DevOps Engineer Interview Q&A – Jenkins-Centric Microservice Platform

A curated set of interview questions and detailed answers derived from this repository's architecture and practices. Use it to prepare for role-based discussions across CI/CD, Kubernetes (EKS), Terraform, security (DevSecOps), SRE, monitoring, and operations.

Table of Contents

- AWS infrastructure design and integrations

Repository Overview and High-Level Architecture

Q1. Can you explain the overall architecture end to end?

- Answer:
  - Internet → ALB → Istio Gateway → EKS → Java microservice Pods → RDS.
  - Istio provides gateway, mTLS, traffic management; Helm manages K8s manifests per environment; EKS handles scaling and node orchestration; RDS provides resilient data tier (Multi-AZ, backups).
  - Observability stack (Prometheus + Grafana + EFK + Jaeger) gives metrics, logs, and distributed traces.
  - CI/CD is Jenkins-centric with Docker, ECR, SonarQube, Trivy, and Helm deploys.

Q2. Why a 3-tier model (web, app, DB) on EKS?

- Answer: Clear separation of concerns enhances scalability and security. ALB/Istio

terminate TLS and apply routing policies, app tier scales via HPA, and DB tier (RDS) isolates state with backups and PITR. This aligns with cloud-native and 12-factor principles.

Q3. How do you keep the application stateless?

- Answer: No local state; config externalized via ConfigMaps/Secrets; sessions avoided or backed by external store; images immutable with environment-specific values passed via Helm; filesystems read-only where possible.

Q4. What are common failure domains and how are they mitigated?

- Answer:
  - Cluster/node failures → Multi-AZ, Cluster Autoscaler, HPA.
  - Networking → ALB health checks, Istio retries/circuit breakers.
  - Database → RDS Multi-AZ, automated backups.
  - CI/CD → Rollbacks with Helm/image tags; artifact immutability; approval gates.

Jenkins CI/CD Pipeline Management

Q5. What’s the pipeline structure and how is it triggered?

- Answer: Declarative pipeline (~676 lines) with stages: checkout → quality (SonarQube) → tests (JUnit) → security scan (Trivy) → Docker build → ECR push → Helm deploy. Triggered by Git webhooks per branch with environment branching strategy.

Q6. What checks form part of your daily routine on Jenkins?

- Answer: Morning review of overnight failures; pipeline success rates and trend charts; agent resource utilization; artifact promotions between environments; pending approvals; and queued builds. Triage red builds, open tickets for flaky tests, and communicate risks for the day’s deployments.

Q7. How do you enforce quality and security gates?

- Answer: SonarQube quality gate is awaited; pipeline fails if status != OK. Trivy runs on the built image with --exit-code 1 for HIGH/CRITICAL vulnerabilities to break the build. Both run before deploy stages.

Q8. How do you ensure consistent build environments?

- Answer: Use Maven-Java17 agents (immutable images), pin tool versions, cache Maven repo between builds (where safe), and isolate with containerized agents.

Q9. How are Docker images tagged and promoted?

- Answer: Tag with build number and commit SHA (e.g., app:build-1234-), push to ECR. Promotions select a known tag via Jenkins parameters (IMAGE_TAG) and redeploy without rebuilds.

Q10. How do you optimize pipeline runtime?

- Answer: Multi-stage Docker builds with layer caching; parallelize tests where

applicable; skip unnecessary steps for non-prod via parameters; only run deploy on relevant branches. Reduced image build time from ~15 minutes to ~3.

Q11. How do you handle flaky builds or dependency issues?

- Answer: Clean workspace, re-resolve dependencies, verify Maven repo connectivity, pin dependency versions in pom.xml, and implement retries with exponential backoff for transient network issues.

Q12. What does a typical deploy command look like from Jenkins?

- Answer: Helm upgrade --install with environment-specific overrides, e.g., setting image.tag=$BUILD_NUMBER and Spring profile via --set config.application.profiles.active.

Infrastructure as Code with Terraform (via Jenkins)

Q13. How do you manage Terraform state and concurrency?

