

A DevOps Pipeline:

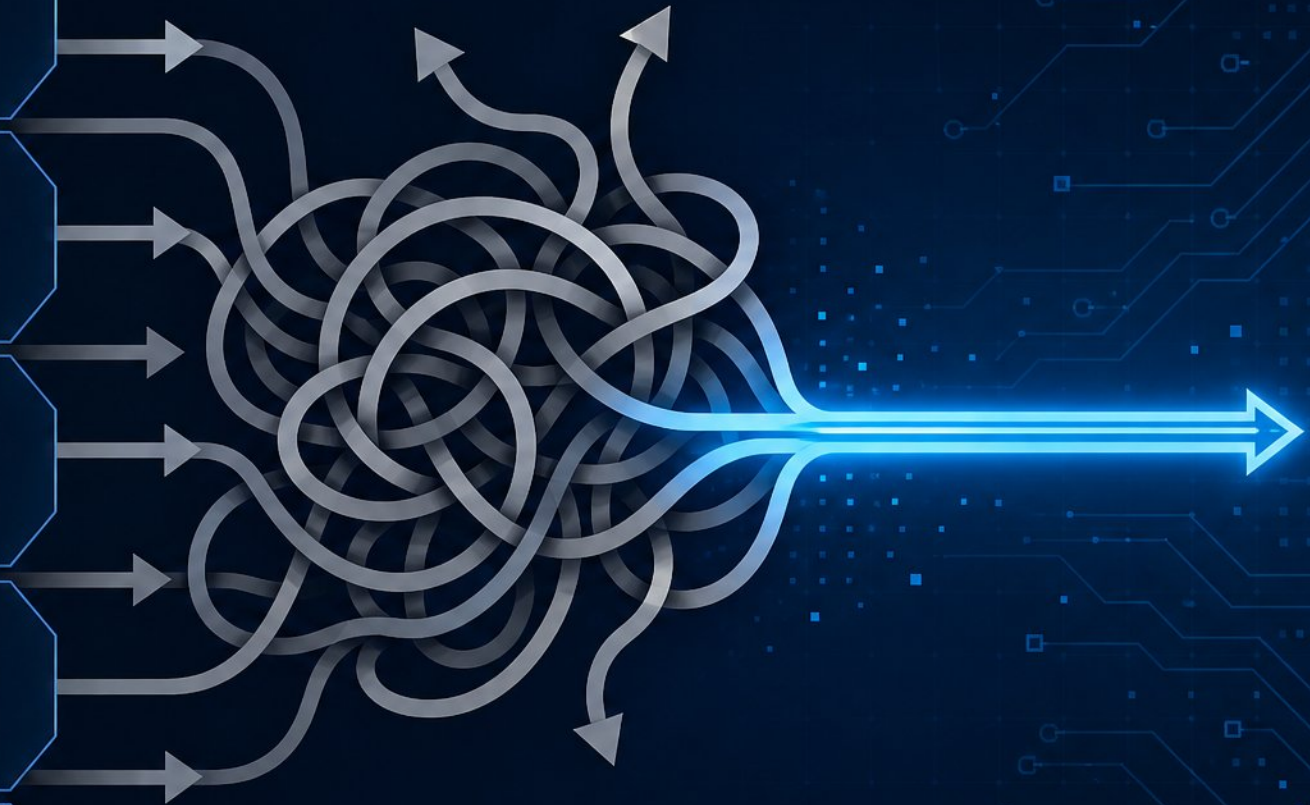
Concepts, Purpose, and Examples



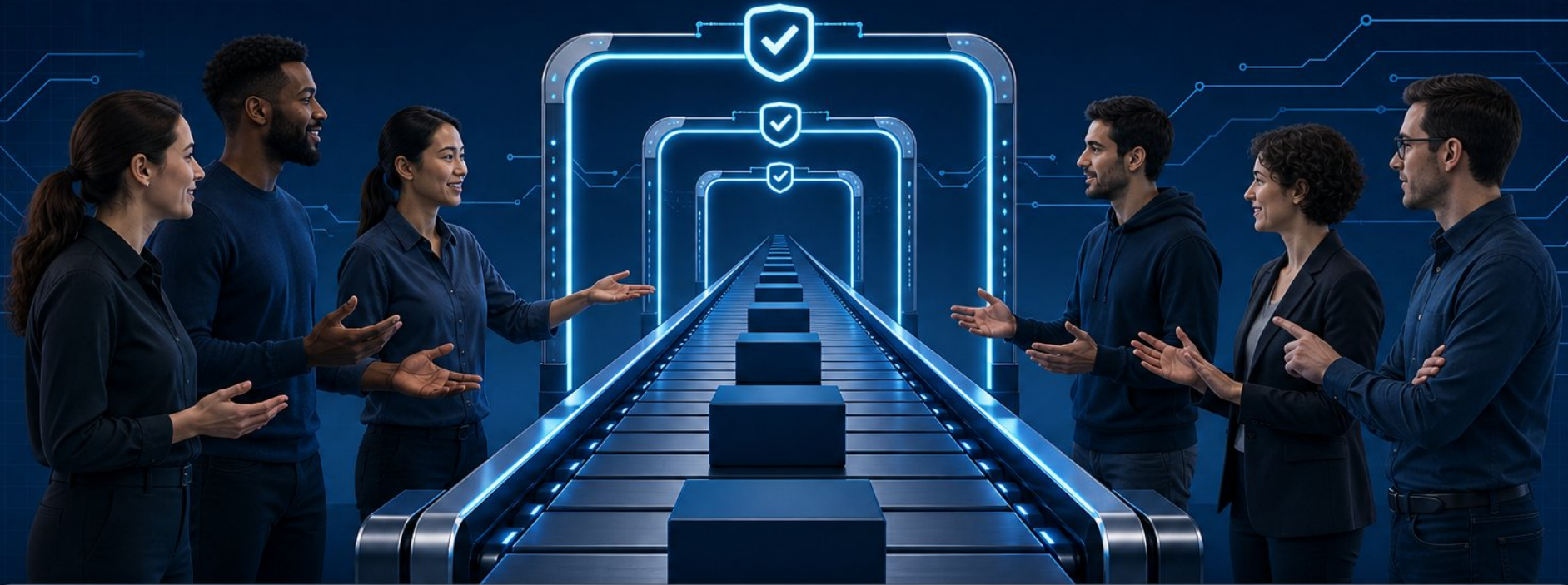
- ▶  - Automated workflow moving code from commit to production with quality and security gates
- ▶  - Solves delivery bottlenecks for developers, operators, and managers
- ▶  - Agenda: core concepts, real-world purposes, and practical examples
- ▶  - Learn to explain pipeline stages, value, and adoption pathways

The Problem Pipelines Solve

- ▶ - Manual handoffs and long release cycles create delivery bottlenecks
- ▶ - Manual steps introduce process defects unrelated to code
- ▶ - Batching changes increases release risk and slows diagnosis
- ▶ - Deployment speed depends on human availability, not code readiness
- ▶ - Slow manual recovery extends the blast radius of production failures



