📝 Created by: Lamhot Siagian 🔗

111 CI/CD Interview Questions & Hints — SDET/QA

🎯 12 Cards 💡 Questions + Hints 🤖 SDET/QA • CI/CD

🧩 1. CI/CD Fundamentals — 10 Qs

Understand CI vs CD (Delivery vs Deployment) and the goal of small, frequent, automated changes.

- **Differentiate CI, Continuous Delivery, and Continuous Deployment.** CI integrates frequently; Delivery keeps main deployable; Deployment auto-releases with safeguards.
- **Why small batch sizes?** Reduces risk, eases rollback, accelerates feedback.
- **Key CI signals of health?** Green builds, short lead time, low MTTR, high change frequency.
- **What breaks CI most?** Long-lived branches, flaky tests, manual gates, env drift.
- **Build vs release vs deploy artifacts?** Build produces versioned artifact; release tags it; deploy promotes to env.
- **"Shift left" meaning?** Testing/quality activities move earlier in pipeline.
- **CI success metrics?** DORA metrics: Lead time, Deployment freq, MTTR, Change fail rate.
- **Trunk-based vs GitFlow?** Trunk favors rapid CI; GitFlow helpful for release lines but can slow feedback.
- **Immutable builds?** Never rebuild same SHA; promote identical artifact across envs.
- **Blue/Green vs Canary?** Blue/Green swaps entire stacks; Canary releases to subset for risk control.

⚙️ 2. Pipeline Stages & Orchestration — 10 Qs

Source → Build → Test → Deploy → Monitor using Jenkins, GitHub Actions, GitLab CI, or Azure DevOps.

- **Typical pipeline stages?** Checkout → Build → Unit → Integration → E2E → Package → Deploy → Verify.
- **How to gate promotions?** Quality gates (tests, coverage, SAST/DAST), manual approvals for prod only.
- **Fan-out/fan-in patterns?** Parallelize independent jobs; aggregate results before promotion.
- **Artifact versioning scheme?** SemVer + build metadata (git SHA, timestamp) for traceability.
- **Idempotent jobs?** Re-running a stage should not double-deploy or corrupt state.
- **Retry vs fail-fast?** Fail fast on critical tests; retry transient infra/network steps.
- **Self-hosted vs cloud runners?** Self-host for control/perf; cloud for elasticity/maintenance offload.
- **Pipeline as code benefits?** Versioned, reviewable, reusable; enables PR validation.
- **Matrix builds?** Strategy to test variants (OS, language, SDK, DB) concurrently.
- **Enforce PR checks?** Block merges until CI passes and reviewers approve.

🧪 3. Automated Testing Integration — 9 Qs

Integrate unit, API, UI, E2E with fail-fast policies.

- **Test pyramid vs trophy?** Favor many fast unit/API tests; minimal critical UI/E2E.
- **Fail-fast strategy?** Stop pipeline on unit/API failures to save compute time.
- **Static vs dynamic test selection?** Run impacted tests by change set for speed (e.g., path-based).
- **Handling flaky tests?** Quarantine, retry with evidence, fix or delete quickly.
- **Contract tests in CI?** Verify provider/consumer schemas before integration.
- **UI test stability?** Use resilient locators, explicit waits, stable test data.
- **API test data setup?** Use fixtures, seeding endpoints, or mocks for determinism.
- **Coverage gates?** Enforce min thresholds but avoid gaming; focus on critical paths.
- **Parallel test hazards?** Shared state collisions; use isolated data/env namespaces.

📦 4. Build Automation & Artifact Management — 9 Qs

Automate builds (Maven/Gradle/npm) and store artifacts in Artifactory/Nexus/GitHub Packages.