- Answer: Remote state in S3 with versioning and encryption; DynamoDB table for state locks to prevent concurrent applies. Jenkins pipeline runs plan/apply, gated by branch (apply only on main).

Q14. How do you structure environments with Terraform?

- Answer: Use -var-file=terraform.tfvars (and/or workspace-specific var files). Optionally Terraform workspaces or separate state buckets per env. Sensitive variables come from Jenkins credentials/Parameter Store.

Q15. How do you secure AWS credentials in Jenkins?

- Answer: Use Jenkins credentials bindings, short-lived AWS STS tokens (where possible), and IRSA for in-cluster access.

Never commit secrets; audit usage and rotate periodically.

Q16. How do you recover from state or drift issues?

- Answer: Use terraform state list/show to inspect, terraform import for existing resources, terraform refresh/plan to view drift, and apply to reconcile. Back up state via S3 versioning; lock via DynamoDB to avoid corruption.

Q17. What happens in the pipeline if plan shows destructive changes?

- Answer: For non-prod: pipeline may proceed after approval; for prod: manual approval gates and peer review. Guardrails enforce tagging and resource policies; cost-impact annotations are reviewed.

Multi-Environment Deployment Strategy and Promotions

Q18. Describe your dev → staging → prod flow.

- Answer: Dev auto-deploy on feature/branch commits; staging deploy after integration testing; prod deploy requires manual approval. Promotions use the same container image with environment-specific Helm values and IMAGE_TAG parameter.

Q19. How do you parameterize builds?

- Answer: Jenkins parameters include DEPLOYMENT_ENVIRONMENT, SKIP_TESTS, FORCE_REBUILD, IMAGE_TAG. These control conditional stages and deploy targets.

Q20. How do you handle rollbacks?

- Answer: Use Helm rollback to previous revision or redeploy a previous IMAGE_TAG. Ensure config compatibility

and perform canary or blue/green where possible to minimize risk.

Q21. How do you guarantee zero-downtime?

- Answer: Rolling updates, readiness/liveness probes, Istio traffic shifting if needed, preStop hooks for graceful shutdown, and health checks at ALB and K8s level.

Application Architecture (Spring Boot, Actuator, Micrometer)

Q22. What key endpoints are exposed and why?

- Answer: /actuator/health (K8s probes, ALB checks), /actuator/prometheus (metrics scraping), /actuator/info (build metadata), and /api/metrics (custom business metrics). They enable health monitoring and observability.

Q23. How do you externalize configuration across environments?

- Answer: Use Helm values to populate ConfigMaps and Secrets, selecting Spring profiles via --set config.application.profiles.active=$ENV. The same container image runs in all environments.

Q24. What JVM/container optimizations are used?

- Answer: Set proper memory/CPU requests/limits, enable container-aware JVM flags (Java 17 does this automatically), tune GC based on workload, and right-size thread pools and DB connection pools.

Q25. How do you surface business and JVM metrics?

- Answer: Micrometer publishes to /actuator/prometheus; define custom counters/gauges/timers for business events; export JVM metrics (heap, GC, threads) automatically.

Q26. How do you ensure graceful shutdown?

- Answer: Implement preStop hook with sleep/drain, handle SIGTERM to stop intake, wait for inflight requests to finish, and ensure readiness probe fails before termination begins.

Kubernetes Orchestration (EKS, Helm) and Istio Service Mesh

Q27. How are Helm charts structured for multiple environments?

- Answer: A base chart with values.yaml and env-specific values-[dev|staging|prod].yaml. Deployment, Service, HPA, and Config templates use values for image tags, resources, autoscaling thresholds, and environment-specific configs.

Q28. What are your autoscaling policies?

- Answer: HPA targets CPU at ~70% and memory at ~80%. Cluster Autoscaler adds nodes when pending pods can’t schedule. Vertical Pod Autoscaler can be used for right-sizing.

Q29. How do you enforce pod-level security?

