**How is the Release Orchestration process handled?**

**1. Understanding Release Orchestration in Harness**

Release orchestration in Harness refers to automating, coordinating, and managing the entire software release process across multiple environments. Harness ensures deployments are reliable, repeatable, and auditable.

**2. Key Components of Release Orchestration in Harness**

Harness consists of various components that streamline the release orchestration process:

**A. Pipelines**

Pipelines in Harness define the complete workflow for a release. A pipeline consists of multiple stages, each representing a step in the release process.

**B. Stages**

Each stage in the pipeline represents a part of the release process, such as:

* **Build & Artifact Management**: Integrates with CI tools (Jenkins, GitHub Actions, GitLab CI/CD, etc.).
* **Approval & Verification**: Manual approvals, automated verification with AI-based anomaly detection.
* **Deployment**: Deploys to Kubernetes, VMs, Serverless, etc.
* **Testing**: Runs automated tests (integration, smoke, performance tests).
* **Rollback**: Defines rollback strategies in case of failures.

**C. Service & Environment**

* **Service**: Represents an application or microservice being deployed.
* **Environment**: Specifies where the service will be deployed (e.g., Dev, QA, Staging, Production).

**D. Infrastructure & Connectors**

Harness integrates with cloud providers (AWS, Azure, GCP) to manage infrastructure and deployments.

**E. Templates & Reusable Components**

Harness supports YAML-based configuration and allows the creation of reusable templates for pipeline standardization.

**3. Release Orchestration Process in Harness**

**Step 1: Pipeline Definition**

* A pipeline is created using the Harness UI or YAML.
* Define stages like Deployment, Verification, and Rollback.

**Step 2: Artifact Management**

* Harness fetches artifacts from a repository (Docker Hub, ECR, Nexus, JFrog, etc.).
* Supports versioning and rollback strategies.

**Step 3: Deployment Execution**

* Harness deploys applications using canary, blue-green, rolling, or other deployment strategies.
* Uses Infrastructure as Code (IaC) to provision and configure environments.

**Step 4: Approvals & Governance**

* Incorporates manual or automated approvals.
* Role-based access control (RBAC) ensures security compliance.

**Step 5: Automated Verification**

* Uses AI-based monitoring to detect deployment anomalies.
* Integrates with observability tools (Datadog, Splunk, Prometheus).

**Step 6: Rollback & Recovery**

* Automatically rolls back in case of failures based on predefined rules.
* Can be triggered manually or automatically.

**Step 7: Post-Deployment Activities**

* Notifies stakeholders via Slack, email, or other collaboration tools.
* Generates reports and logs for auditing.

**4. Advanced Features in Harness Release Orchestration**

* **Feature Flags**: Gradual rollout of features with feature toggles.
* **Chaos Engineering**: Introduces controlled failures for resilience testing.
* **Multi-cloud Deployments**: Deploy across AWS, Azure, and GCP simultaneously.
* **Custom Scripts & Integrations**: Supports Python, Bash, Groovy scripts for custom automation.

**5. Example Harness YAML for a Simple Release Pipeline**

Here’s an example YAML pipeline in Harness:

pipeline:

name: Sample Release Pipeline

identifier: sample\_release\_pipeline

stages:

- stage:

name: Deploy to Dev

type: Deployment

service:

identifier: my-service

environment:

identifier: dev-env

execution:

steps:

- step:

name: Deploy Application

type: KubernetesRollingDeploy

spec:

manifest:

identifier: app-manifest

valuesPaths:

- values-dev.yaml

- step:

name: Verify Deployment

type: HarnessApproval

spec:

approvers:

userGroups:

- DevOps-Team

- stage:

name: Deploy to Prod

type: Deployment

service:

identifier: my-service

environment:

identifier: prod-env

execution:

steps:

- step:

name: Deploy Application

type: KubernetesRollingDeploy

spec:

manifest:

identifier: app-manifest

valuesPaths:

- values-prod.yaml

- step:

name: Post Deployment Notification

type: SlackNotification

spec:

channel: "#releases"

message: "Deployment to Prod completed successfully"

**6. Benefits of Using Harness for Release Orchestration**

* **Automation & Efficiency**: Reduces manual effort in deployments.
* **AI-driven Verification**: Ensures safe rollouts with anomaly detection.
* **Security & Compliance**: Provides RBAC and audit logs.
* **Scalability**: Works across multiple cloud and hybrid environments.
* **Cost Optimization**: Monitors and optimizes cloud costs.

**Conclusion**

Harness simplifies release orchestration by automating deployments, approvals, verification, and rollbacks. It ensures seamless software releases with governance, security, and observability.

