Brought to you by:

harness

Next-Generation CI/CD

for dummies®

A Wiley Brand

Automate and accelerate your pipelines

Enforce governance and security best practices

Optimize software delivery with AI

Harness Special Edition

Kenneth Hess

About Harness

Harness is the leading end-to-end AI-native platform for complete software delivery. It provides a simple, safe, and secure way for engineering and DevOps teams to release applications into production. Harness uses AI and machine learning to monitor the quality of deployments and automatically roll back failed ones, saving time and reducing the need for custom scripting and manual oversight, giving engineers their nights and weekends back. Harness customers accelerate deployments by up to 75 percent, reduce infrastructure costs by up to 60 percent, and decrease lead time for changes by up to 90 percent. Harness is based in San Francisco.

Next-Generation CI/CD

Harness Special Edition

by Kenneth Hess

for dummies®
A Wiley Brand

Next-Generation CI/CD For Dummies®, Harness Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN 978-1-394-31327-3 (pbk); ISBN 978-1-394-31328-0 (ebk); ISBN 978-1-394-31329-7 (ebk)

Publisher's Acknowledgments

Editor: Elizabeth Kuball
Acquisitions Editor: Traci Martin
Senior Managing Editor: Rev Mengle
Client Account Manager: Jeremith Coward
Production Editor: Tamilmani Varadharaj

Table of Contents

Introduction

Continuous integration/continuous delivery (CI/CD) is about pushing frequent code changes into production via a reliable and consistent system of tools and procedures. It allows incremental changes to be added to a production application, alleviating the need to wait for a major revision, especially when providing updates based on user feedback.

When done well, CI/CD streamlines development, reduces software flaws and bugs, increases delivery efficiency, and lowers testing and development costs. But CI/CD practices face hurdles, including tool integration, infrastructure complexity, and the goal of balancing speed with quality and security. Integrating with various tools — such as version control, build systems, application security tools and deployment tools — can be challenging, especially when dealing with diverse technologies or legacy systems.

This book explains how to achieve engineering excellence and improve the developer experience using an artificial intelligence (AI)–native software delivery platform.

About This Book

Next-Generation CI/CD For Dummies, Harness Special Edition, consists of five chapters that explore the following:

- Modernizing your DevOps processes (Chapter 1)
- Optimizing your continuous integration efforts (Chapter 2)
- Streamlining continuous delivery using next-generation CI/CD (Chapter 3)
- Standardization and governance (Chapter 4)
- Shaping the future of CI/CD (Chapter 5)

Each chapter is written to stand on its own, so if you see a topic that piques your interest feel free to jump ahead to that chapter. You can read this book in any order that suits you.

Foolish Assumptions

It has been said that most assumptions have outlived their uselessness, but I assume a few things nonetheless!

Primarily, I assume you have some experience with CI/CD as a developer, code integrator, or operations team member. Because you're reading this book, I also assume you're having a less-than-optimal experience with your current software solution. You could also be searching for a new approach to CI/CD — a next-generation, AI-assisted integration and delivery platform.

If any of these assumptions describes you, this book is for you! If none of these assumptions describes you, keep reading anyway — it's a great book, and you'll expand your knowledge of next-generation CI/CD by reading it!

Icons Used in This Book

Throughout this book, I use special icons to call attention to important information. Here's what to expect:


REMEMBER

The Remember icon points out important information you should commit to your nonvolatile memory, your gray matter, or your noggin.


TECHNICAL STUFF

The Technical Stuff icon explains the jargon beneath the jargon and is the stuff legends — well, legendary nerds — are made of.


TIP

Tips are appreciated, but never expected — and I sure hope you'll appreciate these useful nuggets of information.


WARNING

These alerts point out the stuff your mother warned you about. Well, probably not, but they do offer practical advice.

Beyond the Book

There's only so much I can cover in this short book, so if you find yourself at the end of this book wondering, "Where can I learn more?" go to `www.harness.io` for more information.

IN THIS CHAPTER

- Defining CI/CD
- Improving on current CI/CD practices
- Leveraging AI for software delivery
- Exploring feature flags

Chapter 1
Modernizing Your DevOps Processes

The primary goal of continuous integration/continuous delivery (CI/CD) is to streamline and accelerate the software development life cycle (SDLC). CI starts with developers frequently integrating changes into a source code repository. Those changes are then automatically built and tested. CD releases new code into production.