Pipeline Thinking as a Cultural Shift



- Automation moves risk detection earlier in delivery



- Delivery is a system, not isolated tasks



- Automated releases reshape ownership and accountability



- Culture precedes pipeline; principles become practice

Core Concepts: CI, CD, and Continuous Deployment



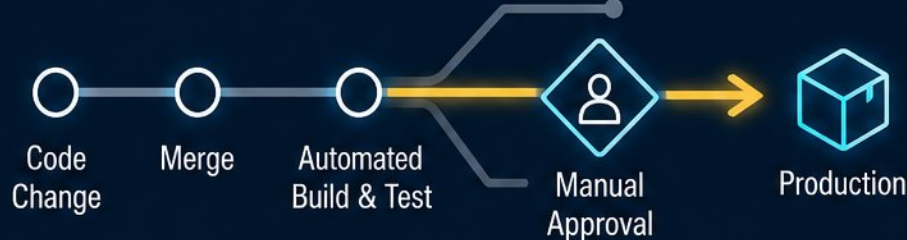
CI

Continuous Integration



CD

Continuous Delivery



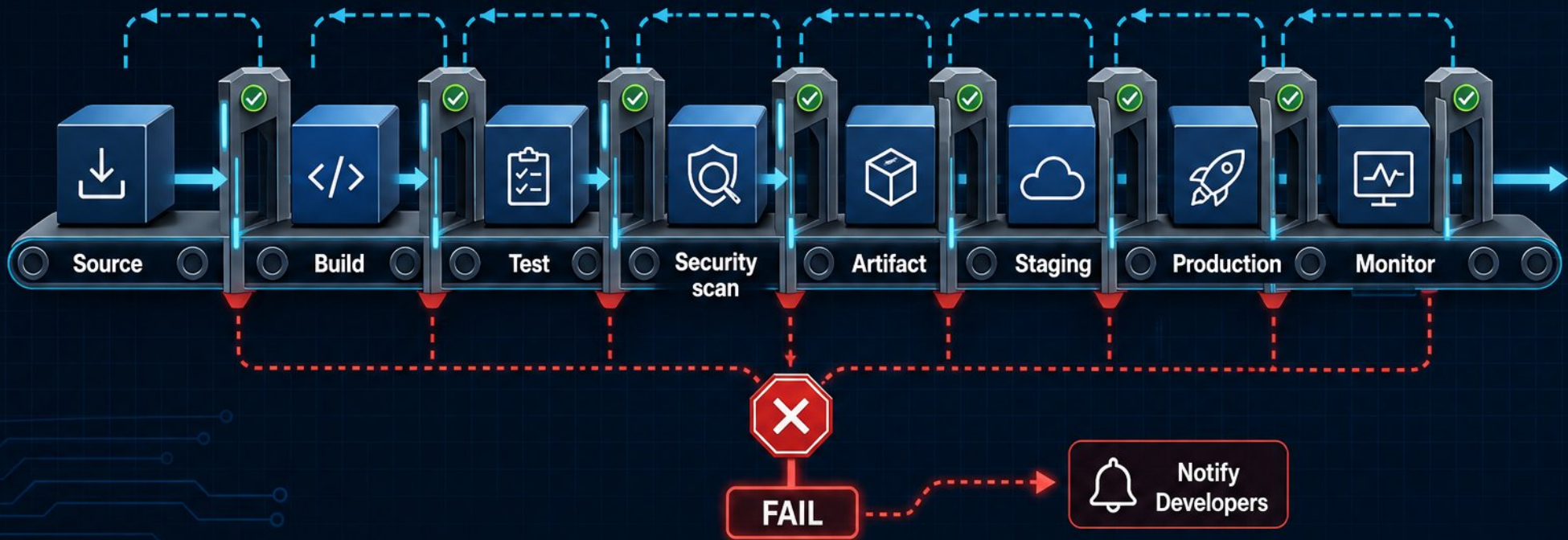
Continuous Deployment



— Gold path = the primary release path — Gray paths = alternate possibilities

- Continuous integration (CI): merge and test changes frequently
- Continuous delivery (CD): every change is deployable; manual approval before production
- Continuous deployment: removes the manual gate; ships every passing change automatically
- The difference is release control, not build quality

Stages, Gates, and Feedback Loops



- Source, build, test, security scan, artifact, staging, production, monitor
- Every stage is a quality gate with a clear pass/fail result
- Failure stops the pipeline immediately and notifies developers
- Automated gates enforce quality and security thresholds without human review
- Feedback loops at each stage enable faster recovery and learning

