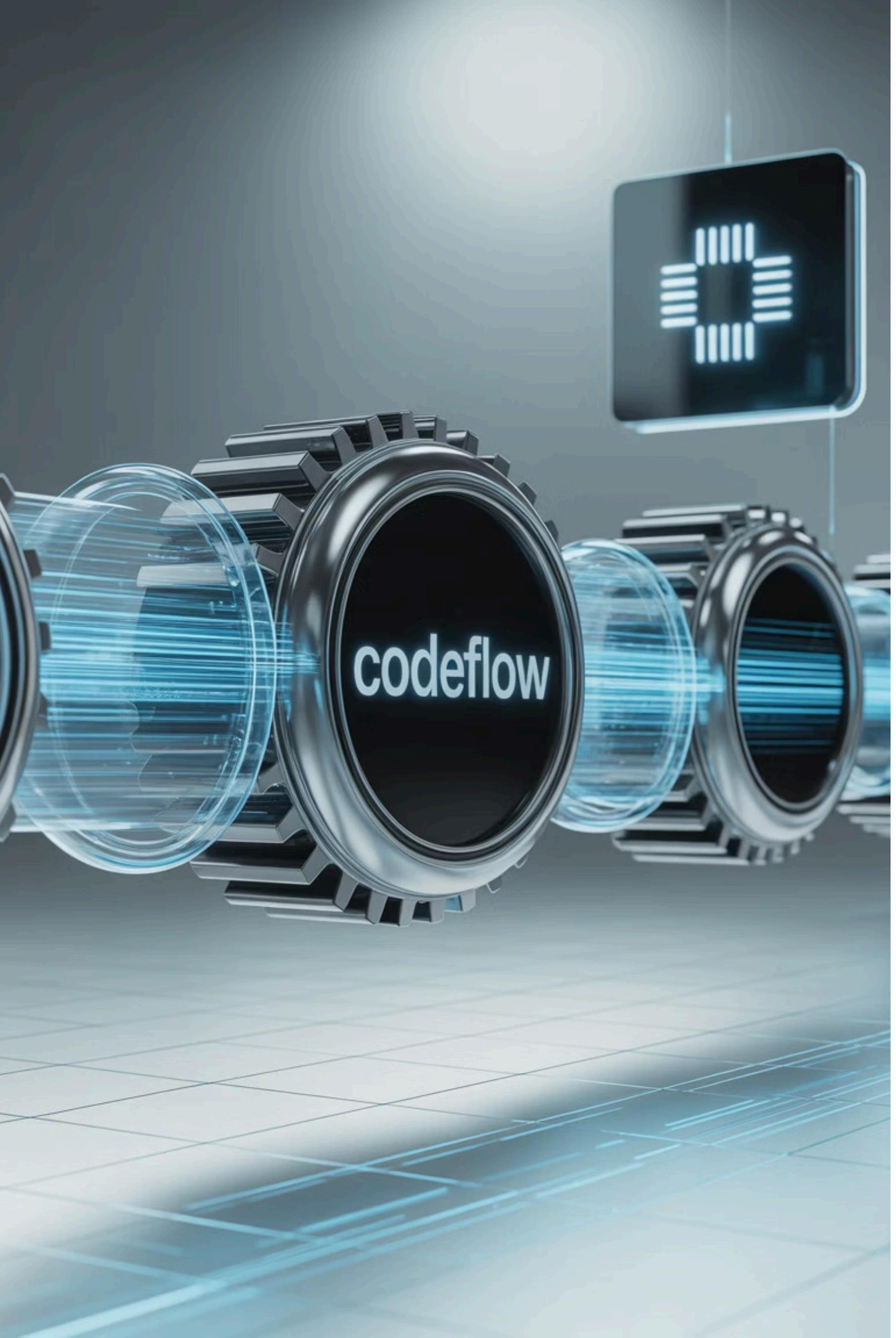


Continuous Integration and Continuous Deployment (CI/CD)

Transforming how software teams build, test, and deliver applications through automated pipelines and streamlined workflows.

The Story Behind CI/CD

The Old Way

Manual builds taking hours or days, deployment fears keeping teams up at night, integration surprises after weeks of isolated work, and stressed developers scrambling to fix broken builds right before release deadlines.