CI/CD helps organizations avoid bugs, code failures, and security vulnerabilities while maintaining continuous updates in the SDLC, decreasing complexity, increasing efficiency, and streamlining workflows.

In this chapter, I cover the basic definition of CI/CD, current problems with existing CI/CD technologies, using artificial intelligence (AI) for software delivery, and a brief introduction to feature flags.

Identifying Problems with Current DevOps Processes

If you've worked in DevOps, you know that currently accepted CI/CD strategies, solutions, and tools have many problems and shortcomings. This section exposes the three most common DevOps pain points and their solutions.

Maintaining security throughout the development life cycle

Working in an organization that uses a CI/CD model can make it challenging to deliver secure applications because of the frequency of changes and deployments. Integrating security throughout the development process — especially in the SDLC's planning, analysis, and design stages — is critical to delivering secure code.

REMEMBER

The earlier in the development process you focus on security, the better your overall application security posture will be. This is called "shifting security left" in DevSecOps speak.

Developers can easily fix security vulnerabilities when they're found early and prioritized and contextualized, along with prescriptive remediation guidance. Many vulnerabilities can now be automatically remediated using AI.

Increasing complexity with microservices

Microservices can push the limits of your pipelines because of the velocity and independence needed to deploy successfully. You may ask, "Why do microservices increase complexity when they should *decrease* complexity?" As dependency complexity is reduced, deployment complexity is increased.

Smaller and more granular services increase the number of deployed services. The sheer number of deployments and managing dozens or hundreds of service components makes tracking dependencies between different services more challenging.

This explosion of microservices also elevates the complexity of troubleshooting, monitoring, logging, and overall system health

checks. The solution is to use more robust tools that simplify creating and managing these new pipelines.

Selecting the best DevOps tools

Making a wise choice is difficult because of the complexity and rapid service growth coupled with a competitive DevOps tools market. The success or failure of development and deployment is often a function of the tools an organization chooses. Selecting inferior or insufficient DevOps software causes developers to create their own generally unsupportable tool suites that they cherry-pick from various sources. This can be costly for an organization, leading to developer toil and missing product deadlines.


TIP

To overcome this challenge, DevOps teams should select an integrated toolset that includes version control, CI, testing, deployment, application security testing integrations, and monitoring capabilities. The suite should also be flexible, security-focused, relatively simple or intuitive, and capable of integrating into your current workflows and organizational structure.

Changing CI/CD Processes

To end the CI/CD process failure cycle, you must change your development culture to focus on end-to-end security, continuous testing and monitoring, and performance tracking. The short-answer solution to resolving failures is to implement an automated testing framework.

CI enables automated builds and tests. A robust ecosystem and various plug-ins make integrating modern testing methodologies and new languages easy. Pipelines, such as automated builds and tests, can be configured as code and are declarative, expressing goals instead of lengthy scripting.

The primary practice in a CI pipeline is to automate the build process. Within a build process, you want to ensure you're integrating and performing unit, functional, integration, and security tests so that builds fail for code that doesn't meet functional requirements. Build and test automation is about integrating changes early and often. This prevents maintenance complexity on feature and main branches as developers progress on feature development.

Change isn't easy, but moving toward a more efficient CI/CD process is essential by automating processes that can avoid human error, prevent vulnerabilities, and speed up delivery.

Harnessing the Power of Feature Flags

Feature flags allow organizations to deliver features faster, with less risk. Companies constantly develop new software features for their customers. Traditionally, these features are made available via a software deployment and become visible to all users simultaneously. As a result, every time a new feature is deployed, customers risk having a bad experience due to a poorly implemented feature. The only way to fix the problem is to roll back to the prior version immediately or to create a fix as soon as possible and roll forward by deploying a new version.

The risks associated with new feature releases slow developers, who must thoroughly validate design and implementation before releasing. Multiple possible versions of a new feature's look and function often exist, and it's up to the developer to choose and implement one.

Sometimes, the developer's choice isn't the correct one. Developers often decide when to release new software features instead of relying on business needs. Sunsetting old features introduces risks to newer features that may depend on the old code. When teams depend on engineering release cycles (for example, once a month), multiple features are released simultaneously, introducing complexity and risk that can result in deployment war rooms, large rollbacks, and dissatisfied customers.