What a Pipeline Is Actually For



- - Faster safe delivery, not shipping without control



- - Consistency and repeatability across every environment



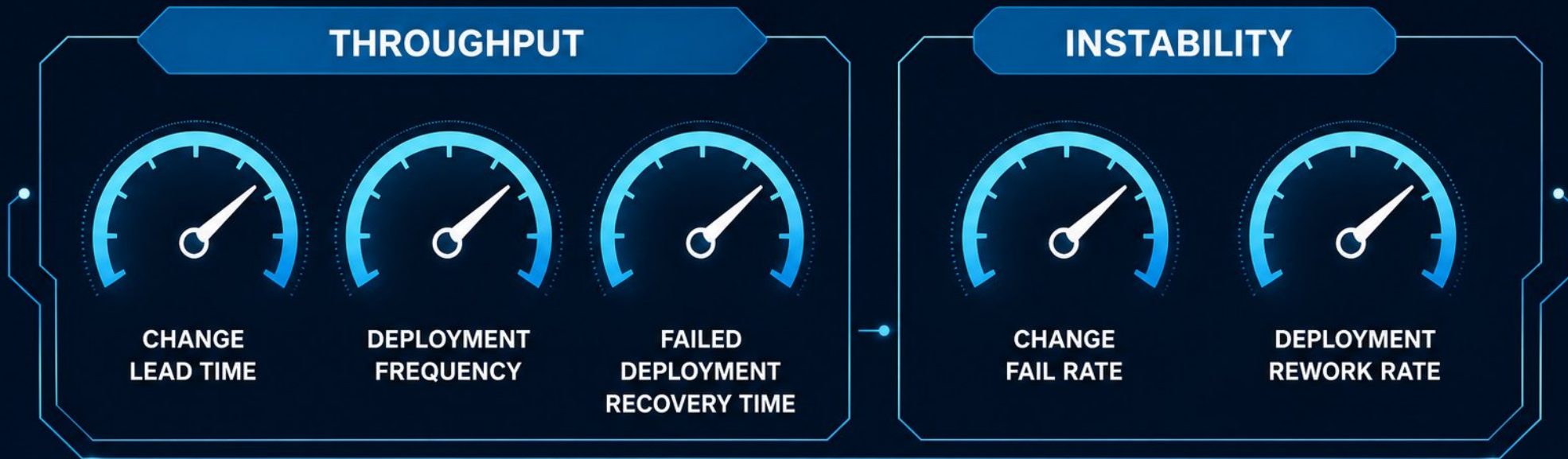
- - Early detection of security, quality, and integration issues



- - Measurable value through DORA metrics: throughput and instability



Measuring Pipeline Value with DORA Metrics



- Five current metrics grouped into throughput and instability
- **Throughput:** change lead time, deployment frequency, failed deployment recovery time
- **Instability:** change fail rate, deployment rework rate
- Read metrics together as a system, not isolated targets

A Simple Working Example



- A small service: build, unit test, container image, deploy, smoke test
- Pipeline catches failing tests, build errors, and security findings
- Developer sees fast feedback in the pull request and pipeline view
- Operator sees deployment status, health checks, and rollback options



