Harness Platform

Comprehensive Overview of Modern CI/CD and Platform Engineering

Prepared By: Rashi Rana
Corporate Trainer

Introduction to Harness

> **Harness** is a comprehensive Software Delivery Platform that provides end-to-end automation for building, testing, deploying, and managing applications across the entire software development lifecycle.

Key Characteristics

- **Cloud-Native Platform:** Built for modern, distributed applications and infrastructure
- **AI-Powered:** Leverages machine learning for intelligent automation and decision-making
- **Enterprise-Grade:** Designed for large-scale organizations with complex deployment requirements
- **Developer-Centric:** Focuses on improving developer experience and productivity

Platform Philosophy

Harness aims to eliminate the complexity and manual effort traditionally associated with software delivery by providing:

- Automated pipelines with intelligent rollback capabilities
- Real-time monitoring and verification
- Risk-based deployment strategies

- Comprehensive observability and governance

Prepared By: Rashi Rana
Corporate Trainer

What is Harness?

Harness is a unified platform that transforms how organizations approach software delivery, providing a complete suite of tools for modern DevOps practices.

Core Value Proposition

Automation First

Eliminates manual processes through intelligent automation across the entire delivery pipeline.

Risk Reduction

Advanced verification and rollback mechanisms ensure safe and reliable deployments.

Scalability

Handles deployments from small applications to enterprise-scale systems with thousands of services.

Observability

Comprehensive insights into deployment performance, system health, and business metrics.

Target Use Cases

- Organizations transitioning to cloud-native architectures

- Teams requiring advanced deployment strategies (Blue-Green, Canary, Rolling)
- Companies seeking to reduce deployment failures and mean time to recovery
- Enterprises needing governance, compliance, and audit capabilities

Prepared By: Rashi Rana
Corporate Trainer

Harness Role in Continuous Delivery

Continuous Delivery Transformation

Harness revolutionizes traditional CD practices by introducing intelligence and automation at every stage of the delivery process.

> **Traditional CD Challenges Addressed:**
>
> - Manual deployment processes prone to human error
> - Lack of automated verification and testing
> - Inconsistent deployment practices across teams
> - Limited visibility into deployment health and performance
> - Slow rollback and recovery procedures

Harness CD Capabilities

- **Intelligent Pipelines:** AI-powered pipeline creation and optimization
- **Automated Verification:** Continuous monitoring during deployments with automatic rollback
- **Progressive Delivery:** Advanced deployment patterns with traffic splitting and gradual rollouts
- **Infrastructure Abstraction:** Deploy to any cloud, Kubernetes, or traditional infrastructure
- **Governance and Compliance:** Built-in approval workflows, audit trails, and policy enforcement

CD Pipeline Enhancement

Harness enhances traditional CI/CD pipelines by providing:

- Real-time health verification during deployments
- Automatic anomaly detection and response
- Integration with monitoring and observability tools
- Self-healing deployment capabilities

Prepared By: Rashi Rana
Corporate Trainer

Harness vs. Jenkins Comparison

Aspect	Harness	Jenkins
Platform Type	Cloud-native SaaS/On-premise	Self-hosted, server-based
Setup & Maintenance	Minimal setup, managed infrastructure	Requires installation, configuration, and ongoing maintenance
Pipeline Creation	Visual pipeline builder with AI assistance	Code-based (Jenkinsfile) or UI configuration
Deployment Verification	Built-in automated verification and rollback	Requires custom scripting and external tools
Scalability	Auto-scaling with managed infrastructure	Manual scaling with master-slave architecture
Security	Enterprise-grade security, built-in secrets management	Requires additional configuration and plugins
Monitoring	Integrated monitoring and analytics	Requires external monitoring solutions
Cost Model	Subscription-based pricing	Open source (infrastructure and maintenance costs)

When to Choose Harness

- Organizations prioritizing speed and reduced operational overhead
- Teams requiring advanced deployment strategies and automated verification
- Companies needing enterprise-grade governance and compliance features
- Organizations with limited DevOps engineering resources for tool maintenance

Prepared By: Rashi Rana
Corporate Trainer

Harness Modules Overview

Harness provides a comprehensive suite of modules addressing different aspects of the software delivery lifecycle.

Continuous Integration

Build and test automation with intelligent caching, parallel execution, and comprehensive reporting.

Continuous Delivery

Advanced deployment orchestration with automated verification, rollback, and progressive delivery patterns.

Feature Flags

Runtime feature control enabling safe feature releases and A/B testing capabilities.

Chaos Engineering

Proactive resilience testing through controlled failure injection and system validation.