TECHNICAL STUFF

Feature flags are used as software switches to hide or activate features. Developers can turn features on or off without changing the source code or deploying new code to apply or remove a feature from a software deployment.

Feature flags are used to toggle new features on or off for a subset of users for testing. However, beyond this simple use case, advanced capabilities are available through Harness's unique feature flag tools.

Simple UI-based feature release workflows

Customers can create templates and processes to standardize across feature flags with the same operational needs. Feature flags can be included in a visual pipeline that includes steps related to governance and verification.

Governance and verification

Customers can ensure that production pushes always meet defined organizational standards and minimize the negative impact of any issues in production. Harness allows customers to create controls in their feature release process, including mandating approvals and creating audit trails. In addition, customers can automate service verification after a feature is live, ensuring that if an issue occurs, the feature is turned off to minimize impact.

Integration into CI/CD

Many feature flag tools separate feature flags from the rest of the SDLC, even though they go hand in hand with CD. This creates a problem for customers, who inherently view feature flags as just another stage of that life cycle (many companies even repurpose their existing software release processes to mimic the functionality of feature flags). By representing feature flags as a pipeline, feature flag management becomes a natural step in the everyday workflow of development teams. It's integrated into CI/CD as a unified pipeline.

IN THIS CHAPTER

- Speeding up the process
- Securing the process
- Improving the developer experience
- Keeping costs in check

Chapter 2
Optimizing Continuous Integration

Continuous integration (CI) is a fundamental practice in modern software development that aims to streamline integrating code changes from multiple developers into a shared repository. It involves automating the build, test, and integration of code changes regularly, ensuring that conflicts are detected early in the process and the code base remains stable.

This chapter discusses the advantages of the CI model, including reducing the time required to debug and troubleshoot, minimizing risk, implementing secure practices, facilitating agile development and collaboration, increasing efficiency, and reducing costs.

Enhancing Speed

Parallel execution of builds and tests significantly speeds up the CI process. Instead of running tasks sequentially, multiple tests and build jobs can be executed simultaneously, reducing the time it takes to validate changes.

Intelligent caching strategies reduce redundant work. By caching dependencies, intermediate build outputs, and even docker

layers, the system avoids repeatedly rebuilding unchanged code or downloading dependencies, speeding up the build process.

Speed is also increased by using machine learning (ML) to automatically select the most relevant tests for a particular code change. This reduces the overall test time by focusing on only the necessary tests without compromising quality.

Artificial intelligence (AI) capabilities quickly identify pipeline issues and automatically suggest remediation so that failures do not cause long development delays. Another way to speed up builds is by selecting infrastructure optimized for builds and not generic infrastructure — for example, if a company is building iOS applications, then using macOS infrastructure instead of running Mac emulators on a Windows or Linux OS infrastructure.

Ensuring Security

Integrating security checks directly into the CI pipeline allows teams to perform security scans, such as scanning static code, vulnerability assessments, and compliance checks, such as ensuring artifacts have not been tampered with during the CI process, as part of the build process. Security is addressed early in the software development life cycle (SDLC), catching issues long before they reach production.

With role-based access control (RBAC), access to specific actions within the CI pipeline is based on a user's role. This minimizes the risk of unauthorized access to sensitive environments or actions, enhancing overall security.


TIP

RBAC is a security model that regulates access to system resources by assigning permissions to specific roles rather than individual users. Roles are created based on an organization's tasks or job functions, and permissions are associated with them. Users are then assigned one or more roles, and they inherit the permissions needed to perform their specific duties through these roles.

You can enhance security with secret management tools, such as Vault by Hashicorp or Amazon Web Services (AWS) Secrets Manager, to securely manage and inject credentials, keys, and other sensitive data into pipelines. This guarantees that secrets are never exposed in plain text.

The Harness platform allows for enforcing security and compliance policies through Open Policy Agent (OPA) integration, ensuring that only compliant builds are promoted to the next pipeline stage.

Improving the Developer Experience

Careful, developer-centric design features improve a developer's experience, such as a user-friendly interface that abstracts away much of the complexity of pipeline management. Developers can focus on coding, while automation handles the details of testing, building, and deployment.

TIP

Developers can create and manage their own pipelines using preconfigured templates. This reduces dependency on DevOps teams to set up and maintain CI pipelines, speeding up development cycles and allowing for faster iterations.