The CI/CD Revolution

Automated builds completing in minutes, confident deployments happening multiple times daily, continuous integration catching issues early, and developers focusing on creating value instead of managing deployment chaos.

Why CI/CD Matters Today

In today's fast-paced digital landscape, companies like Netflix deploy code thousands of times per day. Amazon makes changes every 11.7 seconds. These aren't just impressive statistics—they represent a fundamental shift in how successful organizations deliver software.

CI/CD pipelines are the invisible infrastructure powering modern software delivery, enabling teams to move from idea to production with unprecedented speed and reliability.

Designing and Implementing CI/CD Pipelines

Building the foundation for automated software delivery

The Anatomy of a CI/CD Pipeline

Source Control

Developer commits code to repository, triggering the pipeline automatically

Build

Code is compiled, dependencies resolved, and artifacts created

Test

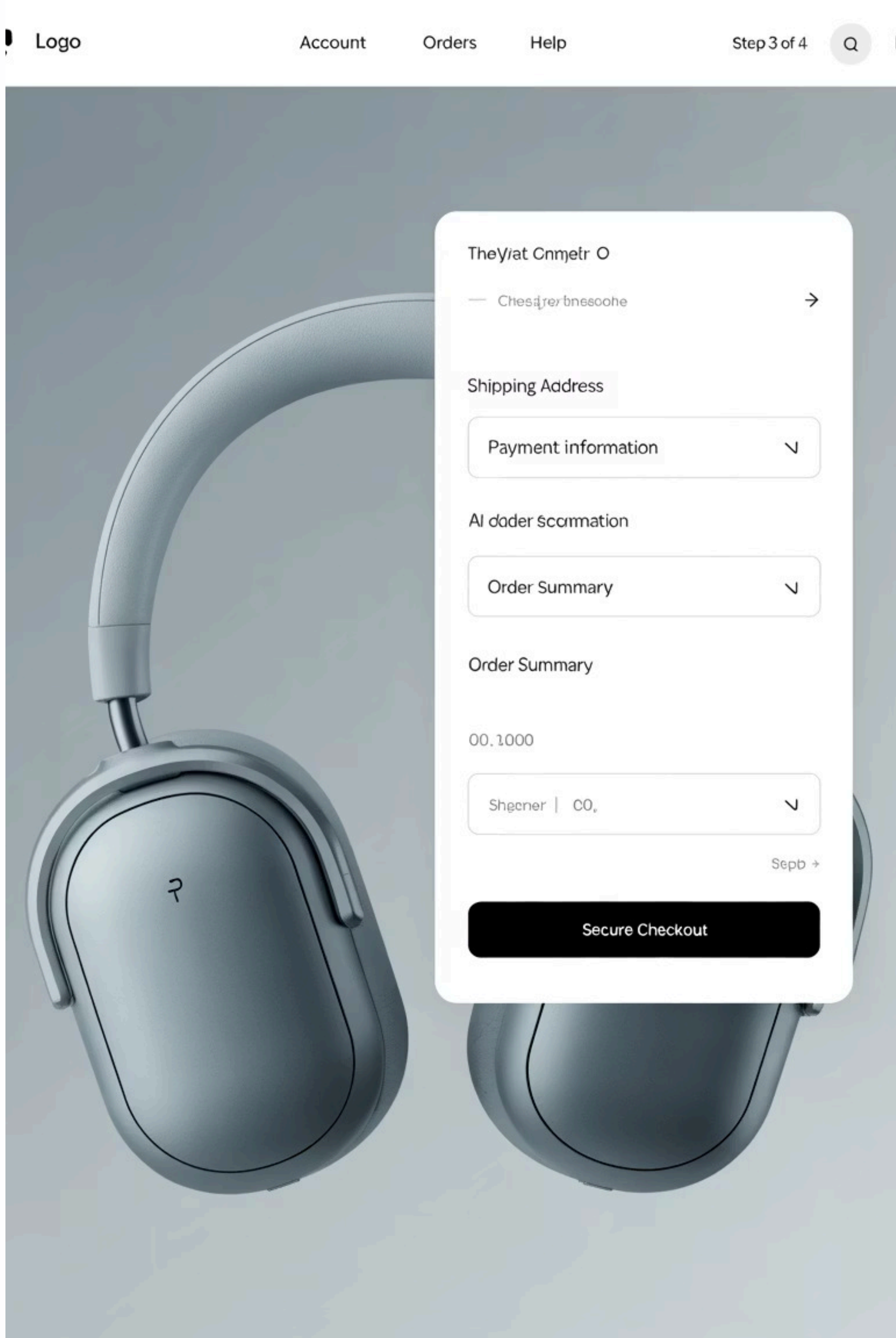
Automated tests validate functionality, security, and performance