Cloud Cost Management

Comprehensive cost visibility, optimization recommendations, and automated cost governance.

Security Testing Orchestration

Automated security scanning integration throughout the development and deployment pipeline.

Service Reliability Management

SLO management, error budgets, and automated incident response capabilities.

Internal Developer Portal

Centralized developer experience with service catalogs, documentation, and self-service capabilities.

Prepared By: Rashi Rana
Corporate Trainer

Continuous Delivery Module Deep Dive

Core Capabilities

> The Continuous Delivery module is the flagship offering of Harness, providing sophisticated deployment orchestration with built-in intelligence and safety mechanisms.

Key Features

- **Pipeline Orchestration:** Visual pipeline designer with complex workflow support
- **Deployment Strategies:** Blue-Green, Canary, Rolling, and custom deployment patterns
- **Automated Verification:** Real-time health monitoring during deployments
- **Rollback Automation:** Intelligent rollback triggers based on performance metrics
- **Infrastructure Provisioning:** Terraform, CloudFormation, and Kubernetes integration
- **GitOps Support:** Git-based configuration management and synchronization

Deployment Verification Process

1. **Pre-deployment:** Infrastructure and dependency validation
2. **Deployment:** Progressive rollout with traffic management

3. **Verification:** Automated health checks, performance monitoring, and business metric validation
4. **Decision:** AI-powered analysis determines deployment success or triggers rollback
5. **Post-deployment:** Continuous monitoring and optimization recommendations

Integration Ecosystem

Seamless integration with monitoring tools (Datadog, New Relic, AppDynamics), cloud platforms (AWS, Azure, GCP), and container orchestration systems (Kubernetes, ECS, AKS).

Prepared By: Rashi Rana
Corporate Trainer

Feature Flags Module

Feature Flag Fundamentals

Feature flags enable teams to decouple code deployment from feature releases, providing runtime control over application behavior without additional deployments.

Core Capabilities

- **Flag Management:** Centralized flag configuration with real-time updates
- **Targeting:** Advanced user segmentation and percentage-based rollouts
- **A/B Testing:** Experimentation framework with statistical analysis
- **Safety Controls:** Flag dependencies, prerequisites, and automated kill switches
- **Observability:** Flag usage analytics and performance impact measurement

Use Cases

Progressive Rollouts

Gradual feature introduction to minimize risk and gather user feedback.

Emergency Controls

Instant feature disabling capability for rapid incident response.

User Experience Testing

A/B testing and multivariate experimentation for optimization.

Operational Toggles

Runtime configuration changes for performance optimization and system tuning.

Integration Benefits

Seamlessly integrates with the Continuous Delivery module to enable feature-flag-driven deployments, reducing deployment risk while maintaining development velocity.

Prepared By: Rashi Rana
Corporate Trainer

Chaos Engineering Module

Chaos Engineering Principles

> Chaos Engineering is the practice of intentionally injecting failures into systems to identify weaknesses and improve overall resilience before issues occur in production.

Harness Chaos Engineering Features

- **Experiment Library:** Pre-built chaos experiments for common failure scenarios
- **Hypothesis Testing:** Define success criteria and automatically validate system behavior
- **Progressive Chaos:** Gradual increase in experiment complexity and scope
- **Blast Radius Control:** Precise targeting and containment of chaos experiments
- **Observability Integration:** Real-time monitoring during experiments with automatic abort mechanisms
- **Compliance and Governance:** Audit trails, approval workflows, and scheduling controls

Experiment Types

Infrastructure Chaos

Application Chaos

CPU stress, memory exhaustion, disk I/O saturation, and network partitioning.

Service failures, latency injection, and dependency unavailability scenarios.

Platform Chaos

Kubernetes pod failures, container crashes, and node unavailability testing.

Cloud Provider Chaos

Regional outages, service degradation, and multi-zone failure simulation.

Business Value

Proactive identification of system weaknesses, improved incident response capabilities, increased confidence in system resilience, and reduced mean time to recovery (MTTR).