Providing real-time feedback to developers through integrations with popular collaboration tools like Slack and Microsoft Teams, developers are notified immediately if a build or test fails, allowing them to address issues quickly and continue coding without unnecessary delays.

Selecting tools that use AI to debug failed pipelines or security issues by analyzing logs and test results, suggesting root causes, and providing recommendations for fixes helps developers identify and resolve issues faster without needing deep expertise in CI/CD.

Simplified pipeline management, fast feedback loops, and AI-powered debugging work together to improve the developer experience and optimize CI.

Managing Infrastructure Costs

Almost as much as security, managing costs is everyone's concern. Although cloud infrastructure proves to be less expensive in most use cases, those costs can quickly skyrocket if they aren't thoughtfully managed. Managing sprawl is one of the most time-consuming aspects of working in and with a cloud environment.

An extremely valuable business ally is a tool suite that helps you manage infrastructure.

REMEMBER

In today's competitive business environment, managing costs can mean preventing financial losses later on.

DevOps teams need a solution that integrates with cloud platforms to dynamically scale build resources up or down based on current needs to ensure that teams aren't paying for idle resources, optimizing cloud spend. They need a tool that provides visibility into the cost and usage of CI/CD resources. Teams can track how much is spent on building infrastructure and set policies to control or limit resource usage, avoiding unexpected cost overruns.

By leveraging cloud-based build agents that run only when needed (that is, ephemeral build environments), you avoid the inefficiencies of maintaining static, always-on infrastructure. This results in lower costs compared to traditional on-premises CI solutions. By using the techniques mentioned earlier to speed up builds, you can reduce the compute time and storage required for builds, lowering infrastructure costs. This is particularly effective for large, complex projects with frequent build and test cycles.

Managing infrastructure costs through dynamic scaling, cost governance, and efficient resource utilization allows teams to deliver faster without overspending.

IN THIS CHAPTER

- Recognizing that scripting is not required
- Focusing on automation
- Delivering continuously
- Guaranteeing safety
- Reining in infrastructure costs

Chapter 3
Streamlining Continuous Delivery

Continuous delivery (CD) is a software development approach that automates the entire software release process. It aims to enable frequent and reliable software releases by ensuring that code changes can be deployed to production quickly and with minimal manual intervention. CD builds upon the foundation of continuous integration (CI) and extends it further by automating the deployment and release stages.

This chapter covers delivery and deployment without complex scripting, deployment automation, CD, safety, and managing infrastructure costs.

Explaining Continuous Delivery and Continuous Deployment

CD and continuous deployment are two closely related concepts in software development that aim to streamline the release process and deliver software more efficiently. While they share similarities, there are significant differences between the two approaches.

TECHNICAL STUFF

You'll often see the *CD* acronym used to refer to *both continuous delivery* and *continuous deployment.* In this book, for the sake of clarity, we use it only to refer to continuous delivery.

Continuous deployment is an extension of CD that takes automation to the next level. In continuous deployment, every code change that passes the automated tests is automatically deployed to production without human intervention. This means that new features, bug fixes, and improvements are released to users as soon as they're ready.

Alternatively, CD focuses on ensuring that software is always reliable but leaves the decision of when to deploy to production in the hands of the development team. With CD, the software is built, tested, and packaged in an automated manner, and it can be deployed to production at any time with minimal effort.

The main difference between CD and continuous deployment lies in the level of automation and control over the release process:

- Continuous deployment eliminates manual approval or intervention, allowing faster and more frequent releases. It requires a high level of confidence in the automated testing and deployment processes to ensure that only stable and reliable code reaches production.
- CD allows the development team to choose when to deploy the software to production. It also may include additional manual steps, such as final user acceptance testing or regulatory compliance checks, before releasing the software. This gives the development team more control over the release process and allows for a more cautious approach.

REMEMBER

Both CD and continuous deployment rely on similar practices and tools, such as continuous integration, automated testing, infrastructure such as code, and deployment automation. They aim to reduce the time and effort required to deliver software while maintaining high quality and reliability.

The choice between CD and continuous deployment depends on various factors, including your organization's risk tolerance, the complexity of the software, and the regulatory requirements. Continuous deployment is well suited for organizations prioritizing speed and agility, while CD balances automation and control.

Avoiding Script Writing

A common way to set up CD systems is to use the tool as an orchestrator that runs scripts that your team authors. However, scripting can be labor intensive and require ongoing maintenance for each pipeline the script has been copied to. To reduce your toil, minimize the amount of scripting necessary.