Would you like a deeper dive into a specific feature, such as rollback strategies or approval workflows?

**Does it have any Configuration management capability?**

Yes, Harness has configuration management capabilities, but it primarily integrates with external Configuration Management (CM) tools rather than being a full-fledged CM tool itself. Here’s how Harness handles configuration management:

**1. Configuration Management in Harness**

Harness allows you to manage configurations through various mechanisms, including environment variables, secrets management, infrastructure as code (IaC) integrations, and templating.

**2. Key Features of Configuration Management in Harness**

**A. Service Configurations & Environment Variables**

* Harness allows you to define configuration variables for different environments.
* Supports parameterized values that can change between deployments.
* These values can be stored in:
  + YAML files
  + Environment variables
  + Config maps or secrets in Kubernetes

**B. Secrets Management**

* Harness integrates with external secret management tools:
  + **AWS Secrets Manager**
  + **HashiCorp Vault**
  + **Azure Key Vault**
  + **CyberArk**
  + **Google Secret Manager**
* Sensitive values such as API keys, database credentials, and tokens are securely managed.

**C. Infrastructure as Code (IaC) Integrations**

* Harness supports configuration management through **Terraform** and **CloudFormation**.
* You can provision infrastructure dynamically before deploying applications.
* Example: Run Terraform plans before deployment to ensure the right infrastructure is in place.

**D. Kubernetes Config Management**

* Supports Kubernetes **ConfigMaps** and **Secrets** to store application configuration.
* Configurations can be injected into deployments dynamically.
* Example:

apiVersion: v1

kind: ConfigMap

metadata:

name: app-config

data:

APP\_ENV: "production"

LOG\_LEVEL: "info"

**E. Helm & Kustomize for Configuration Management**

* Harness supports **Helm Charts** to manage application configurations.
* **Kustomize** can be used to override configurations per environment.
* Example Helm values override:

valuesPaths:

- values-dev.yaml

- values-prod.yaml

**F. GitOps for Configuration Management**

* Harness integrates with GitOps tools like **ArgoCD** and **Flux**.
* Application configuration can be stored in Git and automatically synchronized.

**G. Configurations as Code (YAML)**

* Harness uses YAML-based configuration files, allowing version control.
* Supports pipeline as code and infrastructure as code.

**3. Example Configuration Management in Harness**

**Example: Injecting Configurations Using Environment Variables**

pipeline:

name: Sample Release Pipeline

stages:

- stage:

name: Deploy to Dev

type: Deployment

service:

identifier: my-service

environment:

identifier: dev-env

execution:

steps:

- step:

name: Set Environment Variables

type: ShellScript

spec:

script: |

export APP\_ENV=development

export DATABASE\_URL=jdbc:mysql://dev-db:3306/mydb

**Example: Using a Secret in a Deployment**

service:

name: My Service

artifact:

source: docker.io/my-app:latest

configuration:

secrets:

- name: DB\_PASSWORD

secretManager: HashiCorpVault

**4. Does Harness Replace Configuration Management Tools?**

No, Harness does not replace traditional configuration management tools like **Ansible, Chef, or Puppet**. Instead, it integrates with these tools to provide a seamless deployment experience.

* **For full infrastructure provisioning**: Use Terraform, CloudFormation, or Pulumi.
* **For application configuration**: Use Helm, Kustomize, Kubernetes Secrets, or GitOps.
* **For secrets management**: Use AWS Secrets Manager, HashiCorp Vault, or CyberArk.

**5. When to Use Harness for Configuration Management?**

Use Harness for: ✅ Managing application configurations dynamically across environments.  
✅ Injecting environment variables and secrets securely.  
✅ Integrating with external CM tools (Terraform, Helm, GitOps).  
✅ Automating infrastructure provisioning before deployments.

**Conclusion**

Harness provides strong configuration management capabilities by integrating with existing tools and managing configurations dynamically. However, it is not a full replacement for dedicated configuration management tools like Ansible or Chef.

**How good is the deployment capabilities?**

Harness has **best-in-class deployment capabilities** with a focus on automation, flexibility, security, and governance. It simplifies complex deployments across multiple environments, ensuring reliability and efficiency. Below is a detailed breakdown of its deployment capabilities:

**1. Deployment Strategies Supported by Harness**

Harness supports a variety of deployment strategies, making it suitable for different use cases:

**A. Rolling Deployment**

* Gradually updates instances in batches while keeping the application running.
* Reduces downtime but may cause temporary performance issues.
* **Use case**: Stateless applications and microservices.

