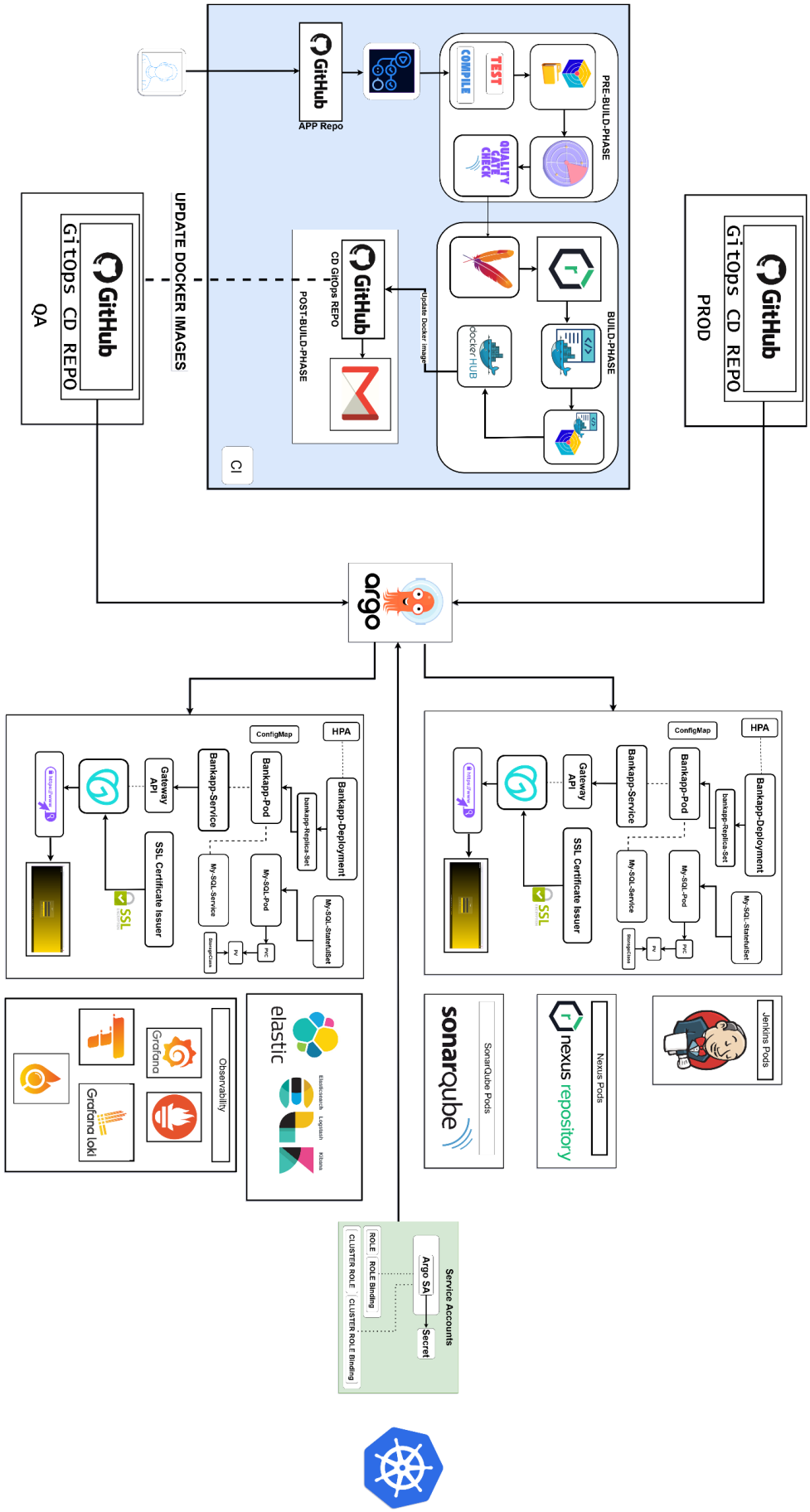
Project Workflow



Enterprise-Grade DevSecOps CI/CD with GitOps (QA & PROD)

0. Why This Architecture Exists (Context First)

In real companies:

- **Code, deployment, and environments are owned by different teams**
- Direct access to production is restricted
- Everything must be:
    - Auditable
    - Reproducible
    - Secure
    - Rollback-friendly

That's why we separate:

- **Application lifecycle (CI)**
- **Environment lifecycle (CD)**
- **QA vs PROD**
- **Automation vs human approval**

---

1. Repository Design – Deep Reasoning

1.1 Application Repository (App Repo)

Purpose

- Single source of truth for application logic

Contains

- Source code
- Dockerfile
- Unit tests
- Build configs
- CI pipeline definition

Does NOT contain

- Kubernetes manifests
- Environment-specific configs

Why?

- Developers should not accidentally change infra
- CI should be repeatable and environment-agnostic

1.2 QA GitOps CD Repository

Purpose

- Represents the *desired state* of QA environment

Contains

- Kubernetes manifests
- QA-specific values:
    - Replica counts
    - Resource limits
    - URLs
    - TLS issuers

Ownership

- DevOps / Platform team

Branch Meaning

- dev: proposed QA state
- main: live QA state

Why separate repo?

- GitOps principle: *Git = Truth*
- Argo CD watches only this repo
- Full audit trail of QA changes

1.3 PROD GitOps CD Repository

Purpose

- Represents *production truth*

Key Rule

CI never touches PROD repo automatically

Why?