You may prefer tools with prebuilt integrations with your deployment targets, like Kubernetes, and other DevOps tools used by your team. Instead of writing custom scripts to integrate these tools, users can configure them through the interface. This plug-and-play setup eliminates the need for custom code.

A next-generation CI/CD suite provides reusable and preconfigured workflows for common tasks, such as deploying applications, running tests, or rolling back failed deployments. These workflows come with built-in logic and automation for handling complex scenarios (for example, canary and blue-green deployments) without requiring teams to script them manually.

Leveraging machine learning (ML) to handle tasks like identifying performance regressions, automating rollbacks, and selecting optimal tests for specific changes gives you intelligent automation that reduces the need to script custom validation or monitor logic within deployment pipelines.


TIP

Instead of manually scripting governance and compliance checks, built-in policies offer security, compliance, and operational standards. These policies can be enforced automatically across all pipelines without additional scripting.

Relying on these declarative and automated features significantly reduces the need for custom scripts in the software delivery and deployment processes, leading to faster, more reliable, and easier-to-manage pipelines.

Automating Deployment

Automating deployments in a CD solution focuses on streamlining the release process. Automation minimizes manual interventions, reduces errors, and ensures consistent, reliable software delivery.

TECHNICAL STUFF

Reliable software delivery is achieved through advanced automation, intelligent orchestration, and policy-driven governance, all of which contribute to a more efficient and robust CI/CD pipeline.

Here are the top five automated deployment features, how they work, and how they streamline CD:

- **Automating deployment workflows:** Users may define reusable deployment workflows that can be triggered automatically when specific conditions are met.
- **Continuous verification (CV):** Integrating AI-powered CV into its automated deployment process automatically monitors key performance indicators (KPIs), logs, and metrics during and after a deployment to identify potential issues, anomalies, or regressions.
- **Automated rollbacks:** Built-in strategies can be triggered based on predefined failure conditions or during the CV process.
- **Infrastructure as code (IaC) integration:** By integrating IaC into deployment workflows, infrastructure changes, such as scaling or configuration updates, happen in tandem with application deployments, providing a seamless and consistent delivery pipeline.
- **Declarative pipelines and templates:** Templates enable teams to create standardized deployment processes and enforce best practices so that every deployment follows the same rigorous standards, regardless of who executes the pipeline, which reduces human error and increases ease of management.

Implementing Progressive Delivery

Progressive delivery is incremental delivery that offers granular control over the delivery process. Engineers can set up pipelines to release new versions of their software initially to few users or servers, and then progress to more.

Progressive delivery has multiple benefits, including risk reduction, an improved user experience, faster time to market, and automated safeguards.

By gradually rolling out changes, developers can catch and resolve issues early — before they impact a larger user base. These controlled rollouts provide users a better experience using feature flags because new features are more stable and well-tested. Continuous delivery removes the need for major rollbacks and decreases the occurrence of failures, which accelerates the release cycle.

A progressive delivery model that uses AI-driven CD and automated rollbacks verifies that deployments are safe, reducing the burden on developers and operations teams. This model enables teams to ship software faster, with greater confidence and lower risk. It provides a comprehensive, automated framework for deploying updates incrementally and safely through canary releases, feature flags, and automated rollbacks.


TIP

Intelligent monitoring, CV, and granular control make progressive delivery a powerful tool for delivering smooth, low-risk software releases.

Ensuring Safety

A streamlined CD solution ensures safety by combining advanced automation, intelligent monitoring, built-in security features, and policy-based governance. It minimizes the risk of failed deployments, security breaches, and noncompliance while consistently and reliably delivering applications.

AI-driven verification, automated rollbacks, and built-in security scanning prevent faulty or insecure code from reaching production. Role-based access control (RBAC), policy enforcement, and templated pipelines standardize and secure deployment.

Managing Infrastructure Costs

As I explain in Chapter 2, managing infrastructure costs is an important consideration for DevOps personnel. A next-generation CI/CD suite provides visibility into the cost of deployments, particularly for cloud-based resources.

By analyzing usage patterns and deployment frequency, teams can optimize resource allocation and avoid over-provisioning and sprawl, helping manage infrastructure costs while maintaining the speed and efficiency of automated deployments.