**B. Blue-Green Deployment**

* Deploys a new version (green) alongside the existing one (blue).
* Switches traffic to the new version after validation.
* **Use case**: Zero-downtime deployments with fast rollbacks.

**C. Canary Deployment**

* Rolls out changes to a small subset of users before full deployment.
* Traffic is gradually increased while monitoring for issues.
* **Use case**: Risk-sensitive applications (e.g., banking, healthcare).

**D. Feature Flags Deployment**

* Allows enabling/disabling features without redeploying code.
* Can be used for gradual rollouts and A/B testing.
* **Use case**: Controlled feature releases and experimentation.

**E. Shadow Deployment**

* Routes real traffic to a new version without affecting users.
* Useful for performance testing before a full rollout.
* **Use case**: AI/ML model validation, traffic mirroring.

**F. Custom Deployment Workflows**

* Harness allows custom deployment strategies using YAML.
* Users can define specific rollback steps, manual approvals, and automated testing.

**2. Deployment Targets and Integrations**

Harness supports deployment to various platforms:

**A. Kubernetes & Containers**

* Deploy to **Kubernetes** (EKS, AKS, GKE, OpenShift).
* Supports **Helm** and **Kustomize** for configuration management.
* **Integration**: Works with Istio, Linkerd for service mesh-based deployments.

**B. Virtual Machines (VMs) & Bare Metal**

* Supports AWS EC2, Azure VMs, Google Compute Engine.
* Uses SSH or WinRM to manage deployments.

**C. Serverless Deployments**

* Deploys AWS Lambda, Azure Functions, Google Cloud Functions.
* Built-in rollback in case of function failures.

**D. On-Premises & Hybrid Deployments**

* Supports hybrid cloud deployments with VPN or private link integration.

**E. Edge & IoT Deployments**

* Can deploy applications to edge devices or remote locations.

**3. Deployment Automation & CI/CD Integration**

* **Fully Automated Pipelines**: Supports **multi-stage deployments** with approvals, testing, and verification.
* **CI/CD Integrations**: Works with **Jenkins, GitHub Actions, GitLab CI/CD, CircleCI, and Bitbucket Pipelines**.
* **Infrastructure Provisioning**: Uses **Terraform, CloudFormation, and Pulumi** for pre-deployment infrastructure setup.

**4. Deployment Verification & Rollback**

**A. AI-Powered Deployment Verification**

* Uses machine learning to analyze logs and metrics for anomalies.
* Integrates with **Datadog, Splunk, New Relic, Prometheus, and AppDynamics**.

**B. Automated Rollback**

* If deployment verification fails, Harness **automatically rolls back** to the previous version.
* Supports **manual rollback triggers** and custom rollback scripts.

**C. Real-Time Monitoring & Alerts**

* Sends alerts to **Slack, Teams, PagerDuty** on deployment failures.
* Custom dashboards for tracking deployment health.

**5. Security & Governance in Deployments**

* **Role-Based Access Control (RBAC)**: Restricts who can deploy to production.
* **Audit Logs**: Tracks all deployment activities for compliance.
* **Secrets Management**: Works with **AWS Secrets Manager, HashiCorp Vault, CyberArk**.

**6. Example Deployment Pipeline in Harness (YAML)**

pipeline:

name: Canary Deployment Pipeline

identifier: canary\_deployment

stages:

- stage:

name: Deploy Canary

type: Deployment

service:

identifier: my-service

environment:

identifier: prod-env

execution:

steps:

- step:

name: Deploy Canary

type: KubernetesCanaryDeploy

spec:

canaryWeight: 10

- step:

name: Verify Deployment

type: HarnessApproval

spec:

approvers:

userGroups:

- SRE-Team

- step:

name: Full Deployment

type: KubernetesRollingDeploy

spec:

manifest:

identifier: app-manifest

valuesPaths:

- values-prod.yaml

**7. How Does Harness Compare to Other CD Tools?**

| **Feature** | **Harness CD** | **Spinnaker** | **ArgoCD** | **Jenkins (with plugins)** |
| --- | --- | --- | --- | --- |
| Deployment Strategies | ✅ All major | ✅ All major | ✅ Kubernetes-focused | ⚠️ Limited |
| Automated Rollback | ✅ Yes | ✅ Yes | ✅ Yes | ⚠️ Script-based |
| AI-Powered Verification | ✅ Yes | ❌ No | ❌ No | ❌ No |
| Multi-Cloud Support | ✅ Yes | ✅ Yes | ✅ Yes | ✅ Yes |
| Feature Flags | ✅ Yes | ❌ No | ❌ No | ⚠️ Requires plugins |
| Security & Compliance | ✅ Yes | ⚠️ Basic | ⚠️ Basic | ⚠️ Basic |
| Ease of Use | ✅ Very easy | ⚠️ Complex | ⚠️ Kubernetes only | ❌ Manual effort |

**Conclusion**

Harness **excels** in deployment capabilities due to its **automation, AI-powered verification, rollback mechanisms, and flexibility across cloud and on-prem environments**. Compared to tools like Spinnaker, ArgoCD, and Jenkins, Harness is **easier to use and more feature-rich for complex deployments**.