Deploy

Validated code automatically deploys to target environments

Real-World Example: E-Commerce Platform

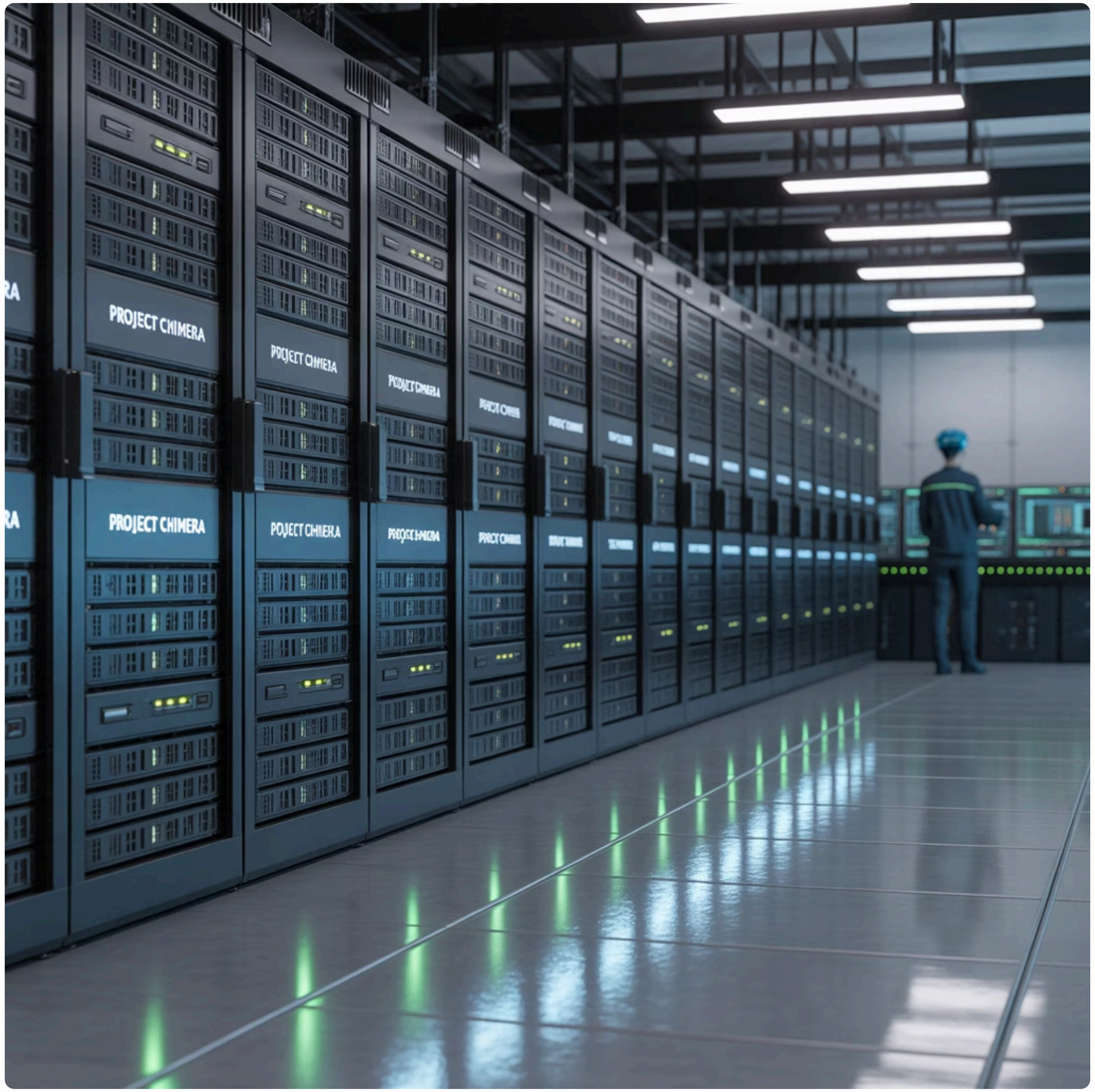
Consider an e-commerce company processing thousands of orders daily. Their CI/CD pipeline automatically builds and tests every code change, from payment processing updates to UI improvements. When a developer fixes a checkout bug at 2 PM, automated tests verify the fix works correctly, and by 2:30 PM, customers are already benefiting from the improvement—all without manual intervention.



