Middleware Statement of Ownership of Liberty

This document provides a structured overview of the existing Liberty application architecture and defines Middleware ownership, responsibilities, and governance in collaboration with DevOps, Platform, and Security teams. It also includes an updated architecture diagram and RACI model.

# Table of Contents

1. 1. Introduction
2. 2. Existing Architecture Overview
3. 3. Existing Architecture Flow Diagram
4. 4. Middleware Ownership (Legacy Stack)
5. 5. RACI Matrix
6. **MiddleWare Statement of Ownership of Liberty:**

This document outlines the current technical and ownership structure for the Liberty application ecosystem deployed on OpenShift.  
The purpose is not to introduce new software, but to clarify service-level ownership and architectural responsibilities across Middleware, DevOps, Platform, and Security teams.

Middleware is establishing ownership of the Liberty application stack—including image lifecycle management, configuration, deployment standards, and vulnerability compliance—within the existing CI/CD framework managed by DevOps.

1. **Existing Architecture Overview:**

**How Jenkins, Nexus, and OpenShift work together today:**

* 1. Developers commit Java code to **Git repositories**.
  2. **Jenkins** (managed by DevOps) builds the code using Maven or Gradle.
  3. Build artifacts (JAR/WAR/EAR) are pushed to **Nexus** (Maven repository).
  4. Jenkins then containerizes the app using Liberty base image (provided by IBM).
  5. The container image is pushed to an **OpenShift internal registry**.
  6. OpenShift pulls the image from its internal registry for deployment to its **K8s** as a defined deployment.

**3. Existing Architecture Flow**

|  |
| --- |
| Commits Java code |

|  |
| --- |
|  |

**Developer / Source Code  
Commits Java code (Maven/Gradle)**

Builds code, runs tests, packages artifacts

**Jenkins (DevOps-managed)  
Builds code & tests  
Packages artifact**

|  |
| --- |
| Stores build artifacts (WARs, EARs, JARs) |

|  |
| --- |
|  |

|  |
| --- |
| Stores Liberty-WAS container images |

|  |
| --- |
|  |

**Nexus (Maven Repository)  
Stores JAR/WAR/EAR artifacts**

Pulls images and deploys pods

**OpenShift Internal Registry  
Stores Liberty-WAS Docker image**

**OpenShift Runtime  
Pulls image and deploys pods**

**4. Where Middleware must own (legacy stack)**

**1) Git (source of truth)**

* **Dockerfile/Containerfile for liberty-was base and app images**
* **Liberty server.xml / config** (kept as code; externalized via env/ConfigMap/Secret)
* **Helm/Kustomize templates** (readiness/liveness, JVM opts, resources)
* **Versioning/tagging scheme** (e.g., liberty-base:23.0.0.7-<patch> pinned by digest)

**2) Jenkins (DevOps runs Jenkins; Middleware owns Liberty logic inside the pipeline)**

* **Shared pipeline library or Jenkinsfile sections** for Liberty builds
* **Build steps** (Podman) to assemble Liberty base and app images
* **Tagging rules** (semver, build numbers, and digest pinning)
* **Quality gates required by Middleware** (e.g., run Trivy, generate SBOM, fail-on-high CVEs)
* **Credentials usage spec** (robot account to push to Nexus Docker repo; do not manage Jenkins infra)
* **Promotion logic** (e.g., tag/label rules from -dev → -qa → -prod in Nexus Docker)

DevOps still owns Jenkins itself (controller/agents/upgrades/runners). Middleware owns **what the pipeline does for Liberty**.

**3) Nexus (artifact)**

* **Maven repos used by the Liberty app** (policies for approved dependencies, quarantine if applicable)

Middleware needs **project-level admin/maintainer** on the relevant repos.

**4) OpenShift (cluster + internal image registry)**

* **ImageStreams** for Liberty base/app (if the cluster pulls from Nexus)
  + Define from which Nexus repo/tag/digest to track
* **BuildConfig/DeploymentConfig (or Deployments)** templates owned by Middleware for Liberty apps
* **Namespace-level runtime standards** (requests/limits, health probes, NetworkPolicy patterns)
* **ImagePullSecrets** configuration (to pull from Nexus Docker repo)
* **Rollout strategy** (maxSurge, maxUnavailable) and **PDBs/HPAs** for Liberty runtimes

Platform/DevOps owns the cluster and OpenShift registry infra; Middleware owns **the app/runtime manifests and image choices**.

**5) Release & patching (Liberty runtime lifecycle)**

* **Monthly patch cadence** for Liberty base image (digest pinned base → rebuild → retest → promote)
* **CVE remediation policy** (thresholds, exceptions/waivers with expiry)
* **Changelogs & deprecation windows** communicated to app teams
* **Compatibility matrix** (supported Liberty versions vs JDK/runtime flags)

**6) Security & compliance (in the legacy context)**

* **Define required checks** for Liberty images (Trivy scan, SBOM generation)
* **Library allow/deny posture** (coordinate with Security for Maven dependencies)
* **(If used) Image signing/trust**—old stacks often skipped; Middleware should propose/own the policy even if Platform/Security manages keys

5. RACI

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Area** | **Middleware** | | **DevOps** | **Platform/SRE** | **Security** |
| Liberty Dockerfile, server.xml, templates | | **R** | C | I | C |
| Jenkins pipeline logic for Liberty build/scan/push | | **R** (what) | **A/R** (how, run) | I | C |
| Nexus Docker hosted repo (project-level settings, retention for Liberty) | | **R** | I | **A** (service) | C |
| Maven proxy usage rules for Liberty builds | | **R** | C | **A** (service) | C |
| OpenShift runtime manifests (IS/BC/DC/Deployments) | | **R** | C | **A/R** (cluster ops) | C |
| Image promotion criteria (gates, tags) | | **R** | C | I | C |
| CVE thresholds / SBOM requirement | | **R/C** | I | I | **A/R** |
| Keys/signing (if present) | | C | I | I | **A/R** |

R = Responsible, A = Accountable, C = Consulted, I = Informed