**What is the underlying database used**

Harness primarily uses **MongoDB** as its underlying database. MongoDB is chosen due to its **scalability, flexibility, and support for JSON-like document storage**, which aligns well with Harness’s microservices architecture.

**Why MongoDB?**

1. **Scalability** – Supports high-volume deployments across multiple environments.
2. **Flexible Schema** – Harness stores various pipeline configurations, deployments, and logs in JSON-like documents.
3. **High Availability** – Replica sets ensure fault tolerance.
4. **Fast Query Performance** – Optimized for querying structured and semi-structured data.

**Other Databases in Harness**

Besides MongoDB, Harness also integrates with other storage solutions depending on the feature:

* **TimescaleDB (PostgreSQL-based)** – Used for **time-series data**, such as deployment and performance metrics.
* **Elasticsearch** – Used for **log indexing and search**.
* **Redis** – Used for **caching and speeding up API responses**.
* **Amazon S3/GCS/Azure Blob Storage** – Used for **artifact storage and large logs**.

**Can You Change the Database?**

* No, Harness SaaS (cloud version) manages the database internally.
* If using Harness Self-Hosted (On-Prem), the database setup follows Harness’s standard configurations with **MongoDB** as the default.

**Which coding language to adopt if we chose the tool**

**1. Language Choices Based on Use Case**

| **Use Case** | **Recommended Language** | **Why?** |
| --- | --- | --- |
| **Pipeline Scripting** | **Bash, Python, Groovy** | Used for automation, pre/post-deployment scripts. |
| **Infrastructure as Code (IaC)** | **Terraform (HCL), YAML, Python** | Harness integrates with Terraform, CloudFormation, and Kubernetes manifests. |
| **Custom Plugins & Extensions** | **Java, Go, Python** | Harness SDK and APIs allow customization in these languages. |
| **API Integrations** | **Python, JavaScript (Node.js), Java** | REST & GraphQL APIs for external tool integrations. |
| **Configuration Management** | **YAML, JSON** | Harness pipelines, Kubernetes, Helm, and Ansible use YAML. |

**2. Breakdown of Coding Needs in Harness**

**A. Harness Pipelines & Workflows**

* **Defined in YAML** (Pipeline-as-Code).
* Example pipeline snippet:

pipeline:

name: My Deployment Pipeline

stages:

- stage:

name: Deploy

type: Deployment

execution:

steps:

- step:

name: Run Script

type: ShellScript

spec:

script: echo "Deployment successful!"

* Can include **Shell, Python, or Groovy scripts** for logic execution.

**B. Scripting for Automation & Hooks**

* **Groovy or Python** for pre/post-deployment tasks in **Jenkins, Ansible, or Terraform**.
* **Bash** for custom **deployment scripts**.
* Example **Python script** for an API call in a pipeline:

import requests

url = "https://harness.io/api/deploy"

response = requests.post(url, json={"service": "my-app"})

print(response.status\_code)

**C. Infrastructure as Code (IaC)**

* **Terraform (HCL)** for provisioning infrastructure before deployments.
* **Kubernetes (YAML)** for managing containerized deployments.
* Example **Terraform module**:

resource "aws\_instance" "example" {

ami = "ami-123456"

instance\_type = "t2.micro"

}

**D. API Development for Extensions**

* **Java, Python, or Node.js** for integrating external systems via Harness APIs.
* Example **Node.js API call to trigger a Harness deployment**:

const axios = require('axios');

axios.post('https://harness.io/api/deploy', { app: 'my-app' })

.then(response => console.log(response.data))

.catch(error => console.error(error));

**3. Which Language Should You Choose?**

* ✅ **For DevOps Engineers** → **Bash, Groovy, Python**
* ✅ **For IaC & Cloud Engineers** → **Terraform, YAML, Python**
* ✅ **For Backend API Integrations** → **Python, Java, Node.js**
* ✅ **For Kubernetes & Helm** → **YAML, Golang**
* ✅ **For Custom Plugins & SDKs** → **Java, Go**

**Conclusion**

Harness itself is **language-agnostic**, but **Python, Bash, Groovy, YAML, and Terraform** are the most useful languages for scripting, automation, and integration. If you need to build custom plugins or API integrations, **Java, Go, and Python** are excellent choices.