- Answer: Run as non-root, drop unnecessary Linux capabilities, read-only root filesystem, resource limits, restricted Pod Security

Standards, and dedicated ServiceAccounts with least-privilege RBAC. Use IRSA for AWS integration.

Q30. How is Istio used?

- Answer: Istio Gateway terminates at the mesh boundary; VirtualService manages routing; DestinationRule configures subsets and connection policies; mTLS enforces encrypted service-to-service traffic; canary and traffic-splitting enable safe rollouts.

Q31. How do you manage configuration drift across environments?

- Answer: Single chart with environment values, Git review for changes, automated CI checks. Future: GitOps via ArgoCD to continuously reconcile desired state.

Monitoring and Observability (Prometheus, Grafana, EFK, Jaeger)

Q32. What’s your metrics collection and scrape strategy?

- Answer: Prometheus scrapes every ~15 seconds. Service discovery finds targets in cluster; scrape jobs include application endpoints (/actuator/prometheus) and system components. Recording rules precompute common queries.

Q33. What dashboards and alerts are in place?

- Answer: Grafana dashboards cover application latency (P95/P99), throughput, error rates, JVM metrics, K8s cluster health, and business KPIs. Alerts for latency/error spikes, pod restarts, node pressure, and CI failures via Slack/Email/PagerDuty.

Q34. How do you centralize logs?

- Answer: EFK stack (Elasticsearch, Fluentd, Kibana) ingests pod logs. Structured JSON logging with correlation IDs allows trace-log-metric correlation.

Q35. How do you implement distributed tracing?

- Answer: Jaeger via OpenTelemetry/Jaeger client; propagate trace headers; visualize call graphs and latency per span; integrate with Grafana for cohesive troubleshooting.

Q36. How do you test observability?

- Answer: Synthetic checks, alert rule test fires to Slack, deliberate pod restarts to validate alerts, dashboard review in post-deploy checklists, and SLO burn-rate alerts to protect error budgets.

Security and DevSecOps

Q37. What security checks happen in the pipeline?

- Answer: SAST with SonarQube quality gate; Trivy scans for HIGH/CRITICAL vulnerabilities with fail-fast; optional dependency scanning; IaC linting for Terraform; SBOM generation and artifact signing (where applicable).

Q38. How do you manage secrets?

- Answer: Kubernetes Secrets encrypted at rest; integration with external-secrets operator sourcing from AWS SSM Parameter Store; least-privilege access via RBAC/IRSA; avoid plaintext in repos/logs.

Q39. How do you secure network traffic?

- Answer: Ingress via ALB to Istio Gateway; mTLS inside mesh; NetworkPolicies restrict east-west traffic; security groups at VPC level; WAF on ALB (optional) for L7 protections.

Q40. How do you ensure compliance and auditability?

- Answer: Audit logging for Jenkins and cluster; artifact traceability via tags/SHA; enforce code reviews; retain build logs; map controls to SOC2/PCI/GDPR where in-scope.

Q41. How are RBAC and permissions handled?

- Answer: Namespaced roles with least privilege; service accounts per microservice; separate prod vs non-prod access; change via Git PRs; IRSA for AWS services access without node-level permissions.

Troubleshooting and Problem-Solving

Q42. Pipeline fails due to Maven dependency issues — what's your approach?

- Answer: Verify repo availability; clear local caches; pin/upgrade versions in pom.xml; check proxy settings on agents; retry with backoff; inspect effective POM and dependency tree; isolate flaky repos via mirrors.

Q43. Pods fail to start (CrashLoopBackOff/ImagePullBackOff) — how do you triage?

- Answer: Check kubectl logs -f and describe pod for events; verify image exists and pull secrets; validate resource limits/requests; inspect ConfigMaps/Secrets mounts; ensure correct image tag in Helm values.

Q44. High response times in production — how do you debug?

- Answer: Check Grafana latency panels; correlate with HPA events; inspect JVM GC

and heap; analyze DB pool saturation; profile slow endpoints with tracing; consider caching/hot paths.

Q45. Service is healthy but ALB reports unhealthy — what next?

