DevOps Project Plan: OpenShift CI/CD Platform

Document Version: 1.0 | Date: February 2026 | Status: Initial Draft

Executive Summary

This document provides a comprehensive project plan for establishing a fully automated CI/CD platform on Red Hat OpenShift. Starting from scratch (with the OpenShift cluster already provisioned), the team will implement source control, build pipelines, artifact management, and automated deployment workflows for a multi-tier application stack comprising a React JS frontend, Spring Boot Cloud API layer, and Camunda 8.5 BPM backend.

The project spans 7 months, targeting four sequential environments: DEV → UAT → SIT → PRODUCTION. All pipeline workflows will use GitHub Actions, artifacts will be stored in Sonatype Nexus, and deployments will target OpenShift namespaces per environment.

Category	Technology	Purpose
Platform	Red Hat OpenShift	Container orchestration & deployment target
VCS	GitHub	Source code version control
CI/CD	GitHub Actions	Automated build, test, and deploy pipelines
Frontend	React JS	Web application UI
API Layer	Spring Boot Cloud	RESTful microservices API
BPM/Workflow	Camunda 8.5	Business process management & workflow automation
Artifact Repo	Sonatype Nexus	Docker images, Maven/npm package storage

1. Source Control Repository Strategy

1.1 GitHub Organization & Repository Structure

A GitHub Organization will be created to house all repositories. The recommended mono-repo-per-component strategy provides clear separation of concerns while enabling cross-component workflow triggers.

Repository Name	Type	Contents & Purpose
org/frontend-reactjs	Application	React JS source code, Dockerfile, env configs, unit tests
org/api-springboot	Application	Spring Boot Cloud source, pom.xml, Dockerfile, integration tests
org/bpm-camunda	BPM	Camunda 8.5 Helm charts, BPMN files, DMN tables, worker configs
org/infra-openshift	Infrastructure	OpenShift manifests, Helm charts, namespace configs, RBAC yamls
org/cicd-shared	Shared	Reusable GitHub Actions workflows, composite actions, scripts
org/config-repo	Configuration	Environment-specific configs (Spring Cloud Config compatible)

1.2 Branching Strategy

GitFlow-inspired branching strategy with environment promotion gates:

Branch	Environment	Trigger	Protection Rules
feature/*	Local / PR	On push – runs lint, unit tests	PR required to merge to develop
develop	DEV	On merge – build, push to Nexus, deploy DEV	1 reviewer, CI must pass
release/*	UAT	On branch cut – build, deploy UAT	2 reviewers, QA sign-off gate
release/*	SIT	Manual approval gate – deploy SIT	2 reviewers + Release Manager
main	PRODUCTION	Manual approval gate – deploy PROD	3 reviewers + CAB approval

2. Service Accounts & Secrets Management

2.1 Required Service Accounts

Service Account	System	Permissions Required	Stored As
sa-github-actions	GitHub	Repo read/write, Packages, Actions secrets	GitHub PAT (fine-grained)
sa-nexus-ci	Sonatype Nexus	Push to docker-hosted, npm-hosted, maven-hosted repos	GitHub Actions Secret: NEXUS_USER / NEXUS_PASS
sa-openshift-dev	OpenShift DEV	deploy, get, list, watch in dev namespace	OCP Service Account Token (per-env)
sa-openshift-uat	OpenShift UAT	deploy, get, list, watch in uat namespace	OCP Service Account Token (per-env)
sa-openshift-sit	OpenShift SIT	deploy, get, list, watch in sit namespace	OCP Service Account Token (per-env)
sa-openshift-prod	OpenShift PROD	deploy, get, list, watch in prod namespace (restricted)	OCP Service Account Token (per-env)
sa-camunda-deploy	Camunda 8.5	Deploy BPMN/DMN resources via Zeebe gRPC API	Camunda Client ID + Secret
sa-sonar	SonarQube (optional)	Scan projects, publish quality gates	SONAR_TOKEN secret

2.2 GitHub Actions Secrets Structure

Secrets are organized at the GitHub Organization level with environment-scoped overrides:

- Organization Secrets (shared): NEXUS_URL, NEXUS_USER, NEXUS_PASS, SONAR_TOKEN, CAMUNDA_CLIENT_ID, CAMUNDA_CLIENT_SECRET
- Environment Secrets (per-env): OCP_SERVER_URL, OCP_TOKEN, OCP_NAMESPACE
- Repository Secrets (app-specific): APP_specific overrides as needed

Note: Use GitHub Environments (DEV / UAT / SIT / PROD) with required reviewers and deployment protection rules to gate environment secret access.