**If its installed in SMP form, what servers will it be installed in**

**1. Harness SMP Deployment Architecture**

Harness in SMP mode follows a **microservices-based architecture**, requiring multiple servers or VMs to handle different functions.

**Core Components & Server Requirements**

| **Component** | **Description** | **Recommended Server Type** |
| --- | --- | --- |
| **Harness Manager** (Main Control Plane) | Central orchestrator for all pipelines, deployments, and governance. | **App Server (4+ vCPUs, 16GB RAM, 100GB disk, Linux-based)** |
| **MongoDB** (Primary Database) | Stores pipeline configurations, deployments, logs, secrets. | **Database Server (8+ vCPUs, 32GB RAM, SSD storage, HA recommended)** |
| **Delegate Servers** (Agents) | Executes tasks like deployments, integrations, Terraform runs. | **Worker Nodes (4+ vCPUs, 8GB RAM, Linux/Windows)** |
| **CD Services** | Microservices handling deployment logic and workflows. | **App Server (same as Harness Manager)** |
| **Log Storage (Elasticsearch, S3, or MinIO)** | Stores deployment logs and analytics. | **Log Server (4+ vCPUs, 16GB RAM, 500GB SSD, Object Storage)** |
| **TimescaleDB (Optional)** | Stores time-series data for deployment monitoring. | **DB Server (if enabled, similar to MongoDB requirements)** |

**2. Server Deployment Models**

Harness SMP can be deployed in different infrastructure setups:

**A. Kubernetes-Based Deployment (Recommended)**

* Uses **Kubernetes clusters** (EKS, AKS, GKE, OpenShift).
* Components run as **pods in separate namespaces**.
* Requires:
  + **Control Plane** (Harness Manager, CD Services)
  + **Worker Nodes** (Delegates)
  + **Persistent Storage** (MongoDB, TimescaleDB, Logs)
* Best for **scalability & HA**.

**B. VM/On-Prem Deployment**

* Installed on **bare-metal or virtual machines (AWS EC2, Azure VMs, GCP Compute, VMware)**.
* Requires **manual scaling and maintenance**.
* Minimum recommended setup:
  + 1 **Application Server** (Harness Manager & CD Services)
  + 1 **Database Server** (MongoDB, TimescaleDB)
  + 1+ **Worker Nodes** (Delegates)
  + 1 **Log Server**

**C. Hybrid Deployment (Cloud + On-Prem)**

* Harness Manager runs in **the cloud**, while delegates run on-prem.
* Used when deploying to **air-gapped, regulated environments** (e.g., banking, healthcare).
* Requires **secure connectivity** between cloud and on-prem infrastructure.

**3. Server Requirements Summary**

| **Server Type** | **CPU** | **RAM** | **Storage** | **Purpose** |
| --- | --- | --- | --- | --- |
| **Harness Manager** | 4+ vCPUs | 16GB | 100GB | Core control plane |
| **Database Server (MongoDB)** | 8+ vCPUs | 32GB | SSD, HA | Stores all data |
| **Worker Nodes (Delegates)** | 4+ vCPUs | 8GB | 50GB | Executes deployments |
| **Log Server (Elasticsearch/S3)** | 4+ vCPUs | 16GB | 500GB+ | Stores logs & metrics |

**4. Additional Considerations**

✅ **High Availability (HA)** – Recommended for MongoDB, Harness Manager, and Delegate servers.  
✅ **Scaling Delegates** – Add more **Delegate servers** based on the number of deployments.  
✅ **Network & Security** – Requires secure **VPC setup, TLS encryption, and firewall rules**.

**Will it be able to deploy to both on-prem and cloud environments?**

**Yes, Harness can deploy to both On-Prem and Cloud environments.**

Harness is designed for **multi-cloud and hybrid deployments**, making it a great choice for organizations with mixed environments.