- Answer: Confirm health check path matches /actuator/health; verify 200 response and readiness gating; check timeout/interval thresholds; network ACLs/security groups; Istio/VirtualService routing rules.

Q46. Terraform apply stuck due to lock — how do you resolve?

- Answer: Inspect DynamoDB lock entry; ensure no other apply is running; if confirmed stale, remove lock entry; re-run plan/apply; study state diffs before force operations.

Configuration Management with Ansible

Q47. How do you use Ansible in this project?

- Answer: A comprehensive playbook (~618 lines) provisions servers, hardens security (fail2ban, UFW, SSH), configures Java app, sets up monitoring (CloudWatch agent). Executed from Jenkins as part of configuration deployments.

Q48. How do you ensure idempotency and safety?

- Answer: Use declarative modules (package, template, systemd), handlers for restarts only on change, check_mode for dry runs, vault/SSM for secrets, and tags to scope runs.

Q49. How do you roll out config changes?

- Answer: Render application.yml via template module, deploy to /opt/java-microservice/config/application.yml, and restart service via systemd handler. Validate via health checks post-deploy.

Q50. How do you integrate Ansible with Jenkins?

- Answer: Jenkins stage triggers ansible-playbook with inventory and extra-vars per env; credentials injected securely; logs captured as artifacts; failures fail the pipeline.

Q51. What’s your rollback strategy for config?

- Answer: Keep prior templates/versioned artifacts; if regression, re-apply previous version and restart; maintain change history in Git and annotate releases.

Metrics and KPIs (Engineering and Business)

Q52. What pipeline KPIs do you track and why?

- Answer: Build success rate (>95%), duration (~8–12 min), deployment frequency (daily dev, weekly prod), MTTR (<30 min). These reflect delivery performance and reliability.

Q53. What application SLIs/SLOs do you enforce?

- Answer: Latency P95 <200ms, P99 <500ms; error rate <0.1%; uptime 99.9%. Monitored via Prometheus/Grafana with burn-rate alerting on error budgets.

Q54. How do you monitor infrastructure efficiency and costs?

- Answer: CPU <70%, Memory <80%, HPA activity, node utilization, and monthly cost reviews. Use right-sizing, cleanup

automation, and spot usage where appropriate.

Q55. How do KPIs feed into continuous improvement?

- Answer: Bottlenecks (e.g., long builds) drive optimizations like caching and parallelism; incident postmortems reduce MTTR; capacity trends inform scaling and cost controls.

Disaster Recovery and Backup

Q56. What’s your backup strategy across tiers?

- Answer: RDS automated backups with 7-day retention; stateless app design; Terraform state in S3 with versioning; Jenkins configuration backed up to Git. Artifacts are immutable in ECR.

Q57. How do you perform recovery?

- Answer:
  - App: rollback to previous image tag or Helm revision.
  - DB: RDS PITR to specific timestamp.
  - Infra: re-deploy from Terraform code and recover state from S3.
  - Environment: leverage blue/green to minimize downtime.

Q58. How do you validate DR readiness?

- Answer: Game days, restore drills, RTO/RPO targets (RTO <1h, RPO <15m), documented runbooks, and post-exercise reviews.

Key Talking Points and Trade-offs

Q59. Why Jenkins over GitHub Actions in this environment?

- Answer: Greater control over agents (including on-prem), rich plugin ecosystem, and legacy integrations. Hybrid/on-prem compliance needs favor Jenkins. Actions is great for GitHub-centric workflows but can be limited for complex enterprise agent topologies.

Q60. How did you scale Jenkins?

- Answer: Kubernetes plugin for dynamic agents, scaling build capacity with demand, cutting queue times by ~60%. Use labels for workload types (maven-java17), node affinities, and resource quotas.

Q61. How did you optimize Docker builds and tests?

- Answer: Multi-stage builds, minimal base images, layer caching, and parallel test stages. Reduced image build time from

~15m to ~3m and overall pipeline time by ~40%.

Q62. How does observability support uptime targets?