Configuring Build Agents and Agent Pools

What Are Build Agents?

Think of build agents as specialized workers in a factory. Each agent is a machine (physical or virtual) that executes your pipeline tasks—compiling code, running tests, or creating deployment packages.



Agent Pools: Organizing Your Build Resources

Windows Pool

Dedicated to .NET applications and Windows-specific builds

- Visual Studio installations
- Windows SDK tools
- PowerShell capabilities

Linux Pool

Optimized for containerized applications and microservices

- Docker support
- Kubernetes tools
- Shell scripting

macOS Pool

Essential for iOS and macOS application builds

- Xcode environment
- iOS simulators
- Swift compiler

Microsoft-Hosted vs. Self-Hosted Agents

Microsoft-Hosted Agents

Best for: Getting started quickly, small to medium projects

Example: A startup building a mobile app uses Microsoft-hosted agents to avoid infrastructure costs. They get pre-configured environments with all common tools installed, perfect for their 5-person development team.

Self-Hosted Agents

Best for: Enterprise needs, specialized tools, security requirements

Example: A healthcare company uses self-hosted agents behind their firewall to maintain HIPAA compliance. Their custom security scanning tools run on these dedicated machines, ensuring patient data never leaves their controlled environment.

Scaling Your Build Infrastructure

5

Small Team

Agents sufficient for startups with sequential builds

20

Growing Company

Agents needed for parallel builds across multiple teams

100+

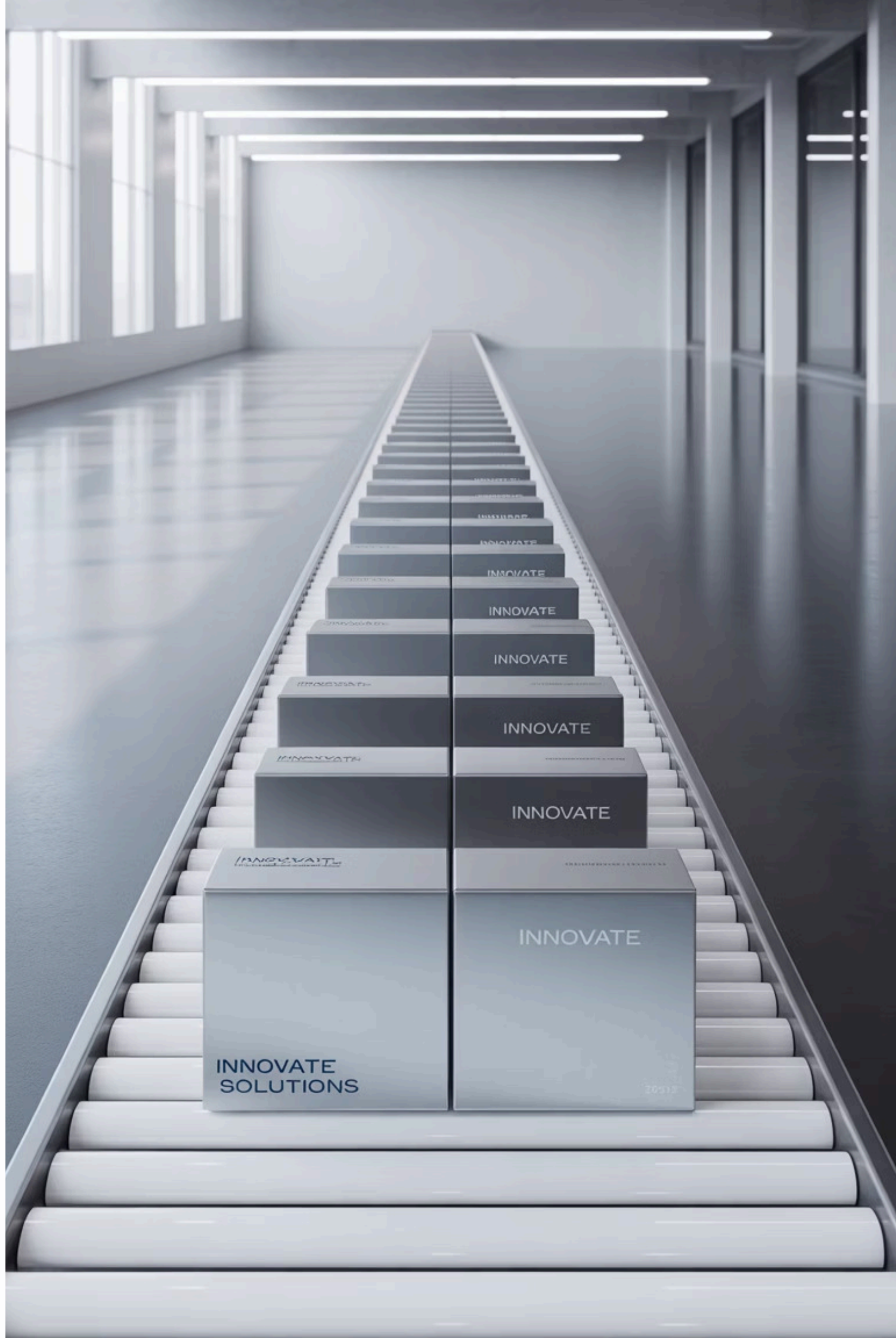
Enterprise Scale

Agents required for large organizations with continuous deployment

A financial services company started with 5 agents. As they grew from 50 to 500 developers, they scaled to 150 agents across multiple pools, ensuring every team could build and deploy without waiting in queue.

Building and Packaging Applications

The build process transforms your source code into deployable artifacts. This isn't just compilation—it's about creating consistent, reproducible packages that can be deployed with confidence across any environment.