**1. Deployment Capability Across Environments**

Harness supports **automated deployments** to:

| **Environment Type** | **Supported?** | **Deployment Method** |
| --- | --- | --- |
| **On-Prem Servers** (Bare Metal, VMs) | ✅ Yes | SSH, WinRM, Ansible, Terraform |
| **Private Cloud** (VMware, OpenStack, Nutanix) | ✅ Yes | Kubernetes, Helm, Terraform |
| **Public Cloud** (AWS, Azure, GCP) | ✅ Yes | Kubernetes, CloudFormation, Terraform |
| **Hybrid Cloud** (On-Prem + Cloud) | ✅ Yes | Harness Delegates enable hybrid deployments |
| **Edge & IoT Devices** | ✅ Yes | Over-the-air (OTA) updates, IoT integration |

**2. How Harness Deploys to On-Prem and Cloud**

**A. On-Prem Deployments**

Harness can deploy applications to on-prem **VMs, Bare Metal, or Private Cloud** using:

* **SSH/WinRM** → Deploys artifacts to servers using secure connections.
* **Ansible, Chef, Puppet** → Automates configuration management.
* **VMware & OpenStack** → Supports deployments to virtualized environments.
* **Air-Gapped Mode** → Runs without internet access for high-security environments.

✔ **Example:** Deploying a Java app to an on-prem server using SSH:

pipeline:

name: On-Prem Deployment

stages:

- stage:

name: Deploy App

type: Deployment

execution:

steps:

- step:

name: Copy Artifact

type: SshCopyArtifact

spec:

destinationPath: "/opt/app/"

- step:

name: Restart Service

type: SshCommand

spec:

script: "systemctl restart myapp"

**B. Cloud Deployments**

Harness integrates with **AWS, Azure, GCP, and others** using:

* **Terraform & CloudFormation** → Automates cloud infrastructure provisioning.
* **Kubernetes & Helm** → Deploys to managed clusters (EKS, AKS, GKE).
* **Serverless Support** → Deploys AWS Lambda, Azure Functions, Google Cloud Functions.

✔ **Example:** Deploying an app to AWS using Terraform:

pipeline:

name: Cloud Deployment

stages:

- stage:

name: Provision Infrastructure

type: TerraformApply

spec:

configuration: aws-infra.tf

- stage:

name: Deploy Application

type: KubernetesRollingDeploy

spec:

cluster: eks-cluster

manifest: my-app-helm-chart

**C. Hybrid Deployments (On-Prem + Cloud)**

Harness **Delegates** enable deployments to both cloud and on-prem environments **from a single pipeline**.

* **Hybrid Mode** → Deploys workloads to on-prem **AND** cloud at the same time.
* **Multi-Cluster Kubernetes** → Supports hybrid Kubernetes clusters.
* **Multi-Region Deployments** → Deploys to AWS, Azure, GCP, and on-prem simultaneously.

✔ **Example:** Deploying to both **on-prem VMs and AWS Cloud** in one pipeline:

pipeline:

name: Hybrid Deployment

stages:

- stage:

name: Deploy to On-Prem

type: Deployment

execution:

steps:

- step:

name: Deploy on VM

type: SshCommand

spec:

script: "docker run -d my-app"

- stage:

name: Deploy to AWS

type: KubernetesRollingDeploy

spec:

cluster: eks-cluster

manifest: my-app-helm-chart

**3. Key Features for On-Prem + Cloud Deployments**

✅ **Unified Pipeline** – Deploy to on-prem & cloud from a **single workflow**.  
✅ **Multi-Cloud Support** – AWS, Azure, GCP, VMware, OpenStack, and on-prem VMs.  
✅ **Hybrid Security** – Works in **air-gapped and connected** environments.  
✅ **Rollback & Verification** – AI-powered **rollback if deployment fails**.  
✅ **Self-Hosted or SaaS** – Choose **SMP (self-managed) or SaaS** deployment.

**4. Summary: Why Choose Harness for Hybrid Deployments?**

🔹 **Deploy Anywhere** – Supports both on-prem & cloud environments.  
🔹 **Agent-Based Model** – Harness **Delegates** allow secure deployments to private networks.  
🔹 **Infrastructure Automation** – Integrates with Terraform, Ansible, Helm.  
🔹 **AI-Powered Rollbacks** – Auto-detects failures & rolls back automatically.  
🔹 **Multi-Cloud Native** – Seamlessly handles hybrid & multi-cloud workloads.

**What are the integration points associated with the tool and which are the mandatory ones**

**Harness Integration Points & Mandatory Components**

Harness provides **various integrations** to support deployment automation across cloud, on-prem, and hybrid environments. Below is a detailed breakdown of integration points and which ones are **mandatory vs. optional** based on your use case.