- **Reproducible builds?** Pin dependencies; lockfiles; deterministic compilers.
- **Artifact retention policy?** Keep latest N + released tags; purge snapshots by age.
- **SBOM in pipeline?** Generate CycloneDX/SPDX for dependency transparency.
- **Cache strategy?** Cache package managers and build outputs per key (OS+hash).
- **Binary provenance?** Sign artifacts; attach metadata (build id, SHA, SBOM).
- **Promotion model?** Move same artifact from dev→staging→prod; no rebuilds.
- **Monorepo builds?** Incremental builds by changed packages; re-use shared caches.
- **Releases vs snapshots?** Releases immutable; snapshots for ongoing work.
- **Dependency vulnerability handling?** Fail gate on critical CVEs; auto-PR upgrades.

🔐 5. Environment Configuration & Secrets — 9 Qs

Use env vars & secret vaults (Azure Key Vault, AWS Secrets Manager, HashiCorp Vault).

- **12-factor config?** Store config in env, not code; separate build from config.
- **Secret sprawl prevention?** Central vault + short-lived tokens + least privilege.
- **Masking in logs?** CI should redact patterns and block artifact uploads of secrets.
- **Rotating credentials?** Automate rotation and update dependent services/tests.
- **Secure PR previews?** Scoped secrets; ephemeral envs with role-based access.
- **KMS vs Vault?** KMS = key mgmt/crypto; Vault = secret lifecycle + policies.
- **Parameterize test config?** Use env matrices and config files per environment.
- **Secret scanning?** Pre-commit and CI scanners (e.g., trufflehog, Gitleaks).
- **Handling SSO/MFA in CI?** OIDC-based federation for short tokens, no long-lived creds.

🧰 6. Infrastructure as Code (IaC) — 9 Qs

Automate provisioning via Terraform/Ansible/CloudFormation; ensure test env parity.

- **Why IaC for testing?** Consistent, reproducible envs → fewer "works on my machine".
- **Validate IaC changes?** Plan/diff checks, policy-as-code (OPA), unit tests for modules.
- **Ephemeral envs?** Create per-PR envs; destroy on merge to cut costs.
- **Secrets in IaC?** Use references to vaults, never commit plaintext.
- **Drift detection?** Run scheduled plans and alerts when infra diverges.
- **Test data stores?** Seed with fixtures; use snapshots for rollback.
- **Idempotency in Ansible?** Tasks report "changed" only when necessary; safe re-runs.
- **Module versioning?** Pin provider/module versions; renovate with CI PRs.
- **Cost guardrails?** Tags, budgets, and policies to block oversize resources.

🧬 7. Parallel & Distributed Test Execution — 9 Qs

Use Selenium Grid/Playwright parallelism; Dockerized, isolated envs.

- **Horizontal vs vertical scaling?** More nodes vs beefier nodes; choose based on bottleneck.
- **Data isolation?** Unique accounts/DB schemas per worker to avoid flake.
- **Dynamic test sharding?** Balance by historical duration to minimize tail latency.
- **Container start-up cost?** Pre-warm images; cache dependencies; reuse runners.
- **Headless UI scaling?** Tune concurrency; disable video unless debugging.
- **Service dependencies?** Mock/virtualize unstable third-parties during parallel runs.
- **Resource quotas?** CPU/mem caps per job; prevent noisy-neighbor effects.
- **Flake triage at scale?** Tag failures, auto-create issues with traces/logs.
- **When not to parallelize?** Stateful legacy flows; opt for serial suites with clear reasons.

☁️ 8. Containerization & Orchestration — 9 Qs

Package tests with Docker; orchestrate on Kubernetes for scalable CI runners.

- **CI-ready Dockerfile?** Multi-stage build, cache layers, non-root user, tini/entrypoint.
- **Ephemeral runners?** Spin up per job; ensure cleanup with TTL controllers.
- **Test config via K8s?** Use ConfigMaps/Secrets mounted as env/volumes.
- **Stateful test needs?** Use ephemeral PVCs; snapshot/restore if needed.
- **Observability in pods?** Sidecar log agents; scrape metrics; structured logs.
- **Image provenance?** Sign images (cosign); enforce admission policies.
- **Rollout strategies?** Recreate, rolling, canary with HPA for load.
- **Resource requests/limits?** Right-size to avoid throttling or bin-packing issues.
- **Docker-in-Docker caveats?** Prefer buildkit or rootless; mount docker socket carefully.

