cloud resources using code.

166 Why Terraform is used?

- Automation
 - Version control
 - Reusable modules
 - Multi-cloud support
-

167 Terraform Workflow (VERY IMPORTANT)

1. `terraform init`
2. `terraform plan`

3. `terraform apply`
 4. `terraform destroy`
-

168 What is Terraform Provider?

Answer:

Provider connects Terraform with cloud platforms.

Example:

- AWS
 - Azure
 - GCP
-

169 What is Terraform State?

Answer:

State file tracks **real infrastructure**.

⚠ Critical file – never delete.

170 Local vs Remote Backend

Local	Remote
Stored locally	Stored in S3
Risky	Safe
No locking	State locking

171 What is Terraform Variable?

Answer:

Variables make code **dynamic and reusable**.

172 What are Output Variables?

Answer:

Display resource information after apply.

173 What is Terraform Module?

Answer:

A module is **reusable Terraform code**.

Example:

- VPC module
 - EC2 module
-

174 What is Terraform Workspace?

Answer:

Used to manage **multiple environments**.

Example:

- dev
 - qa
 - prod
-

175 What is Terraform Lifecycle?

Answer:

Controls resource behavior.

```
lifecycle {  
  prevent_destroy = true  
}
```

176 What is Terraform Taint?

Answer:

Forces recreation of resource.

177 What is Terraform Import?

Answer:

Imports existing infrastructure into Terraform state.

178 Terraform vs Ansible (INTERVIEW FAVORITE)

Terraform	Ansible
Provisioning	Configuration
Declarative	Procedural
Immutable	Mutable

179 Common Terraform Mistakes

- ✗ Not using remote backend
 - ✗ Editing state manually
 - ✗ Hardcoding secrets
 - ✗ No modules
-

180 DevOps Engineer Cloud Responsibilities

- Cloud infra design

- IaC automation
- Security
- Monitoring
- Cost optimization