**1. Core Mandatory Integrations**

These are the minimum required integrations for **Harness to function properly**.

| **Integration Category** | **Examples** | **Why It's Mandatory?** |
| --- | --- | --- |
| **Source Control** (SCM) | ✅ GitHub, GitLab, Bitbucket, Azure Repos | Stores pipeline YAMLs & app code. |
| **Artifact Repositories** | ✅ Docker Hub, AWS ECR, JFrog Artifactory, Nexus | Stores deployable artifacts (images, JARs, etc.). |
| **Infrastructure Provider** | ✅ AWS, Azure, GCP, Kubernetes, VMware, On-Prem Servers | Required for provisioning & deployment targets. |
| **Harness Delegate** | ✅ Self-hosted on a VM, Kubernetes, or Cloud instance | Executes pipeline tasks & communicates with Harness. |

**Example: Basic Harness Setup (Minimum Requirements)**

* **GitHub** (SCM) → Stores pipeline YAMLs.
* **Docker Hub** (Artifacts) → Stores container images.
* **AWS EC2** (Infrastructure) → Deploys to cloud VMs.
* **Harness Delegate** (Agent) → Executes deployments.

**2. Optional (But Recommended) Integrations**

These integrations enhance **security, monitoring, and automation**.

| **Category** | **Examples** | **Purpose** |
| --- | --- | --- |
| **CI/CD Tools** | Jenkins, GitHub Actions, GitLab CI | CI pipeline triggers. |
| **Secret Management** | HashiCorp Vault, AWS Secrets Manager, CyberArk | Securely store credentials. |
| **Configuration Management** | Terraform, Ansible, Helm, CloudFormation | Automate infrastructure provisioning. |
| **Monitoring & Logging** | Datadog, Prometheus, Splunk, ELK, New Relic | Post-deployment verification & logging. |
| **ITSM & Ticketing** | ServiceNow, Jira | Change approvals & tracking. |
| **Security & Compliance** | Snyk, SonarQube, Aqua Security | Vulnerability scanning & compliance. |
| **ChatOps & Notifications** | Slack, Microsoft Teams, Email | Deployment alerts & notifications. |

**3. Integration Categories Explained**

**A. Source Code Management (SCM) – Mandatory**

Stores application code & pipeline configurations.  
✅ **Supported SCMs:**

* GitHub
* GitLab
* Bitbucket
* Azure Repos

✔ **Example:** Harness fetching pipeline YAML from GitHub

source:

repo: "https://github.com/org/repo"

branch: "main"

**B. Artifact Repositories – Mandatory**

Stores **Docker images, JAR files, binaries** for deployment.  
✅ **Supported Repos:**

* Docker Hub, AWS ECR, Azure ACR, GCP Artifact Registry
* JFrog Artifactory, Nexus, Google Container Registry

✔ **Example:** Using Docker Hub in a Harness pipeline

artifact:

type: Docker

repository: "my-docker-app"

tag: "latest"

**C. Infrastructure Providers – Mandatory**

Defines **where applications will be deployed**.  
✅ **Supported Environments:**

* **Cloud**: AWS, Azure, GCP, Kubernetes
* **On-Prem**: VMware, OpenStack, Bare Metal
* **Hybrid**: Private cloud + public cloud

✔ **Example:** Deploying to AWS EKS (Kubernetes)

infrastructure:

provider: AWS

cluster: "eks-cluster"

**D. Secrets Management – Recommended**

Stores **API keys, database credentials, SSH keys** securely.  
✅ **Popular Integrations:**

* HashiCorp Vault
* AWS Secrets Manager
* CyberArk
* Azure Key Vault

✔ **Example:** Using AWS Secrets Manager

secrets:

database\_password: aws-secrets/db-password

**E. Configuration Management – Recommended**

Manages **Terraform, Helm, Ansible, CloudFormation** deployments.  
✅ **Popular Tools:**

* Terraform (IaC)
* Ansible (Server config)
* Helm (Kubernetes)

✔ **Example:** Running a Terraform apply in Harness

infrastructure:

type: Terraform

script: "terraform apply -auto-approve"

**F. Monitoring & Logging – Recommended**

Verifies deployments & captures logs.  
✅ **Popular Integrations:**

* Datadog
* Prometheus
* Splunk
* ELK Stack (Elasticsearch, Logstash, Kibana)

✔ **Example:** Post-deployment verification with Datadog

verification:

type: Datadog

metrics: "response\_time"

threshold: "<500ms"

**G. ITSM (Change Management) – Optional**

Manages **approval workflows for deployments**.  
✅ **Popular Integrations:**

* Jira
* ServiceNow

✔ **Example:** ServiceNow integration for deployment approval

approval:

type: ServiceNow

ticket: "INC123456"

