

Q1. What problem does GitLab CI/CD solve in DevOps or infrastructure?

A: GitLab CI/CD addresses a specific concern provisioning, configuration management, orchestration, or CI/CD by automating manual steps that are error-prone, slow, or not reproducible when done by hand.

Q2. What is GitLab CI/CD's core architecture?

A: GitLab CI/CD has a control plane that stores desired state and a data plane (agents or push-based SSH) that enforces it. Understanding this distinction clarifies where configuration is evaluated and where changes are applied.

Q3. How does GitLab CI/CD define infrastructure or configuration as code?

A: GitLab CI/CD uses declarative files (YAML, HCL, or a DSL) to express desired state. A plan/diff phase shows what will change before applying. Version-controlling these files enables peer review and rollback.

Q4. How does GitLab CI/CD ensure idempotency?

A: GitLab CI/CD operations are designed to produce the same result when run multiple times. Applying the same config twice converges to the desired state rather than creating duplicate resources. This makes automation safe to re-run.

Q5. How do you handle secrets in GitLab CI/CD?

A: Secrets should never be stored in plain text in GitLab CI/CD config files. Use a secrets backend (Vault, AWS Secrets Manager) and inject values at runtime via environment variables or encrypted vaults supported by the tool.

Q6. How does GitLab CI/CD integrate with CI/CD pipelines?

A: CI/CD pipelines call GitLab CI/CD in automated steps: plan on a pull request to preview changes and apply on merge to main. Approval gates can require a human to review the plan before changes go to production.

Q7. How do you debug a failed GitLab CI/CD deployment?

A: Check GitLab CI/CD's logs for the specific error message and the resource that failed. For infrastructure tools, re-run with increased verbosity (-v, --debug). Review the state file or event history to understand what changed before the failure.

Q8. What are security best practices for GitLab CI/CD?

A: Rotate credentials used by GitLab CI/CD, restrict network access to its control plane, enable audit logging of all operations, and use least-privilege service accounts. Never run GitLab CI/CD with admin credentials in production.

Q9. How does GitLab CI/CD scale for large environments?

A: Large GitLab CI/CD deployments use workspaces or namespaces to isolate environments, remote state backends for team collaboration, and modular code to avoid a single monolithic config that is slow to plan/apply.

Q10. What are common mistakes teams make when adopting GitLab CI/CD?

A: Common pitfalls include storing secrets in plain text config, skipping the plan step before apply, not reviewing state drift, and not having a rollback strategy when a deployment fails partway through.