Prepared By: Rashi Rana
Corporate Trainer

Laboratory Exercise: Environment Setup

Harness Platform Access Options

Option 1: Harness SaaS Platform

1. Navigate to `https://app.harness.io` in your web browser
2. Click "Start Free" to create a new account or "Sign In" if you have existing credentials
3. Complete the registration process with your professional email address
4. Verify your email address through the confirmation link
5. Complete the initial setup wizard including organization and project configuration

Option 2: Harness Community Edition (Self-Hosted)

```
# Prerequisites: Docker and Docker Compose installed
# Download Harness Community Edition
wget https://github.com/harness/harness-cd-community/releases/latest/download/harness-cd-community.tar.gz

# Extract and setup
tar -xzf harness-cd-community.tar.gz
cd harness-cd-community
```

```
# Start Harness services
docker-compose up -d
```

System Requirements

- **Browser:** Chrome 90+, Firefox 88+, Safari 14+, Edge 90+
- **Network:** Stable internet connection for SaaS platform
- **Local Setup (Community Edition):** 8GB RAM minimum, 16GB recommended
- **Storage:** 50GB available disk space for local installations

Prepared By: Rashi Rana
Corporate Trainer

Hands-on: Navigate Harness UI

UI Navigation Exercise

1. **Dashboard Overview:** Familiarize yourself with the main dashboard, noting key metrics, recent activities, and quick access panels
2. **Navigation Menu:** Explore the left navigation menu structure:
    - 3 Projects and Organizations
    - 4 Pipelines and Services
    - 5 Infrastructure and Environments
    - 6 Feature Flags and Chaos Engineering
    - 7 Settings and Administration
8. **Account Settings:** Access account settings to understand user management, authentication options, and billing information
9. **Help and Documentation:** Locate and explore the built-in help system, documentation links, and support resources
10. **Search Functionality:** Test the global search feature to find projects, pipelines, and services efficiently
11. **Notification Center:** Check the notification panel for system updates, deployment status, and important alerts

Key UI Components

Command Palette

Quick access to all platform functions using Ctrl+K (Cmd+K on Mac)

Breadcrumb Navigation

Hierarchical navigation showing current location within the platform

Context Switching

Easy switching between different accounts, organizations, and projects

Prepared By: Rashi Rana
Corporate Trainer

Hands-on: Create Basic Project

Project Creation Workflow

1. **Access Project Creation:** Click on "Projects" in the main navigation, then select "New Project" button
2. **Project Configuration:**
    - 3. Name: "Development-Workflow-Demo"
    - 4. Identifier: Auto-generated or custom
    - 5. Description: "Demonstration project for learning Harness platform capabilities"
    - 6. Color/Tags: Optional customization for organization
7. **Module Selection:** Choose the modules you want to enable for this project:
    - 8. Continuous Integration
    - 9. Continuous Delivery
    - 10. Feature Flags
    - 11. Chaos Engineering (if available)
12. **Organization Assignment:** Assign the project to the appropriate organization (default or custom)
13. **Collaboration Setup:** Configure initial team members and their roles:

- 14 Project Admin: Full project management capabilities
- 15 Pipeline Operator: Execute and manage pipelines
- 16 Viewer: Read-only access to project resources

17 **Project Validation:** Review project settings and confirm creation

Post-Creation Verification

- Verify project appears in the projects list
- Access the project dashboard and explore available options
- Check that selected modules are properly enabled
- Test navigation between project sections

Prepared By: Rashi Rana
Corporate Trainer

Sample Pipeline: Spring Boot Hello World

Pipeline Overview

We'll create a complete CI/CD pipeline for a Spring Boot "Hello World" application demonstrating the full Harness workflow from code to deployment.

Application Structure

```
spring-boot-hello/
├── src/main/java/com/example/demo/
│   └── HelloWorldController.java
├── src/main/resources/
│   └── application.properties
├── Dockerfile
├── k8s-manifest.yaml
└── pom.xml
```

Sample Spring Boot Controller

```
@RestController
public class HelloWorldController {

    @GetMapping("/hello")
    public ResponseEntity<String> hello() {
        return ResponseEntity.ok("Hello World
from Harness!");
```

```
    }

    @GetMapping("/health")
    public ResponseEntity<String> health() {
        return ResponseEntity.ok("Application
is healthy");
    }
}
```

Prepared By: Rashi Rana
Corporate Trainer

Pipeline Configuration Steps

Step 1: Create Pipeline Infrastructure

1. **Source Code Repository:** Connect to GitHub/GitLab repository containing the Spring Boot application
2. **Build Infrastructure:** Configure Harness CI with Java 17 and Maven build environment
3. **Container Registry:** Set up Docker Hub or AWS ECR for storing application images
4. **Target Environment:** Configure Kubernetes cluster (local minikube or cloud-based)

Step 2: CI Pipeline Configuration

```
# Build Stage Commands
mvn clean compile
mvn test
mvn package -DskipTests

# Docker Build
docker build -t spring-boot-
hello:${HARNESS_BUILD_ID} .
docker push your-registry/spring-boot-
hello:${HARNESS_BUILD_ID}
```

