# DevOps Interview Q&A

## What source code management tool are you using?

Most teams use Git with platforms like GitHub, GitLab, or Bitbucket for version control. These tools help manage code repositories, branching, merging, and collaboration.

## What CI/CD tools do you use?

Common CI/CD tools include:

- Jenkins

- GitLab CI/CD

- GitHub Actions

- CircleCI

- Azure DevOps

- ArgoCD (for Kubernetes)

## How do you troubleshoot issues in CI/CD pipelines?

Steps include:

1. Check logs

2. Validate configuration

3. Environment checks

4. Rollback

5. Test locally

## What kind of pipelines have you worked on?

Examples:

- Build pipelines

- Deployment pipelines

- Infrastructure pipelines

- Container pipelines

## What is a Jenkinsfile?

A Jenkinsfile is a script that defines a Jenkins pipeline using Groovy syntax. It includes stages like build, test, and deploy.

## What are your daily DevOps activities?

Typical tasks:

- Monitoring CI/CD pipelines

- Managing infrastructure (IaC)

- Handling deployments

- Troubleshooting issues

- Writing automation scripts

- Collaborating with developers

## Explain Git merge and branching strategies.

- Merge: Combines changes from one branch into another.

- Branching strategies:

- Git Flow

- GitHub Flow

- Trunk-based development

## Describe a full CI/CD flow in your project.

1. Developer pushes code to Git.

2. CI tool triggers build and test.

3. Artifacts are stored.

4. CD tool deploys to staging.

5. Approval gates for production deployment.

6. Monitoring and rollback mechanisms.

## How do you build a Docker image? Explain the steps.

1. Create a Dockerfile.

2. Run: docker build -t <image-name> .

3. Verify: docker images

4. Push: docker push <repo>/<image-name>

## How do you check CPU usage of a server?

- Linux: top, htop, vmstat, mpstat

- Windows: Task Manager, Get-Process

- Cloud: Monitoring tools like CloudWatch, Datadog

## What operating systems have you worked with?

- Linux (Ubuntu, CentOS, RHEL)

- Windows Server

- macOS

## What is Ansible? Why did you use it in your project?

Ansible is a configuration management tool. Used for:

- Automating server setup

- Deploying applications

- Managing infrastructure

## What is Terraform used for?

Terraform is an Infrastructure as Code (IaC) tool used to provision and manage cloud resources declaratively.

## What's the difference between Ansible and Terraform?

Feature | Ansible | Terraform

-------------- | ---------------------------- | --------------------------

Purpose | Configuration management | Infrastructure provisioning

Language | YAML | HCL

Execution | Agentless, push-based | Declarative, state-based

## Have you worked with Kubernetes? Explain your use case.

Yes. Use cases:

- Deploying microservices

- Managing containerized workloads

- Auto-scaling and self-healing apps

- Helm charts for packaging

## What AWS services are you familiar with?

Examples:

- EC2, S3, RDS, Lambda

- CloudFormation, IAM, VPC

- EKS, CloudWatch, CodePipeline

## Have you used monitoring tools? How and why are they used?

Yes. Tools like:

- Prometheus + Grafana

- Datadog

- New Relic

- AWS CloudWatch

Used for:

- Tracking performance

- Alerting on failures

- Capacity planning

## What do you do if there's a Git issue while merging or pushing?

Steps:

1. Check error message.

2. Resolve conflicts manually.

3. Use git status, git log, git diff.

4. Rebase or reset if needed.

5. Push with --force only if safe.

## What kind of Jenkins plugins have you used?

Examples:

- Git plugin

- Pipeline plugin

- Docker plugin

- Slack notification plugin

- Blue Ocean UI

## How do you trigger an Ansible playbook? What’s the command?

ansible-playbook playbook.yml

## How to run a specific task in Ansible?

Use tags:

ansible-playbook playbook.yml --tags "install"

## How do you manage Jira tickets or handle tasks?

- Use Jira boards for sprint planning.

- Update ticket status (To Do → In Progress → Done).

- Link commits and PRs to Jira issues.

- Add comments and time logs.

Q: Your EC2 has a public IP and the port is open in the security group, but it's unreachable. Why?  
  