Developer

Sees fast feedback in the pull request and pipeline view



Operator

Sees deployment status, health checks, and rollback options

Pipeline Architecture in Practice



- Pipeline-as-code: version-controlled, reviewable, repeatable definitions



- Build once, promote everywhere: immutable artifacts by digest



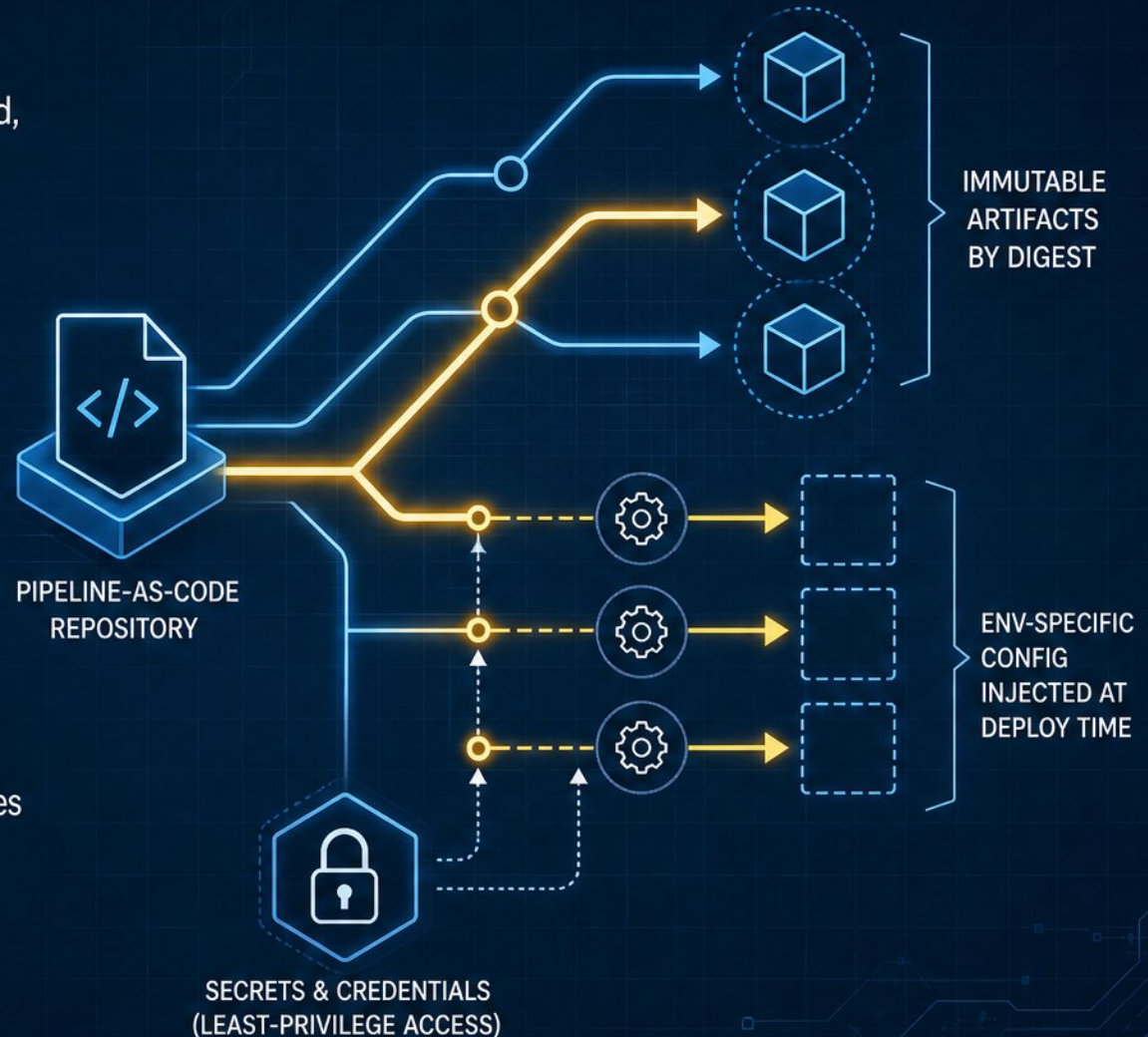
- Inject environment-specific config at deploy time, not build time