Step 3: Dockerfile Configuration

```
FROM openjdk:17-jre-slim
WORKDIR /app
COPY target/spring-boot-hello-1.0.jar app.jar
EXPOSE 8080
HEALTHCHECK --interval=30s --timeout=3s --
retries=3 \
  CMD curl -f http://localhost:8080/health ||
exit 1
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Prepared By: Rashi Rana
Corporate Trainer

Deployment Configuration

Kubernetes Manifest

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-boot-hello
spec:
  replicas: 3
  selector:
    matchLabels:
      app: spring-boot-hello
  template:
    metadata:
      labels:
        app: spring-boot-hello
    spec:
      containers:
      - name: app
        image: <+artifact.image>
        ports:
        - containerPort: 8080
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 10
```

CD Pipeline Stages

1. **Artifact Source:** Pull the Docker image built in CI stage
2. **Environment Setup:** Deploy to Development environment first
3. **Health Verification:** Automated health checks using /health endpoint
4. **Canary Deployment:** 25% traffic to new version, monitor for 5 minutes
5. **Production Rollout:** Full deployment after successful verification

Prepared By: Rashi Rana
Corporate Trainer

Pipeline Testing and Verification

Verification Configuration

Health Check Verification

```
# Harness Health Check Configuration
Verification Type: HTTP
URL: http://<service-url>/health
Method: GET
Expected Response Code: 200
Expected Response Body: "Application is
healthy"
Timeout: 30 seconds
Retry Attempts: 3
```

Performance Verification (Optional)

```
# Load Testing with Harness
curl -X GET "http://<service-url>/hello" \
  -H "accept: application/json" \
  --connect-timeout 5 \
  --max-time 10
```

Pipeline Execution Flow

1. **Trigger:** Git push to main branch triggers the pipeline automatically
2. **Build:** Maven build, test execution, and Docker image creation

3. **Security Scan:** Container vulnerability scanning (optional)
4. **Deploy to Dev:** Automatic deployment to development environment
5. **Verification:** Automated health checks and basic functionality tests
6. **Approval Gate:** Manual approval for production deployment
7. **Production Deploy:** Canary deployment with traffic splitting
8. **Monitor:** Real-time monitoring and automatic rollback if issues detected

Prepared By: Rashi Rana
Corporate Trainer

Hands-on: Create the Spring Boot Pipeline

Practical Exercise

1. **Repository Setup:**
   - 2. Fork the sample Spring Boot repository: `https://github.com/harness-community/spring-boot-hello-world`
   - 3. Clone to your local environment for code review
4. **Create CI Pipeline:**
   - 5. Navigate to Pipelines → Create New Pipeline
   - 6. Select "Build" pipeline type
   - 7. Configure Git connector to your forked repository
   - 8. Add Build stage with Maven and Docker steps
9. **Configure Build Steps:**
   - 10. Add "Run Tests" step with Maven test command
   - 11. Add "Build JAR" step with Maven package command

- 12 Add "Build and Push Docker Image" step
- 13 Configure Docker registry credentials

14 **Create CD Pipeline:**

- 15 Create new Deploy pipeline
- 16 Define Service with Docker artifact source
- 17 Configure target Kubernetes environment
- 18 Add deployment strategy (Rolling or Canary)

19 **Test Pipeline Execution:**

- 20 Execute the pipeline manually
- 21 Monitor build and deployment logs
- 22 Verify application accessibility via /hello endpoint
- 23 Test rollback functionality by introducing a failure

Prepared By: Rashi Rana
Corporate Trainer

Session Summary and Next Steps

Key Concepts Covered

- Harness as a comprehensive software delivery platform
- Role and advantages in modern continuous delivery practices
- Comparative analysis with traditional tools like Jenkins
- Overview of integrated modules for complete DevOps lifecycle
- Practical experience with platform navigation and project setup

Practical Skills Acquired

- Platform navigation and user interface familiarity
- Understanding of project structure and organization
- Module selection and configuration basics
- Access to development environment for continued learning

Recommended Next Steps

1. **Pipeline Creation:** Build your first CI/CD pipeline using the project created
2. **Service Configuration:** Define services and deployment environments
3. **Integration Setup:** Connect to version control systems and cloud providers

4. **Advanced Features:** Explore feature flags, chaos engineering, and advanced deployment strategies
5. **Best Practices:** Study enterprise patterns and governance implementation

Additional Resources

- Harness University: Comprehensive learning paths and certifications
- Documentation: Complete technical reference and tutorials
- Community Forums: Peer support and knowledge sharing
- API Reference: Integration and automation capabilities

Prepared By: Rashi Rana
Corporate Trainer