A: Check the subnet’s network ACL. If inbound or outbound rules are blocking traffic, the security group won’t help. NACLs silently drop traffic with no message.  
  
Q: You shared an AMI with another AWS account, but they still can’t launch an instance from it. What’s usually missed?  
  
A: Sharing the AMI isn’t enough. You also need to share the associated EBS snapshot. Without that, the AMI looks valid but fails at launch.  
  
Q: You restored an RDS snapshot for staging, but some queries behave differently than production.   
  
A: When you restore from a snapshot, RDS assigns the default parameter group by default. Custom parameter groups from production are not restored automatically. If not manually reassigned, staging may run with different settings, leading to changes in query behavior or performance.  
  
Q: You enabled IAM roles for service accounts in EKS, but your pod can’t access S3. The role looks fine. What’s the catch?  
  
A: The pod must be using a service account with the right annotation linking to the IAM role. If the pod defaults to the default service account or the annotation is missing, the role doesn’t apply.  
  
Q: ALB is marking your targets as unhealthy, but hitting the app directly works fine.  
  
A: ALB health checks are strict. If your app returns a 301 or a login page without a clean 200 OK, it’ll fail the check even if the app seems fine in the browser.  
  
Q: You pushed a new image to ECR and updated your ECS task definition, but it still runs the old version.  
  
A: If you're using mutable tags like latest, ECS often pulls from cache. Unless you force a new digest or use a unique tag per version, you’ll keep running stale containers.  
  
Q: In EKS, your stateful pod using an EBS volume is stuck in Pending. Why doesn't it reschedule?  
  
A: EBS volumes are limited to a single Availability Zone. If EKS places the pod on a node in a different AZ, the volume cannot attach. Make sure your node group includes nodes in the same AZ as the volume.

𝐂𝐈/𝐂𝐃 𝐚𝐧𝐝 𝐏𝐢𝐩𝐞𝐥𝐢𝐧𝐞𝐬 → Your deployment pipeline frequently fails at random stages. How do you make it more reliable? → The product team wants faster releases, but your builds are slow. How do you speed up without breaking things? 𝐈𝐧𝐟𝐫𝐚𝐬𝐭𝐫𝐮𝐜𝐭𝐮𝐫𝐞 𝐚𝐬 𝐂𝐨𝐝𝐞 → Your Terraform script works in dev but crashes in prod. What’s your next step? → A junior accidentally deleted critical infra using IaC. How do you prevent this next time? 𝐌𝐨𝐧𝐢𝐭𝐨𝐫𝐢𝐧𝐠 𝐚𝐧𝐝 𝐎𝐛𝐬𝐞𝐫𝐯𝐚𝐛𝐢𝐥𝐢𝐭𝐲 → A service went down at 2 AM. You weren’t alerted. What would you fix in your monitoring setup? → You have logs, metrics, traces — but you’re still blind during outages. What’s missing? 𝐂𝐨𝐧𝐭𝐚𝐢𝐧𝐞𝐫𝐢𝐬𝐚𝐭𝐢𝐨𝐧 𝐚𝐧𝐝 𝐊𝐮𝐛𝐞𝐫𝐧𝐞𝐭𝐞𝐬 → Your Kubernetes cluster runs out of resources often. How would you manage and optimise it? → One pod works, the other fails with the same image. How would you debug it? 𝐒𝐞𝐜𝐮𝐫𝐢𝐭𝐲 𝐚𝐧𝐝 𝐀𝐜𝐜𝐞𝐬𝐬 𝐂𝐨𝐧𝐭𝐫𝐨𝐥 → Developers have too many permissions in production. How would you enforce least privilege? → A secret got exposed in a Git commit. What steps would you take now and for future protection? 𝐂𝐨𝐥𝐥𝐚𝐛𝐨𝐫𝐚𝐭𝐢𝐨𝐧 𝐚𝐧𝐝 𝐂𝐮𝐥𝐭𝐮𝐫𝐞 → Developers blame Ops. Ops blame Dev. The release is stuck. How would you bring ownership and alignment? → Your team is stuck firefighting. How would you build a culture of automation and stability?