The Build Process in Action

01

Retrieve Source Code

Pipeline pulls latest code from Git repository, ensuring all team changes are included

02

Restore Dependencies

NuGet, npm, or Maven packages are downloaded and cached for faster subsequent builds

03

Compile Application

Source code is compiled into binaries, with compiler optimizations applied for production

04

Run Unit Tests

Automated tests verify code quality, catching issues before they reach production

05

Create Artifacts

Deployable packages are generated, versioned, and stored for deployment

06

Publish Results

Build artifacts are uploaded to artifact repository, ready for release pipeline

Real Example: Building a .NET Web Application

```
trigger:
  branches:
    include:
      - main
      - develop

pool:
  vmImage: 'windows-latest'

steps:
- task: NuGetRestore@1
- task: VSBuild@1
  inputs:
    solution: '**/*.sln'
    configuration: 'Release'
- task: VSTest@2
  inputs:
    testAssemblyVer2: |
      **\*test*.dll
- task: PublishBuildArtifacts@1
```

What This Pipeline Does

Every time code is pushed to the main branch, this pipeline automatically restores packages, builds the solution in Release configuration, runs all tests, and publishes the compiled application as an artifact.

Real impact: A financial software company uses this pattern to build their trading platform 50+ times daily, catching integration issues within minutes instead of days.

Packaging Strategies for Different Application Types

Container Images

Docker containers package application with all dependencies, ensuring "runs on my machine" becomes "runs everywhere." Perfect for microservices and cloud-native apps.

ZIP Archives

Traditional packaging for web applications and services. Simple, reliable, and works with standard deployment tools like Azure App Service.

NuGet Packages

Ideal for libraries and reusable components. Enables version management and dependency resolution across multiple projects.

Versioning Your Builds

Every build needs a unique version number to track what's deployed where. Semantic versioning (like 2.5.1) tells a story: major version 2, minor version 5, patch 1.