- Secrets management, credentials, and least-privilege access boundaries



- Treat pipeline definitions like production code



Platform Teams and Self-Service Delivery



- Platform engineers turn pipelines into internal products with golden paths



- Golden paths make the recommended route the easiest route



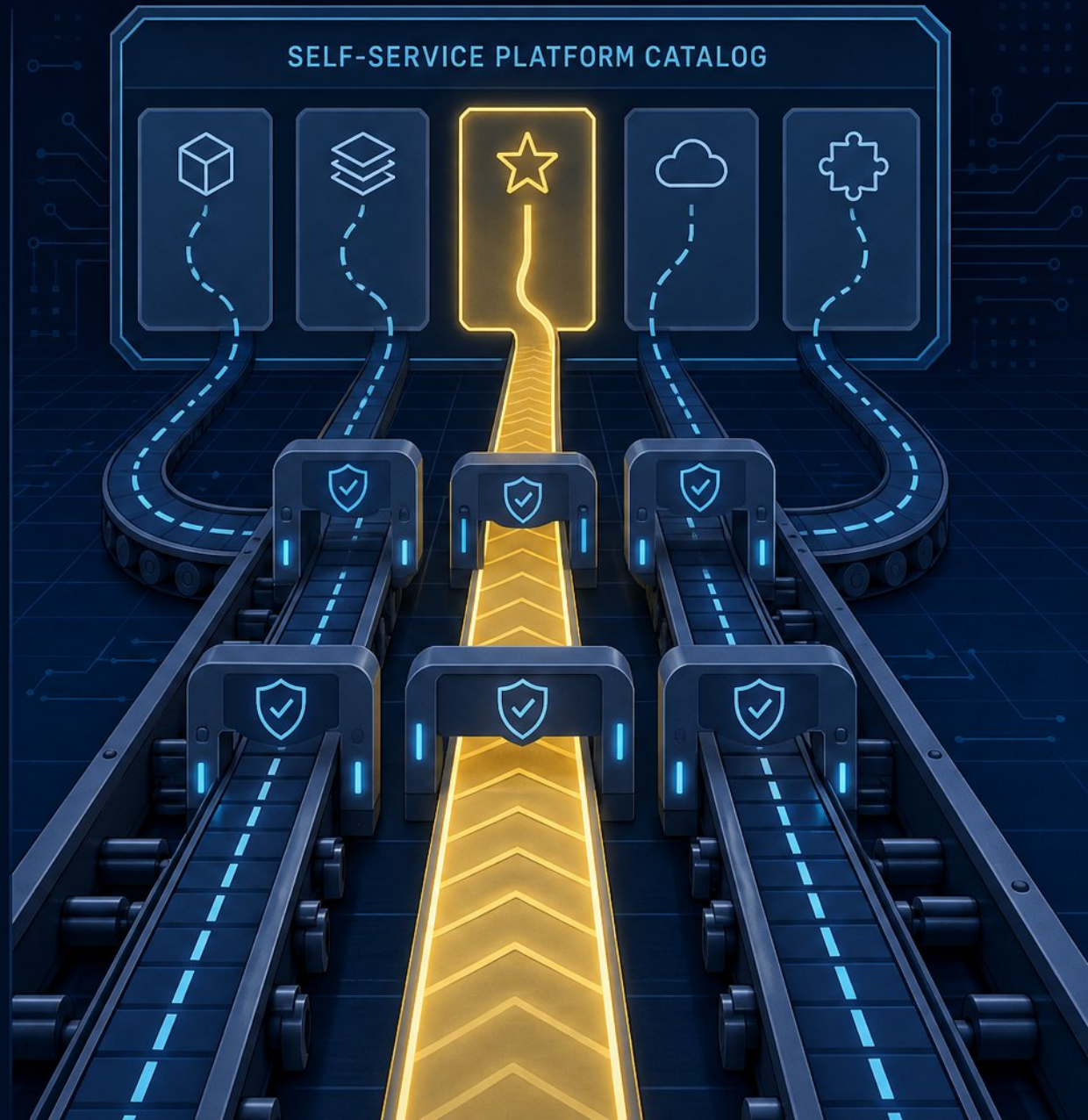
- Teams can diverge for genuine needs, but the paved road is the default



- Policy as code and guardrails replace manual approval bottlenecks



- Product teams focus on code while the platform owns shared delivery logic



Real-World Examples Across Toolchains



- **GitHub Actions:** Git-centric workflows with managed runners and huge marketplace
- **GitLab CI:** integrated platform with built-in security scanning and self-managed options
- **AWS CodePipeline:** cloud-native orchestration tied to AWS identity and infrastructure
- Choose by portability, control, operational overhead, and ecosystem fit
- Match the tool to your source host, cloud depth, and compliance needs