**Copilot said:**

**Copilot**

Great set of DevOps scenario questions! Here's a breakdown of how you might approach each one with practical, reliable solutions:

**CI/CD and Pipelines**

**→ Deployment pipeline fails randomly. How to improve reliability?**

* **Add retry logic** for flaky steps.
* **Isolate and parallelize stages** to identify bottlenecks.
* Use **caching** for dependencies and builds.
* Implement **pipeline health dashboards** to track failure patterns.
* Ensure **environment consistency** (e.g., containerized runners).

**→ Builds are slow, but faster releases are needed. How to speed up safely?**

* Use **incremental builds** and **artifact caching**.
* Run **tests in parallel** and prioritize critical ones.
* Adopt **feature flags** for safer, faster deployments.
* Optimize Docker layers and dependencies.
* Profile and remove **redundant pipeline steps**.

**Infrastructure as Code (IaC)**

**→ Terraform works in dev but crashes in prod. Next steps?**

* Compare **state files** and **provider versions**.
* Use **workspaces** or separate state backends for environments.
* Validate with terraform plan before applying.
* Add **environment-specific variables** and conditionals.
* Use **mock/staging environments** for safer testing.

**→ Junior deleted critical infra. How to prevent this?**

* Implement **role-based access control (RBAC)**.
* Use **approval workflows** for destructive actions.
* Enable **version control and audit logs**.
* Add **pre-commit hooks** and terraform plan reviews.
* Educate with **IaC best practices and simulations**.

**Monitoring and Observability**

**→ Service went down at 2 AM, no alert. What to fix?**

* Review and tune **alert thresholds** and **notification channels**.
* Implement **heartbeat checks** and **synthetic monitoring**.
* Ensure **on-call rotations** and escalation policies are active.
* Use **alert deduplication and suppression rules** wisely.

**→ Logs, metrics, traces exist, but still blind during outages. What's missing?**

* Lack of **correlation** between observability data.
* Missing **contextual dashboards** or **service maps**.
* No **real-time anomaly detection** or **event timelines**.
* Poor **log enrichment** (e.g., missing request IDs).
* Consider **OpenTelemetry** for unified observability.

**Containerisation and Kubernetes**

**→ Cluster runs out of resources. How to manage and optimize?**

* Use **resource requests and limits** properly.
* Implement **Horizontal Pod Autoscaling (HPA)**.
* Monitor with **Kube-state-metrics** and **Prometheus**.
* Clean up **unused resources** and optimize node sizes.
* Consider **Cluster Autoscaler** or **node pools**.

**→ One pod works, another fails with same image. Debug steps?**

* Check **pod logs** and **events** (kubectl describe pod).
* Compare **environment variables**, **volumes**, and **init containers**.
* Validate **network policies** and **service accounts**.
* Look for **node-specific issues** or taints.
* Use **debug containers** or ephemeral containers for inspection.

**Security and Access Control**

**→ Developers have too many prod permissions. Enforce least privilege?**

* Use **IAM roles** with fine-grained policies.
* Implement **Just-In-Time (JIT) access** and **approval workflows**.
* Audit access regularly with **access reviews**.
* Use **separate accounts/projects** for prod and non-prod.
* Educate on **principle of least privilege**.

**→ Secret exposed in Git commit. What now and how to prevent?**

* **Revoke and rotate** the secret immediately.
* Use tools like **GitGuardian** or **TruffleHog** to scan history.
* Add **pre-commit hooks** to prevent secret leaks.
* Store secrets in **vaults** (e.g., HashiCorp Vault, AWS Secrets Manager).
* Educate on **secure coding practices**.

**Collaboration and Culture**

**→ Dev vs Ops blame game. Release stuck. How to align?**

* Promote **DevOps culture**: shared goals and responsibilities.
* Use **blameless postmortems** and retrospectives.
* Implement **cross-functional teams** or squads.
* Define clear **SLAs and ownership boundaries**.
* Encourage **collaborative tooling** (e.g., shared dashboards).

**→ Team stuck firefighting. How to build automation and stability?**

* Identify **repetitive tasks** and automate them.
* Invest in **CI/CD, monitoring, and self-healing infra**.
* Allocate time for **tech debt and reliability work**.
* Celebrate **proactive improvements**, not just reactive fixes.
* Foster a culture of **learning and experimentation**.