- Prevent accidental releases
- Enforce human approval
- Regulatory compliance (SOC2, ISO, PCI)

---

2. Work Starts from Jira (Not Git)

Why Jira is important

- Code should **never start without business context**
- Every change must be traceable

Ticket contains

- Business requirement
- Priority
- Acceptance criteria
- Risk level

This ticket ID is usually:

- Added to commit messages
- Tracked in PR descriptions

---

3. Developer Local Development (Deep View)

Developer responsibilities

- Understand ticket impact
- Pull latest main branch

- Create feature branch

Local checks (ideal practice)

- Unit tests
- Linting
- Local build
- Local Docker build (optional)

Why local checks?

- Save CI resources
- Faster feedback

---

4. CI Pipeline – Detailed Internals

CI pipeline is **stateless** and **re-runnable**.

4.1 Dependency Installation

What happens

- Dependencies downloaded from artifact repos

Why first?

- Everything else depends on it
- Early failure saves time

Failure scenario

- Repo down
- Version mismatch
  → Pipeline fails immediately

4.2 Compilation

Goal

- Validate syntax

- Ensure code builds

Why before security?

- No point scanning broken code

4.3 GitLeaks – Shift-Left Secret Security

What it scans

- Hardcoded secrets
- Tokens
- Credentials

Why before tests?

- Secrets in repo = **critical incident**
- Stop pipeline immediately

Enterprise rule

Any secret leak blocks the release completely

4.4 Unit Testing

Focus

- Business logic
- Code correctness

Outputs

- Test reports
- Coverage metrics

Failure handling

- No deployment without passing tests

4.5 SonarQube Static Analysis

What it checks

- Bugs
- Code smells
- Security vulnerabilities
- Maintainability

Why static analysis here?

- Cheapest place to fix issues
- Before artifact creation

4.6 Quality Gate Enforcement

This is NOT optional

- Gate = policy enforcement

Typical rules

- No critical bugs
- No new security issues
- Coverage threshold

If gate fails

- Pipeline stops
- Developer must fix code

4.7 Artifact Creation

Why artifact before Docker?

- Separation of concerns
- Docker image should consume a *verified artifact*

Example

- Java → JAR

- Node → build bundle

4.8 Nexus Repository Usage

Why Nexus?

- Central artifact store
- Versioned
- Immutable
- Rollback support

Without Nexus

- Docker rebuilds may be inconsistent
- No binary history

4.9 Dependency Vulnerability Scan

Tools

- OWASP Dependency Check
- Trivy FS scan

Purpose

- Detect vulnerable libraries

Important

- Even clean code can be insecure due to dependencies

4.10 Docker Image Build

Uses

- Approved artifact
- Minimal base images

Best practices

- Multi-stage builds

- Non-root users
- Smaller attack surface

4.11 Docker Image Scan (Trivy)

Scans

- OS packages
- App dependencies
- CVEs

Policy

- Critical vulnerabilities → fail
- Medium → warn (org dependent)

4.12 Docker Registry Push

Registry is

- Immutable
- Versioned
- Globally accessible

Image tag strategy

- Commit hash
- Build number
- Semantic version

4.13 Auto-Update QA CD Repo

Why update CD repo from CI?

- CI knows which image is built
- Removes manual errors

What happens

- YAML updated with new tag
- Commit pushed to dev branch

CI ends here.
CI never deploys directly.

---

5. QA Deployment – GitOps Deep Dive

5.1 Pull Request Governance

Why PR even after automation?

- Second layer of safety
- Review YAML changes
- Catch wrong images or configs

5.2 Argo CD Sync Mechanics

Argo CD

- Watches QA CD repo
- Compares:
    - Git desired state
    - Kubernetes live state

Sync types

- Auto-sync or manual
- Drift detection enabled

5.3 Kubernetes Deployment Internals

Resources created

- Deployment
- ReplicaSets

- Services
- Gateway / Ingress
- TLS certificates
- PVCs

Why Gateway API

- Better traffic control
- Future-proof networking

---

6. Post-Deployment QA Validation

6.1 OWASP ZAP (DAST)

Why after deployment?

- Static scans can’t find runtime issues
- DAST simulates real attacker

Finds

- Auth issues
- Input validation flaws
- Headers misconfigurations

6.2 Observability Stack (Why Each Tool Exists)

Prometheus

- Metrics (CPU, RAM, latency)

Grafana

- Visualization
- Alerts

Loki

- Logs without indexing explosion

Tempo

- Request tracing across services

Pyroscope

- Code-level performance profiling

Why all of them together?

- Metrics tell *something is wrong*
- Logs tell *what happened*
- Traces tell *where*
- Profiling tells *why*

---

7. Promotion to Production – Why Manual

QA success ≠ Production readiness

Reasons

- Traffic is higher
- SLAs exist
- Business approval required

---

8. PROD Deployment via GitOps

Same mechanics as QA

- Different repo
- Different configs
- Stricter approvals

Golden rule

Production only changes via Git

---

9. Failure & Rollback Strategy

Rollback via Git

- Revert commit in CD repo
- Argo CD syncs back

Why no kubectl rollback?

- No audit trail
- No consistency
- No compliance

---

10. Interview-Level Summary

This architecture shows:

- Shift-left security
- GitOps maturity
- Zero-trust promotion
- Enterprise compliance
- Production safety