Major Version (2.x.x)

Breaking changes that require code updates. Example: Your API changes from REST to GraphQL—that's major version 3.0.0

Minor Version (x.5.x)

New features that don't break existing functionality. Adding a new endpoint to your API bumps to 2.5.0

Patch Version (x.x.1)

Bug fixes and small improvements. Fixing that checkout button issue becomes 2.5.1

Customizing and Extending Build Pipelines

Out-of-the-box pipelines are great for starting, but real power comes from customization. Think of your pipeline as a recipe—the basic ingredients are there, but you add your own special touches to make it perfect for your needs.

Custom Build Tasks: Adding Your Secret Sauce

Security Scanning

A banking app adds WhiteSource to scan for vulnerable dependencies before every build. One day, it catches a critical security flaw in a popular library, preventing a potential data breach.

Code Quality Gates

An insurance company integrates SonarQube to enforce code quality standards. Builds fail if code coverage drops below 80% or if critical bugs are detected, maintaining high quality standards.

Custom Notifications

A gaming studio sends build results to their Discord channel with memes for success and failure, making pipeline monitoring fun and keeping the team engaged.

Pipeline Templates: Don't Repeat Yourself

The Problem

Your company has 50 microservices, each needing the same build steps: restore, build, test, scan, package. Maintaining 50 identical pipeline configurations is a nightmare.

The Solution

Create a reusable template once, reference it from all pipelines. Update security scanning in one place, and all 50 services benefit immediately.

Environment Variables and Parameters

Development

Database: dev-sql.company.com

API: https://dev-api.company.com

Debug logging: Enabled

Staging

Database: stage-sql.company.com

API: https://stage-api.company.com

Debug logging: Enabled

Production

Database: prod-sql.company.com

API: https://api.company.com

Debug logging: Disabled

One build, multiple configurations. Your pipeline uses parameters to customize behavior for each environment, building once and deploying everywhere with environment-specific settings.

Release Management with Azure Pipelines

Transforming builds into production deployments with confidence and control

From Build to Release: The Journey

Building your application is just the beginning. Release management is the disciplined process of moving your code from "it works on my machine" to "it's serving customers in production." This journey requires careful orchestration, multiple checkpoints, and safety measures at every stage.