Briefly introduce yourself. 2️⃣ What are your day-to-day DevOps activities? 3️⃣ How do you onboard a new service to an EKS cluster? 4️⃣ Write a basic Dockerfile. 5️⃣ What is the purpose of a multi-stage Dockerfile? 6️⃣ What does ENTRYPOINT do in Docker? 7️⃣ Difference between ENTRYPOINT and CMD? 8️⃣ Difference between COPY and ADD in Docker? 9️⃣ Where to place a public key if I give you the key and I hold the private key? 🔟 Redis is down — how will you troubleshoot? 1️⃣1️⃣ How can a private EC2 instance access the internet? 1️⃣2️⃣ Besides NAT Gateway, what other options are there? 1️⃣3️⃣ What are VPC Endpoints? 1️⃣4️⃣ Can you access a private instance via a VPC endpoint? 1️⃣5️⃣ Why use Kubernetes when we have Auto Scaling Groups? 1️⃣6️⃣ Is Kubernetes only for cost optimization? 1️⃣7️⃣ What logging/monitoring system do you use for apps? 1️⃣8️⃣ What is Elasticsearch? 1️⃣9️⃣ Have you worked with Kibana? 2️⃣0️⃣ Do you have any questions for us

**Copilot said:**

**Copilot**

Here’s a brief and structured response to each of your DevOps-related questions:

**1️⃣ Briefly introduce yourself.**

I'm a DevOps engineer with hands-on experience in cloud infrastructure, CI/CD pipelines, container orchestration, and automation. I specialize in deploying scalable applications using tools like Docker, Kubernetes, and AWS.

**2️⃣ Day-to-day DevOps activities**

* Managing CI/CD pipelines (Jenkins, GitHub Actions)
* Monitoring and logging (Prometheus, Grafana, ELK)
* Infrastructure provisioning (Terraform, CloudFormation)
* Container orchestration (Kubernetes, Docker)
* Security patching and compliance
* Incident response and troubleshooting

**3️⃣ Onboarding a new service to an EKS cluster**

1. Containerize the application using Docker
2. Push the image to a container registry (e.g., ECR)
3. Create Kubernetes manifests (Deployment, Service, ConfigMap, etc.)
4. Apply manifests using kubectl
5. Configure ingress and monitoring
6. Validate deployment and scale as needed

**4️⃣ Basic Dockerfile**

**FROM python:3.10-slim**

**WORKDIR /app**

**COPY requirements.txt.**

**RUN pip install -r requirements.txt**

**COPY. .**

**CMD ["python", "app.py"]**

**✅ How do you structure GitHub Actions workflows to support multi-service deployment with rollback?  
 ✅ Explain how to perform canary deployments using Helm and Argo Rollouts.  
 ✅ How do you securely inject API keys in GitHub Actions without exposing them in logs?  
 ✅ What’s the most efficient way to implement release versioning across microservices?  
  
🚀 Kubernetes & Scalability  
 ✅ How would you handle CPU throttling in a busy AKS/EKS cluster under payment traffic load?  
 ✅ What are readiness gates in Kubernetes and when would you use them?  
 ✅ Explain how you’d debug an issue where the HPA is not scaling pods despite high CPU usage.  
 ✅ How do you ensure blue-green or shadow deployment testing for critical services?  
  
🚀 Cloud Infrastructure & IaC (Terraform)  
 ✅ How do you structure Terraform code for a multi-region, multi-account AWS setup?  
 ✅ What is your approach to tagging and cost governance across cloud infrastructure?  
 ✅ How do you rotate AWS IAM credentials and ensure services update them dynamically?  
 ✅ What’s the benefit of using for\_each over count in complex Terraform modules?  
  
🚀 Observability & Debugging  
 ✅ How do you trace a spike in failed transactions to either app logic, infra failure, or a third-party API?  
 ✅ What’s your approach to building actionable alerts and reducing false positives in PagerDuty/Prometheus?  
 ✅ How would you implement distributed tracing in a Kubernetes-based microservices system?  
  