- Answer: Golden signals dashboards, actionable alerts, and tracing reduce MTTR and prevent incidents, enabling 99.9% uptime.

Future Roadmap and Continuous Improvements

Q63. Why move toward GitOps with ArgoCD?

- Answer: Continuous reconciliation, declarative desired state, auditability, and reduced drift. Helm charts become sources of truth; Jenkins triggers Git changes, ArgoCD handles deploy.

Q64. What advanced Istio patterns will you adopt?

- Answer: Canary with traffic splitting, circuit breakers, retries, outlier detection; request-level policies and fine-grained authorization.

Q65. How will you cut costs further?

- Answer: Spot instances for Jenkins agents and non-prod nodes; right-sizing via VPA; scheduled scale-down; registry and log retention policies.

Q66. How will you enhance secrets management?

- Answer: Integrate HashiCorp Vault with dynamic secrets, PKI, and short TTLs; migrate from static K8s Secrets/SSM parameters over time.

Q67. What’s next for observability?

- Answer: Deeper tracing via OpenTelemetry, SLO dashboards with burn-rate alerts, and trace-to-log correlation improvements.

SRE and Operational Excellence

Q68. How do you define and enforce SLOs and error budgets?

- Answer: Define SLIs (latency, availability, error rate); SLOs (P95 <200ms, 99.9% uptime); track error budgets and implement burn-rate alerts; pause risky deploys when budgets are low.

Q69. How do you reduce MTTR?

- Answer: Rich telemetry, clear runbooks, incident command process, on-call rotations, chaos drills, and rapid rollback mechanisms.

Q70. What’s your approach to release safety?

- Answer: Progressive delivery (canary/blue-green), automated tests+gates, feature flags (where applicable), and fast rollbacks.

Q71. How do you manage change risk in production?

- Answer: Approvals, change windows, synthetic checks post-deploy, and continuous verification via metrics and logs.

AWS Infrastructure Design and Integrations

Q72. What’s the VPC and subnetting approach?

- Answer: Custom VPC (e.g., 10.0.0.0/16) with public subnets (ALB/ingress), private app subnets (EKS nodes), and isolated DB subnets (RDS) across AZs. NAT gateways for egress from private subnets.

Q73. How is the web tier implemented?

- Answer: ALB with SSL termination, health checks to /actuator/health, target groups for ingress to Istio gateway. Optional WAF for L7 protections.

Q74. How do EKS and IAM integrate securely?

- Answer: Managed node groups with autoscaling; IRSA maps ServiceAccounts to fine-grained IAM roles; Security Groups and NACLs restrict access.

Q75. How is the container registry managed?

- Answer: Amazon ECR with vulnerability scanning on push; images are tagged and

immutable; lifecycle policies manage retention; Jenkins uses ECR for push/pull with least-privilege permissions.

Q76. What additional AWS services are integrated?

- Answer: Systems Manager Parameter Store for secure params, CloudWatch for supplemental logs/metrics and alarms, S3 for artifacts and Terraform state, DynamoDB for TF state locking.

Appendix: Representative Commands and Snippets

- Jenkins infra pipeline (plan/apply):
  - plan: terraform plan -var-file="terraform.tfvars"
  - apply (main only): terraform apply -auto-approve
- Helm deploy from Jenkins:
  - helm upgrade --install java-microservice ./helm/java-microservice --namespace $ENVIRONMENT --set image.tag=$ENVIRONMENT --set image.tag =BUILD_NUMBER --set config.application.profiles.active=$ENVIRONMENT
- Security gate snippet:
  - Trivy: trivy image --exit-code 1 --severity HIGH,CRITICAL $IMAGE_NAME :IMAGENAME :{BUILD_NUMBER}
  - SonarQube gate: waitForQualityGate() and fail if status != OK

- K8s troubleshooting:
  - kubectl logs -f deployment/java-microservice
  - kubectl describe pod
- Observability checks:
  - /actuator/health, /actuator/prometheus, Grafana dashboards, Jaeger traces

---

End of document.