Designing Release Pipelines

1. **Build Complete**

   Artifact created and validated, ready for deployment journey

2. **Dev Deploy**

   Automatic deployment for immediate developer validation

3. **QA Testing**

   QA team verifies functionality in controlled environment

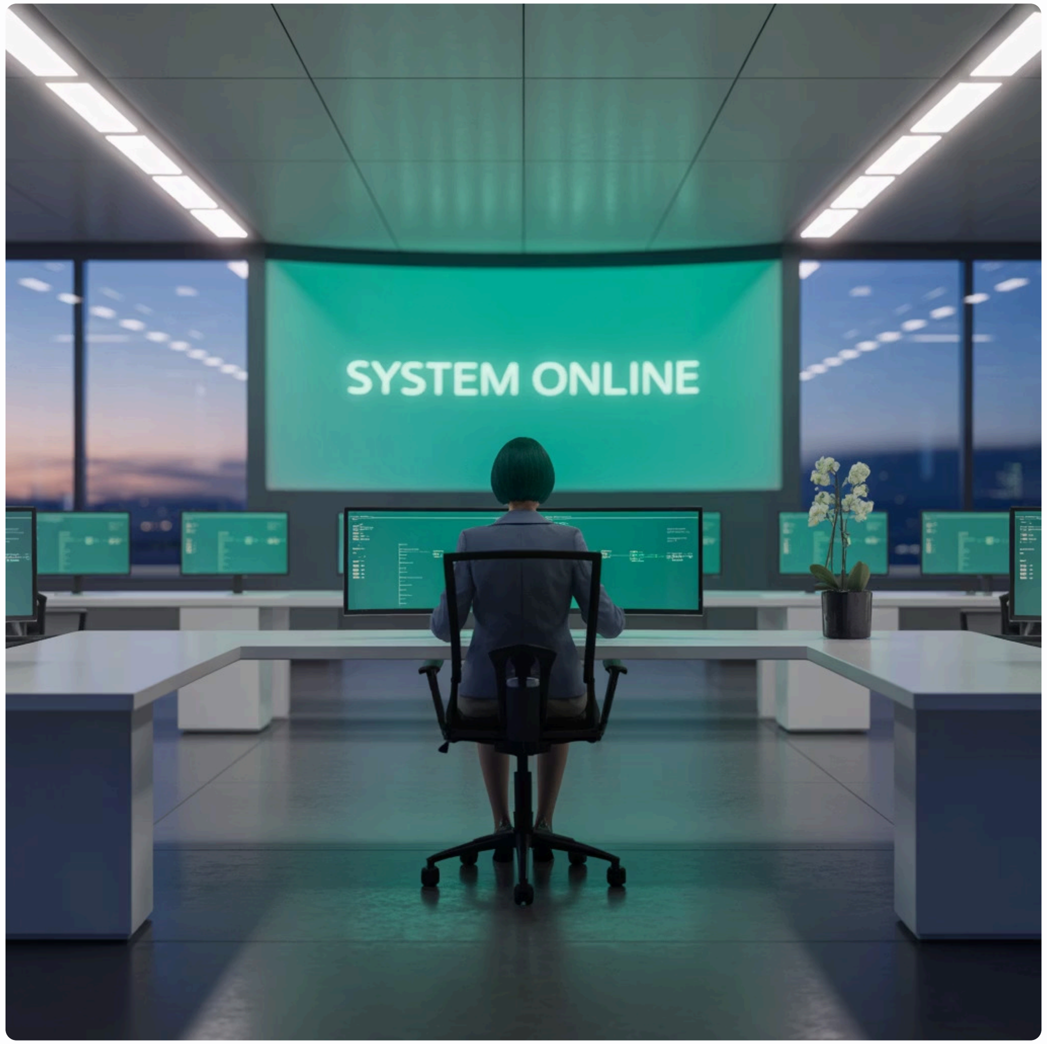
4. **Staging Review**

   Final pre-production validation with production-like data

5. **Production**

   Controlled rollout to customers with monitoring

Real-World Release Pipeline Example



E-Commerce Platform

A major retailer processes $10M in daily transactions. Their release pipeline deploys to dev automatically, then requires QA sign-off before staging. Production deployments happen during low-traffic windows (2-4 AM) with automatic rollback if error rates spike above 0.1%.

Result: 50+ production deployments monthly with 99.99% success rate and zero customer-impacting incidents in the past year.

Deploying to Different Environments

Development Environment

Deployed automatically after every successful build. Developers can immediately test their changes. Database contains synthetic test data. Monitoring is minimal—focus is on rapid feedback.

QA Environment

Deployed on-demand or on schedule. QA team performs exploratory and automated testing. Includes test data covering edge cases. Integration with third-party services uses sandbox accounts.