🚀 Security & Compliance (Fintech-Focused)  
 ✅ How do you prevent hardcoded secrets in Terraform and Dockerfiles?  
 ✅ What’s your strategy for enforcing least privilege access in a CI/CD pipeline?  
 ✅ How do you manage vulnerability scanning and patching in high-frequency deployments?**

**✅ Multi-Service Deployment with Rollback in GitHub Actions**

**Structure your workflow like this:**

* **Matrix Strategy: Deploy multiple services in parallel using a matrix.**
* **Environment-specific Jobs: Separate jobs for staging, canary, and production.**
* **Rollback Logic:**
  + **Use if: failure() to trigger rollback steps.**
  + **Integrate with Helm/ArgoCD to revert to previous revisions.**
  + **Store last known good deployment in a GitHub artifact or tag.**

**✅ Canary Deployments with Helm & Argo Rollouts**

* **Helm: Use values files to control traffic weights (canaryWeight, replicaCount).**
* **Argo Rollouts:**
  + **Define a Rollout CRD with steps like setWeight, pause, and analysis.**
  + **Use metrics (Prometheus, Datadog) for automated analysis.**
  + **Rollback is automatic if metrics fail during pause.**

**✅ Secure API Key Injection in GitHub Actions**

* **Use GitHub Secrets:**
* **Avoid echoing secrets in logs:**
  + **Use set +x in shell scripts.**
  + **Mask secrets using ::add-mask:: in workflows.**

**✅ Efficient Release Versioning Across Microservices**

* **Use Semantic Versioning with Git tags.**
* **Automate version bumps using tools like semantic-release.**
* **Maintain a central version registry or use GitHub Releases.**
* **Include service name + version in Docker tags: serviceA:v1.2.3.**

**🚀 Kubernetes & Scalability**

**✅ Handling CPU Throttling in AKS/EKS**

* **Root Cause: Often due to CPU limits being set too low.**
* **Fixes:**
  + **Adjust resources.limits.cpu and requests.cpu.**
  + **Use Vertical Pod Autoscaler for dynamic tuning.**
  + **Monitor with kubectl top pods and Prometheus.**

**✅ Readiness Gates in Kubernetes**

* **Purpose: Delay pod readiness until external conditions are met.**
* **Use Cases:**
  + **Waiting for cloud load balancer health checks.**
  + **Ensuring secrets/configs are injected.**
* **Defined via readinessGates in PodSpec and custom conditions.**

**✅ Debugging HPA Not Scaling**

* **Check:**
  + **Metrics server is running (kubectl get apiservices).**
  + **HPA target metrics (kubectl describe hpa).**
  + **Pod limits are not throttling CPU.**
  + **Use kubectl top pods to verify actual usage.**

**✅ Blue-Green or Shadow Deployment Testing**

* **Blue-Green: Deploy to a parallel environment, switch traffic using Ingress or service selector.**
* **Shadow: Duplicate traffic using Istio or Envoy to a shadow service.**
* **Use synthetic monitoring and real user metrics to validate.**

**🚀 Cloud Infrastructure & IaC (Terraform)**

**✅ Terraform for Multi-Region, Multi-Account AWS**

* **Use root module with account/region-specific child modules.**
* **Structure:**
* **├── modules/**
* **├── environments/**
* **├── prod/**
* **├── staging/**
* **Use provider aliasing and backend configuration per account.**

**✅ Tagging & Cost Governance**

* **Enforce tagging via terraform validate or Sentinel policies.**
* **Use AWS Cost Explorer + tag-based filters.**
* **Automate tag injection using modules or wrapper scripts.**

**✅ Rotating AWS IAM Credentials**

* **Use IAM Roles with STS for short-lived credentials.**
* **Rotate secrets via AWS Secrets Manager.**
* **Services fetch updated credentials using SDKs or sidecars.**

**✅ for\_each vs count in Terraform**

* **for\_each is better for:**
  + **Associative maps.**
  + **Avoiding index-based errors.**
  + **Managing resources with unique identifiers.**
* **count is simpler but less flexible.**

**🚀 Observability & Debugging**

**✅ Tracing Failed Transactions**

* **Use distributed tracing (OpenTelemetry, Jaeger).**
* **Correlate logs with trace IDs.**
* **Check:**
  + **App logs for exceptions.**
  + **Infra metrics (CPU, memory, disk).**
  + **API gateway logs for third-party failures.**