A common challenge is the cost of test environments. When infrastructure provisioning is tightly integrated with CD, teams can use short-lived, "ephemeral" test environments. When a build is ready to be tested in a running environment, IaC is used to create new infrastructure, the software is deployed to it, tests are run, and then the infrastructure is deprovisioned. Although this adds a few minutes to the test cycle for provisioning, the organization doesn't pay for environments that are not actively in use.

IN THIS CHAPTER

- Guaranteeing consistency with templates
- Committing code changes
- Automating tests
- Checking compliance
- Focusing on pipeline security

Chapter 4
Standardizing and Governing

Standardizing and governing continuous integration/continuous delivery (CI/CD) pipelines helps enforce consistency, reliability, and security across software delivery processes. By implementing standardized pipelines, organizations can ensure that best practices, tools, and processes are uniformly applied across all projects and teams. Standardization reduces the risk of errors, accelerates onboarding, and enhances collaboration by eliminating ad-hoc workflows. It also simplifies maintenance and troubleshooting because engineers can rely on a consistent framework when addressing issues or updating code.

Governance promotes CI/CD pipelines' adherence to security, compliance, and operational policies, minimizing risks in production environments. Through governance, organizations can manually enforce policies such as role-based access control (RBAC), security scans, and compliance audits, reducing human error and enforcing standards, but the process should be automated. Standardization and governance help create scalable, reliable, and secure CI/CD practices that support faster, more controlled software releases.

This chapter covers using templates for standardization, governance, transitions from development to quality assurance (QA) and from QA to production, and addressing security through RBAC.

Using Templates and Open Policy Agent

Templates allow teams to define common pipeline elements, such as steps for testing, build configurations, security checks, and deployment strategies, in a centralized and reusable format. This ensures that all teams and projects follow the same structure and adhere to organizational standards.

Using predefined templates, organizations can enforce specific rules and policies, ensuring that specific security and compliance checks (for example, vulnerability scanning and code quality assessments) are automatically included in every pipeline. This prevents teams from bypassing necessary steps, promoting consistent governance across the organization.


TECHNICAL STUFF

Templates enable organizations to standardize and reuse components across pipelines, but Open Policy Agent (OPA) provides a flexible, declarative way to enforce governance and security policies through policy-as-code.

OPA policies can be embedded directly into templates or workflows to enforce compliance automatically. For example, a deployment template could include an OPA policy that checks whether security scans have passed before allowing deployment to production. This guarantees that every pipeline using the template adheres to governance standards without requiring manual intervention.

Defining the Transition from Development to QA

Six key components are required to define the transition from development to QA in the CI/CD pipeline.

Code commit and integration

The CI pipeline is triggered when developers commit code changes. This includes automated processes such as building the

application, running unit tests, and packaging the code for further testing. Before the transition, the code must pass through peer reviews or automated code quality checks (for example, static code analysis). Only approved code should proceed to the QA stage.

Automated testing

These tests check that individual components and integrations between components work as expected. Passing these tests is a prerequisite before the application code transitions to the QA environment. A strategy of continuous testing with automated test suites should be in place to ensure that the code is stable and functional before the QA stage.

Artifact management

The build artifacts generated from the development stage must be versioned and stored in a central repository. This proves that the same version is tested, deployed, and released in subsequent pipeline stages.

REMEMBER

Artifacts should also be scanned for vulnerabilities.

Environment setup and configuration

A stable, preconfigured QA environment — ideally, one that mirrors production — is required. This environment should be automatically provisioned or verified as part of the transition, and all necessary environment variables, database connections, and secrets should be correctly configured for the QA environment.

Gate checks and approvals

Before code moves from Dev to QA, automated policy checks, such as security scans and compliance audits, may be required to ensure the code adheres to organizational standards. Depending on the organization's governance model, a manual approval step may be required before the code transitions to QA, or the transition may occur automatically based on preset criteria.

Notification and collaboration

Developers and QA engineers should receive automated notifications about the transition so they can prepare for further testing. These notifications should include relevant information about the build, environment, and tests to be run.

TIP

Tools like Microsoft Teams and Slack can be integrated into the pipeline to notify relevant teams when the transition occurs, providing transparency and collaboration.

This structured transition provides a seamless, controlled, and automated handoff from Dev to QA, focusing on code quality, test automation, and proper environment management.