**4. Summary: Which Integrations Are Mandatory?**

| **Category** | **Mandatory?** | **Why?** |
| --- | --- | --- |
| **SCM (GitHub, GitLab, Bitbucket)** | ✅ Yes | Stores pipeline YAMLs. |
| **Artifact Repository (Docker, JFrog, Nexus)** | ✅ Yes | Stores deployable images. |
| **Infrastructure Provider (AWS, Kubernetes, On-Prem)** | ✅ Yes | Defines deployment targets. |
| **Harness Delegate** | ✅ Yes | Executes pipeline tasks. |
| **Secrets Management (Vault, AWS Secrets)** | ⚠️ Recommended | Secure credentials handling. |
| **Monitoring & Logging (Datadog, Splunk)** | ⚠️ Recommended | Tracks deployment health. |
| **ITSM (Jira, ServiceNow)** | ❌ Optional | Used for change approvals. |

**5. Final Recommendation**

* 🔹 **Bare Minimum Setup**: GitHub (SCM) + Docker Hub (Artifacts) + AWS (Infra) + Harness Delegate.
* 🔹 **Best Practice Setup**: Add Terraform, Secrets Management, Monitoring & Logging.
* 🔹 **Enterprise Setup**: Include ITSM (ServiceNow, Jira), Security (Snyk, Aqua), and ChatOps (Slack).

**Does it integrate with litmus and remedy tools?**

**Harness Integration with Litmus and BMC Remedy**

Yes, Harness can integrate with **Litmus Chaos** (for chaos engineering) and **BMC Remedy** (for ITSM/incident management), though the methods of integration vary.

**1. Litmus Chaos Integration (Chaos Engineering)**

✅ **Supported?** **Yes** (via Kubernetes CRDs & APIs)  
Harness integrates with **Litmus Chaos** to introduce controlled failures in Kubernetes environments to test application resilience.

**Integration Methods:**

1. **Harness CD + Litmus Chaos CRDs**
   * Define chaos experiments as Kubernetes **Custom Resources (CRs)** in Harness pipelines.
2. **Webhook Triggering**
   * Call **Litmus API** to start chaos tests during deployment.
3. **Harness STO (Security Testing Orchestration) Integration** *(Planned Feature)*

✔ **Example: Running a Litmus Chaos Test in Harness**

yaml

CopyEdit

pipeline:

name: Chaos Testing Pipeline

stages:

- stage:

name: Deploy App

type: KubernetesRollingDeploy

spec:

cluster: eks-cluster

manifest: my-app-helm-chart

- stage:

name: Chaos Testing

type: HttpStep

spec:

method: POST

url: "http://litmus-control-plane/api/experiments/run"

body: '{"experiment": "pod-kill", "namespace": "default"}'

**Use Cases**

* **Verify app resilience** against pod failures, network delays, or disk failures.
* **Automate chaos testing** in CI/CD pipelines before production deployments.

**2. BMC Remedy Integration (ITSM & Incident Management)**

✅ **Supported?** **Yes** (via Webhooks, APIs, and Custom Connectors)

Harness integrates with **BMC Remedy** for:

1. **Change Management Approvals** – Enforce ITSM approval before deployments.
2. **Incident Creation** – Create Remedy tickets automatically on failure.
3. **Automated Remediation** – Trigger rollback based on Remedy incidents.

**Integration Methods:**

1. **BMC Remedy API Calls (REST/SOAP)**
   * Use Harness **HTTP Step** to create/update Remedy tickets.
2. **Webhook Triggers**
   * Remedy triggers Harness pipeline actions.
3. **Custom Remedy Connector** *(Under Development in Harness Marketplace)*

✔ **Example: Creating a BMC Remedy Ticket from Harness**

yaml

CopyEdit

pipeline:

name: ITSM Change Approval

stages:

- stage:

name: Request Change

type: HttpStep

spec:

method: POST

url: "https://bmc-remedy-server/api/ticket"

headers:

Content-Type: "application/json"

body: '{

"summary": "Deployment Change Request",

"status": "Pending Approval",

"assignedGroup": "DevOps"

}'

**Use Cases**

* **Ensure compliance** by enforcing ITSM approvals before production changes.
* **Automate incident response** by triggering rollbacks based on Remedy ticket status.

**3. Summary: Harness + Litmus + Remedy**

| **Tool** | **Integration Type** | **Use Case** | **Supported in Harness?** |
| --- | --- | --- | --- |
| **Litmus Chaos** | Kubernetes CRDs, Webhooks, API Calls | Chaos testing & resilience validation | ✅ Yes |
| **BMC Remedy** | REST API, Webhooks, Custom Connectors | ITSM approvals & incident response | ✅ Yes |