**✅ Actionable Alerts & Reducing False Positives**

* **Use multi-condition alerts (e.g., error rate + latency).**
* **Group alerts by service and severity.**
* **Use silencing rules during deployments.**
* **Regularly review alert fatigue metrics.**

**✅ Distributed Tracing in Kubernetes**

* **Inject trace headers via sidecars or SDKs.**
* **Use OpenTelemetry Collector to export to Jaeger/Tempo.**
* **Annotate spans with Kubernetes metadata (pod name, namespace).**

**🚀 Security & Compliance (Fintech-Focused)**

**✅ Prevent Hardcoded Secrets**

* **Use tools like tfsec, checkov, and gitleaks.**
* **Store secrets in Vault or AWS Secrets Manager.**
* **Inject secrets at runtime via environment variables or sidecars.**

**✅ Least Privilege in CI/CD**

* **Use scoped GitHub tokens.**
* **Define IAM roles per environment.**
* **Use OIDC with GitHub Actions for temporary credentials.**

**✅ Vulnerability Scanning & Patching**

* **Integrate tools like Trivy, Snyk, or Grype in CI.**
* **Use base image scanning + dependency scanning.**
* **Automate patching via Renovate or Dependabot.**

**5️⃣ Purpose of a multi-stage Dockerfile**

* Reduces image size by separating build and runtime environments
* Improves security by excluding build tools from final image
* Speeds up deployment and reduces attack surface

**6️⃣ What does ENTRYPOINT do in Docker?**

ENTRYPOINT defines the main command that always runs when the container starts. It’s useful for setting fixed behavior.

**7️⃣ Difference between ENTRYPOINT and CMD**

| **Feature** | **ENTRYPOINT** | **CMD** |
| --- | --- | --- |
| Purpose | Fixed command | Default arguments |
| Overridable | No (unless using --entrypoint) | Yes (via CLI) |
| Use Case | Always run a script | Provide default parameters |

**8️⃣ Difference between COPY and ADD in Docker**

| **Feature** | **COPY** | **ADD** |
| --- | --- | --- |
| Source | Local files only | Local + remote URLs + archives |
| Extraction | No | Yes (auto-extracts .tar) |
| Preferred | Yes (for clarity) | Use only when needed |

**9️⃣ Where to place a public key if I give you the key and I hold the private key?**

Place the public key in the target machine’s ~/.ssh/authorized\_keys file to allow secure SSH access.

**🔟 Redis is down — how will you troubleshoot?**

1. Check Redis logs (/var/log/redis)
2. Verify service status (systemctl status redis)
3. Test connectivity (redis-cli ping)
4. Check memory and disk usage
5. Review firewall and port settings
6. Restart Redis and monitor

**1️⃣1️⃣ How can a private EC2 instance access the internet?**

Via a **NAT Gateway** or **NAT Instance** in a public subnet, with proper route table configuration.

**1️⃣2️⃣ Besides NAT Gateway, what other options are there?**

* NAT Instance
* VPN or Direct Connect
* VPC Endpoint (for AWS services only)

**1️⃣3️⃣ What are VPC Endpoints?**

Private connections to AWS services without using the public internet. Two types:

* **Interface endpoints** (ENI-based)
* **Gateway endpoints** (for S3 and DynamoDB)

**1️⃣4️⃣ Can you access a private instance via a VPC endpoint?**

No, VPC endpoints are for accessing AWS services, not EC2 instances. Use a bastion host or VPN for EC2 access.

**1️⃣5️⃣ Why use Kubernetes when we have Auto Scaling Groups?**

* Fine-grained resource management
* Rolling updates and self-healing
* Declarative configuration
* Better workload orchestration

**1️⃣6️⃣ Is Kubernetes only for cost optimization?**

No. It’s primarily for **scalability, resilience, automation**, and **portability** across environments.

**1️⃣7️⃣ What logging/monitoring system do you use for apps?**

* **Logging**: ELK Stack (Elasticsearch, Logstash, Kibana), Fluentd
* **Monitoring**: Prometheus + Grafana, AWS CloudWatch

**1️⃣8️⃣ What is Elasticsearch?**

A distributed search and analytics engine used for log aggregation, full-text search, and real-time data analysis.