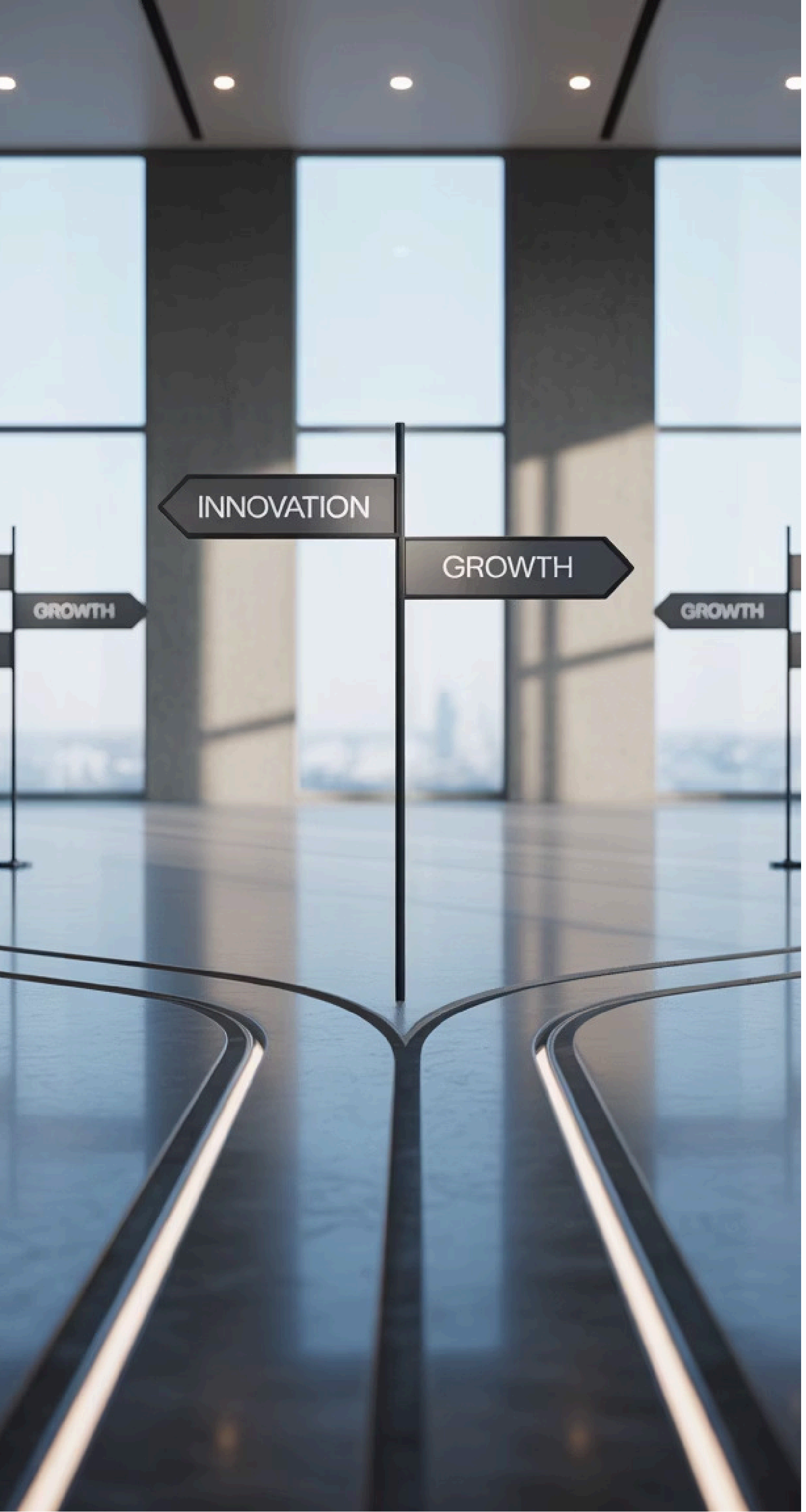
Staging Environment

Mirror of production with anonymized production data. Performance testing and security scans run here. Final validation before production. Stakeholder demos happen in this environment.

Production Environment

Customer-facing deployment with comprehensive monitoring. Gradual rollout to minimize risk. Automated alerts for any anomalies. Rollback capabilities ready at all times.

Deployment Strategies: Choosing Your Approach



Big Bang Deployment

When: Small applications, maintenance windows available

Example: Internal HR system deployed Sunday night, 30-minute downtime acceptable

Rolling Deployment

When: Load-balanced applications, zero downtime required

Example: Social media platform updates 10 servers at a time, maintaining 90% capacity

Blue-Green Deployment

When: Instant rollback critical, database migrations complex

Example: Payment processor maintains two identical environments, switches with DNS update

Canary Deployment

When: High-traffic applications, gradual risk mitigation needed

Example: Streaming service deploys to 1% of users first, monitors metrics, then expands gradually

Your CI/CD Journey Starts Now

Key Takeaways

Start Simple, Scale Smart

Begin with basic pipelines and add sophistication as your needs grow. Even a simple automated build is better than manual processes.

Automate Everything You Can

Every manual step is an opportunity for error. Invest in automation today to save countless hours tomorrow.

Build Safety Into Your Process

Approvals, gates, and testing aren't obstacles—they're insurance policies that let you move faster with confidence.

Measure and Improve Continuously

Track deployment frequency, lead time, and failure rates. Use data to identify bottlenecks and optimize your pipelines.

The companies winning today aren't necessarily the ones with the best developers—they're the ones with the best pipelines. Start building yours today.