Common Failure Modes



- Flaky tests erode trust and hide real failures



- "Works on my machine": environment drift and undeclared dependencies



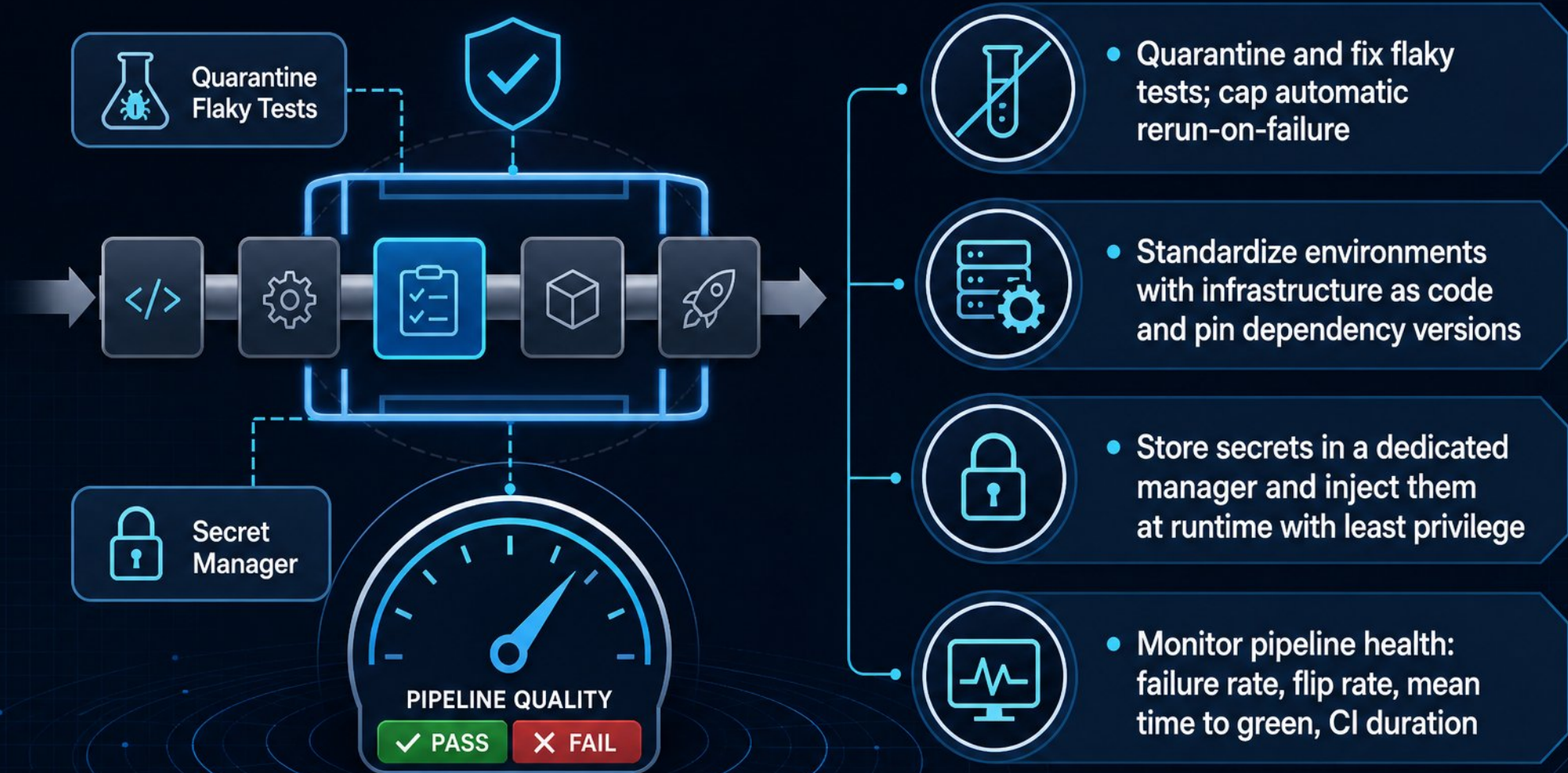
- Hardcoded secrets and configuration drift create security risks



- Slow, overloaded stages push teams to bypass checks



Preventing and Catching Pipeline Deterioration



Getting Started Incrementally



- ▶ Start narrow: one service, a reproducible build, and fast smoke verification
- ▶ Sequence: version control, CI, containers, CD to staging, IaC, observability
- ▶ Run the pipeline in shadow mode before gating releases in unfamiliar codebases
- ▶ Pick the module that changes most often for the fastest feedback wins

Measuring Progress and Supporting Adoption



- Start with DORA throughput and instability metrics plus CI duration and mean time to green



- Build an incremental automation roadmap for both new and existing services



- Balance team autonomy with shared platform standards through golden paths and templates



- Managers invest in platform capability, not forced tool standardization



PersonWise.

PersonWise • AI digital-human course platform



Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/856b1255/public