**1️⃣9️⃣ Have you worked with Kibana?**

Yes. Used it for visualizing logs, creating dashboards, and analyzing application performance metrics.

**2️⃣0️⃣ Do you have any questions for us?**

* What does a typical day look like for your DevOps team?
* What tools and technologies are currently in use?
* How do you handle incident management and postmortems?

# DevOps, AWS, and API Security Q&A Summary

## How does API Gateway work with AWS Lambda?

API Gateway acts as a front door for applications to access data, business logic, or functionality from backend services like AWS Lambda. It:

- Accepts HTTP requests.

- Maps them to Lambda functions.

- Handles request/response transformation, throttling, and authorization.

- Returns the Lambda response to the client.

## How to protect API endpoints?

You can secure APIs using:

- Authentication: Use OAuth2, API keys, or JWT.

- Authorization: IAM roles, scopes, or custom logic.

- Rate limiting: Prevent abuse via throttling.

- WAF (Web Application Firewall): Block malicious traffic.

- HTTPS: Encrypt data in transit.

## What are tokens in API security?

Tokens are credentials used to authenticate and authorize API requests. Common types:

- Bearer tokens: Sent in the Authorization header.

- Access tokens: Short-lived, used to access resources.

- Refresh tokens: Used to obtain new access tokens.

## What is JWT (JSON Web Token) and where is it used?

JWT is a compact, URL-safe token format used for:

- Authentication: Verifying user identity.

- Authorization: Granting access to resources.

It contains:

- Header: Token type and algorithm.

- Payload: Claims (user info, roles).

- Signature: Verifies integrity.

Used in:

- OAuth2 flows.

- Stateless authentication in microservices.

- API Gateway custom authorizers.

## How to deploy a React app using DevOps practices?

Typical DevOps pipeline:

1. Source Control: Push code to Bitbucket/GitHub.

2. CI/CD: Use Bitbucket Pipelines/GitHub Actions.

3. Build: Run npm run build.

4. Test: Linting, unit tests.

5. Deploy: Upload to S3 or a container registry.

6. Notify: Slack/email alerts.

## How to host applications in AWS S3?

Steps:

1. Build your app (npm run build).

2. Create an S3 bucket.

3. Enable static website hosting.

4. Upload files to the bucket.

5. Set bucket policy for public access (if needed).

6. Use Route 53 or CloudFront for custom domains and caching.

## What is the alternative to AWS CloudFront?

Alternatives include:

- Azure CDN

- Google Cloud CDN

- Cloudflare

- Fastly

- Akamai

Each offers global edge caching, DDoS protection, and performance optimization.

## What are Bitbucket runners and how do they work?

Bitbucket Runners are custom execution environments for Pipelines:

- Run builds on your infrastructure.

- Useful for private dependencies or custom tooling.

- Configured via bitbucket-pipelines.yml.

- Supports Docker-based runners.

## How to securely store secrets in Bitbucket Pipelines?

Use Bitbucket Repository Variables:

- Go to Repository Settings > Pipelines > Repository Variables.

- Store secrets like API keys, tokens.

- Access them in pipeline using $SECRET\_NAME.

For extra security:

- Use secured variables (masked in logs).

- Rotate secrets regularly.

## What are pre-commit hooks in DevOps?

Pre-commit hooks are scripts that run before a commit is made:

- Enforce code standards (linting, formatting).

- Run tests.

- Prevent bad commits.

Tools: pre-commit, Husky, lint-staged.

## How to troubleshoot EC2 provisioning issues in a CI/CD pipeline?

Check:

- IAM roles: Ensure pipeline has EC2 permissions.

- Instance limits: Check AWS quotas.

- AMI and subnet: Validate configuration.

- Logs: Use CloudTrail and EC2 console.

- Pipeline output: Look for error messages.

## How to check IAM permission issues during deployment?

Use:

- IAM Policy Simulator: Test permissions.

- CloudTrail: Audit denied actions.

- AWS CLI: Run aws sts get-caller-identity to verify identity.

- Deployment logs: Look for AccessDenied errors.