Defining the Transition from Quality Assurance to Production

Eight essential steps and requirements must be considered to define the transition from QA to production in a CI/CD pipeline, enabling a smooth and secure deployment process.

Final testing and validation

Before moving to production, the software must pass final rounds of testing, such as user acceptance testing (UAT), where users or business stakeholders validate that the software meets requirements and performs as expected. Before transitioning to production, verify that recent changes haven't broken existing functionality.

TIP

To validate the stability of the code base, execute automated regression test suites. Teams should check that applications can handle expected production loads through stress or load testing.

Artifact preparation

The code or build artifact that passed QA must be properly versioned and tagged for production deployment. This guarantees that what is tested in QA is the same as what is deployed to production, that the artifacts are *immutable* (not modifiable after creation), and that they're stored correctly in an artifact repository for traceability and rollbacks if needed.

TECHNICAL STUFF

You can also secure your supply chain against unauthorized changes by achieving SLSA L3 compliance. Learn more at `https://www.harness.io/blog/an-in-depth-look-at-achieving-slsa-level-3-compliance-with-harness`

Approval and governance checks

Transitions to production often require approvals from stakeholders, product owners, or DevOps teams. These approvals can be manual through a sign-off process or automated if predefined criteria such as passing tests or policy checks are met. Final security checks, such as vulnerability scanning or penetration tests, should be reviewed and checks run before approval is requested, so that compliance audits may be enforced to check that the application meets security and legal standards before moving to production.

Production environment setup

Be sure that the production environment is ready for deployment, including proper configuration of servers, networking, databases, and security settings. Ideally, this is automated through infrastructure-as-code tools. All production-specific environment variables and secrets should be configured and checked to test secure and correct application behavior.

Deployment strategy

The deployment strategy should be defined based on the project needs. This could involve canary releases, blue-green deployments, rolling updates, or feature flag toggling to minimize downtime and mitigate risk. A clearly defined and automated rollback strategy should be in place. If the deployment fails or causes issues in production, the system can be quickly reverted to a stable state.

Monitoring and alerts

Before transitioning to production, check that monitoring and logging systems are in place to track the application's health post-deployment.

TIP

Datadog, ELK Stack, or Prometheus can be configured to detect anomalies or errors in real-time.

Set up alerts to notify teams about the deployment status and any issues that arise during or after the transition. This assures rapid response to production incidents.

Post-deployment verification

After they're deployed to production, automated or manual tests should be run to validate the application's key functionality and validate that it functions as expected in the live environment. Post-deployment monitoring should include tracking error rates, performance metrics, and user behavior to quickly detect signs of trouble in production.

Documentation and communication

Document released changes, including bug fixes, new features, and any known issues. If applicable, communicate these notes to relevant teams and users. Notify all stakeholders, including developers, QA, operations, and business teams, about the deployment status and any necessary follow-up actions.

The transition from QA to production requires careful validation, governance approvals, a robust deployment strategy, and post-deployment monitoring to ensure the application is deployed safely and successfully.

Securing Your Pipelines with Role-Based Access Control

Templates can use an RBAC system to govern who can create, modify, or use specific templates. By implementing RBAC in CI/CD pipelines, organizations can limit access to critical components based on user roles, improving security and reducing the risk of unauthorized changes or deployments.

IN THIS CHAPTER

- Preventing pipeline problems
- Determining which tests to run
- Rolling out parallel releases
- Automating with GitOps

Chapter 5
Ten Features Shaping the Future of CI/CD

As this book explains, next-generation, artificial intelligence (AI)–driven continuous integration/continuous delivery (CI/CD) has many advantages over traditional tools. This chapter outlines ten features of Harness's solution that will shape the future of CI/CD.

Using AI for Debugging Pipeline Issues

Harness's use of AI for debugging aims to reduce the complexities of managing and troubleshooting CI/CD pipelines, improving developer productivity and software delivery reliability. Additionally, AI-driven insights can recommend fixes or configurations to prevent future pipeline problems, allowing continuous improvement in CI/CD processes.

Using AI for Selecting the Right Tests to Run

Harness uses AI to optimize test selection by analyzing previously collected test data, code changes, and their impact on various application parts. The AI engine identifies specific tests relevant to the modified code, significantly decreasing the time and effort spent running unnecessary tests. Using machine learning (ML) algorithms, the platform learns from historical patterns and test outcomes, refining its understanding of which tests are critical to run for a particular change or deployment.