🧠 9. Quality Gates & Code Analysis — 9 Qs

Integrate SonarQube/ESLint/Checkstyle; enforce coverage and static analysis.

- **Define a "quality gate".** Thresholds (coverage, bugs, vulnerabilities, duplications) to pass.
- **Coverage pitfalls?** High % ≠ high quality; focus on critical path and mutation testing.
- **Static vs dynamic analysis?** SAST finds code issues pre-run; DAST finds runtime vulns.
- **Pull request decoration?** Annotate PRs with findings and block merges when failing.
- **Secret/secret-pattern checks?** Include secret scanners in gate; fail on leaks.
- **Lint as tests?** Treat lint errors as build failures to keep codebase clean.
- **Tech debt visibility?** Track remediation tasks; enforce "no new debt" policy.
- **License policy?** Allow/deny lists; fail builds on incompatible OSS licenses.
- **Mutation testing in CI?** Use tools to verify assertion strength on critical modules.

📊 10. Monitoring & Feedback Loops — 9 Qs

Post-deploy observability with Grafana/Prometheus/New Relic; auto rollback on regression.

- **Release verification?** Smoke tests + SLO checks (latency, error rate, saturation).
- **Automated rollback?** Detect KPI degradation; revert to last good build.
- **Feature flags?** Gate risky changes; enable progressive delivery.
- **Telemetry in tests?** Emit spans/metrics from test harness for triage.
- **Synthetic vs RUM?** Synthetic = scripted probes; RUM = real user telemetry.
- **Error budgets?** Balance velocity vs reliability; freeze deploys if SLOs breach.
- **Change impact analysis?** Correlate deploy id with metrics/log spikes.
- **Alert fatigue control?** SLO-based alerts, deduplication, on-call runbooks.
- **Post-incident reviews?** Blameless RCA, action items, track MTTR improvement.

🚦 11. Version Control & Branching Strategies — 9 Qs

Adopt GitFlow, Trunk-Based, or Feature Branching with PR validation.

- **Protected branches?** Require PR reviews, CI success, signed commits.
- **Release branching?** Cut release branches for stabilization; hotfix from release/main.
- **Feature toggles vs branches?** Prefer toggles to avoid long-running branches.
- **Commit hygiene?** Small atomic commits with useful messages; reference issues.
- **Monorepo branching tips?** Keep PRs scoped; use codeowners and path-based checks.
- **Rebase vs merge?** Rebase for linear history; merge to preserve context.
- **Signed releases?** Tag and sign artifacts/images for auditing.
- **Changelog automation?** Conventional commits + release notes generators.
- **Pre-commit hooks?** Run linters/tests locally to reduce CI churn.

🤖 12. AI-Assisted & Self-Healing Pipelines — 10 Qs

Use AI for flaky test detection, smart selection, and anomaly detection (Launchable, Testim, Mabl, OpenAI-based models).

- **Smart test selection?** Run tests impacted by code diff or risk model to cut time.
- **Flake classification?** Cluster failures by signature/logs to separate env vs test issues.
- **Auto-retries with evidence?** Retry with trace capture; auto-file tickets with artifacts.
- **Anomaly detection in metrics?** Alert on outliers in build duration, failure rate, resource use.
- **Self-healing locators?** ML suggests alternative locators; log confidence & approvals.
- **Risk-based gating?** Block deploy if risk score high despite green basics.
- **COPILOTs for pipelines?** Generate pipeline YAML, IaC modules, and tests with review gates.
- **Data needed for AI efficacy?** Historical test runs, code churn, flake labels, env metadata.
- **Ethics & governance?** Explainability, audit logs, human-in-the-loop for overrides.
- **ROI measurement?** Track minutes saved, flake reduction, MTTR improvement.

Built with ❤️ for fast CI/CD interview prep. Source → Build → Test → Deploy → Monitor.