Automating Rollbacks

Deployment verification will trigger a defined strategy to respond to failure, such as removing a canary node from a load balancer. If a regression is found, you can roll back a deployment automatically or manually, with human intervention and smart notifications.

Implementing Progressive Delivery

Progressive delivery is a practice that builds on the capabilities of CI and CD to help you deliver with control. You can run new versions in parallel with production releases, allowing you to test before a complete rollout and avoiding the all-or-nothing approach to delivery.

Enabling Automatic Release Verification

Harness continuous verification (CV) integrates with application performance monitoring and logging tools to verify that the deployment is running safely and efficiently. It applies ML algorithms to every deployment to identify normal behavior, allowing Harness to identify and flag anomalies in future deployments.

Integrating with GitOps Workflows

You can set up GitOps agents to manage the state of your deployments and combine that with a centralized user interface (UI) and full-featured deployment pipeline. The result is a truly scalable, fully automated path to delivery that includes continuous operations via GitOps.

Leveraging Machine Learning for Anomaly Detection

Harness uses unsupervised ML to automate the analysis of all the time-series metrics and event data from application key performance indicators (KPIs), data, and metrics. This analysis allows Harness to automatically verify production deployments and identify any regressions, anomalies, or failures that may have been introduced.

Providing Real-Time Feedback Loops

Harness's rapid feedback loop encourages iterative development and empowers developers to improve based on real-time insights. It also facilitates faster bug detection and resolution, resulting in more efficient development cycles.

Streamlining Security Compliance

Harness's CI/CD pipelines help organizations with their DevSecOps to integrate security in the software development life cycle (SDLC) continuously to streamline security compliance requirements.

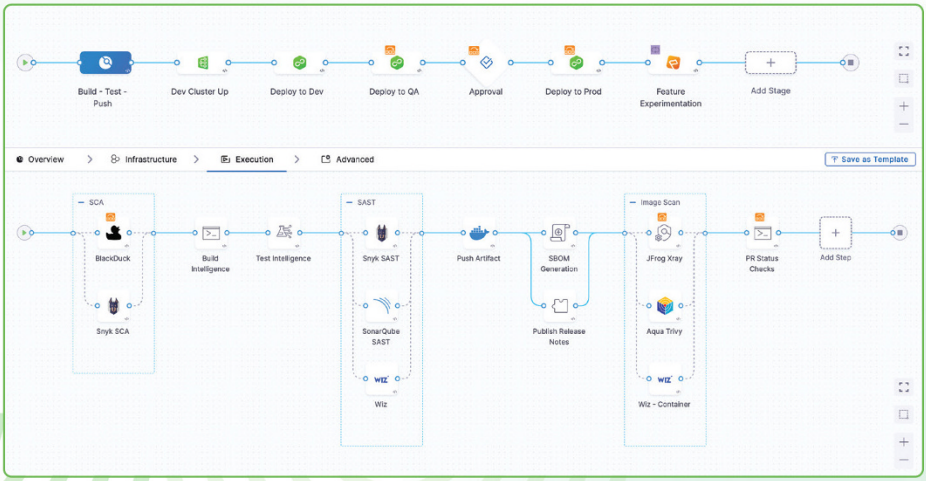
Facilitating Self-Service Deployments

Self-service CI/CD provides visibility, features, and access control to allow individuals to deliver their work without depending on someone else.

harness

Ship 8x Faster

Securely Accelerate Your DevOps: Build Fast, Test Effectively, and Deploy with Confidence.



Start Your Free Trial

See how the industry's first Software Delivery Platform to use AI can simplify your DevOps processes - including CI, CD, Feature Flags, Cloud Costs, and much more.

Experience AI-native software delivery

Discover how automated code testing, deployment, and rollbacks for your on-premises and cloud projects can help you deliver software faster. You'll learn how to streamline your software delivery with automated continuous integration (CI), continuous delivery (CD), feature flags, and embedded security and governance without requiring scripting or custom coding.

Inside...

- Continuous delivery
- Continuous integration
- Chaos engineering
- Infrastructure as code management
- Feature flags
- End-to-end security

Kenneth Hess has more than 20 years of operations and development experience in managing software, staging, deployments, and ongoing support for all stages of the SDLC.

ISBN: 978-1-394-31327-3
Not For Resale

for dummies®
A Wiley Brand

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.