3. CI/CD Pipeline Architecture

3.1 Frontend React JS Pipeline

File: .github/workflows/frontend-ci-cd.yml in org/frontend-reactjs

#	Stage	Tool	Actions
1	Checkout & Setup	actions/checkout	Checkout code, setup Node.js 20 LTS, restore npm cache
2	Install Dependencies	npm ci	Install exact dependencies from package-lock.json
3	Lint & Code Quality	ESLint, Prettier	Enforce code standards, fail fast on violations
4	Unit Tests	Jest, React Testing Library	Run unit tests, generate coverage report (threshold: 80%)
5	Build	npm run build	Production build with environment-specific .env injection
6	Containerize	Docker / Podman	Build Docker image with Nginx, tag with git SHA + version
7	Push to Nexus	docker push	Push image to Nexus Docker hosted repo: nexus/frontend:sha
8	Deploy to OpenShift	oc / kubectl + Helm	Helm upgrade --install using per-env values.yaml, rolling update strategy
9	Smoke Test	curl / Playwright	Health check endpoint validation post-deployment
10	Notify	Slack / Teams	Post build status, image tag, and deployment URL to team channel

3.2 Spring Boot Cloud API Pipeline

File: .github/workflows/api-ci-cd.yml in org/api-springboot

#	Stage	Tool	Actions
1	Checkout & Setup	actions/checkout	Checkout code, setup Java 21 (Temurin), configure Maven cache
2	Maven Build & Test	mvn verify	Compile, unit tests, integration tests, code coverage (JaCoCo 80%)
3	Static Analysis	SonarQube / SpotBugs	Quality gate enforcement, OWASP dependency vulnerability scan
4	Publish to Nexus	mvn deploy	Push JAR/WAR artifacts to Nexus Maven hosted repository
5	Containerize	Jib / Docker	Build Docker image (JVM or native), push to Nexus Docker repo
6	Deploy to OpenShift	Helm	Deploy with ConfigMap injection, env-specific secrets, liveness probes
7	API Health Check	curl /actuator/health	Validate Spring Boot Actuator health endpoint returns UP
8	Notify	Slack / Teams	Post deployment summary to team channel

4. Camunda 8.5 OpenShift Setup Pipeline

4.1 Camunda Installation Strategy

Camunda 8.5 Self-Managed will be installed via the official Camunda Helm chart into dedicated OpenShift namespaces. The CI/CD pipeline handles Helm-based installation and lifecycle management.

Components installed:

- Zeebe (workflow engine) – stateful set with persistent volumes
- Operate (monitoring UI) – deployment + route
- Tasklist (human task UI) – deployment + route
- Optimize (analytics) – deployment + route (licensed feature)
- Connectors – deployment for pre-built and custom connectors
- Elasticsearch – bundled data store (or external cluster recommended for PROD)
- Identity (Keycloak) – OIDC identity provider for Camunda components

#	Stage	Tool	Actions
1	Validate Prerequisites	oc CLI	Verify namespace exists, storage classes available, resource quotas set
2	OpenShift SCC Configuration	oc adm policy	Apply SecurityContextConstraints for Camunda pods (nonroot or custom SCC)
3	Add Helm Repo	helm repo add	helm repo add camunda https://helm.camunda.io && helm repo update
4	Create Namespace & Secrets	oc / kubectl	Create namespace camunda-{env}, inject license key, DB passwords, Keycloak secrets
5	Helm Install / Upgrade	helm upgrade --install	Install camunda/camunda-platform chart with env-specific values-{env}.yaml
6	Wait for Readiness	kubectl rollout status	Wait for all Camunda pods to reach Running/Ready state (timeout: 15min)
7	Configure Routes	oc expose / Route YAML	Create OpenShift Routes for Operate, Tasklist, Optimize, Identity UIs
8	Health Validation	curl + zbctl	Validate Zeebe gateway status, Operate API health, Keycloak realm
9	Create API Clients	Camunda Identity API	Create M2M client for sa-camunda-deploy, store credentials as GitHub secrets
10	Notify	Slack / Teams	Post installation summary with component URLs to team channel

4.2 BPMN/DMN Workflow Deployment Pipeline

File: .github/workflows/bpmn-deploy.yml in org/bpm-camunda

This pipeline deploys custom BPMN process definitions, DMN decision tables, and Camunda Forms to the target Camunda cluster using the Zeebe API.

#	Stage	Tool	Actions

1	Validate BPMN/DMN Files	bpmnlint / xmllint	Lint all .bpmn and .dmn files, validate against BPMN 2.0 schema
2	Detect Changed Files	git diff	Identify which BPMN/DMN/Form files changed to deploy only deltas
3	Authenticate to Camunda	Zeebe gRPC / REST	OAuth2 token exchange using CAMUNDA_CLIENT_ID + CAMUNDA_CLIENT_SECRET
4	Deploy Resources	zbctl / Camunda REST API	zbctl deploy resources *.bpmn *.dmn *.form -- deploy all changed resources
5	Verify Deployment	Camunda API	Query Operate API to confirm process definitions registered with expected version
6	Tag Release	gh CLI / git tag	Tag the deployment in git with version and environment label
7	Notify	Slack / Teams	Post list of deployed process definitions with version numbers

5. OpenShift Environment Configuration

5.1 Namespace Structure

Namespace	Environment	Components	Access
app-dev	DEV	frontend, api, camunda platform	Dev team full access
app-uat	UAT	frontend, api, camunda platform	Dev team deploy-only, QA read
app-sit	SIT	frontend, api, camunda platform	Release Manager + DevOps only
app-prod	PRODUCTION	frontend, api, camunda platform	CAB-gated, DevOps deploy only
nexus	Shared	Sonatype Nexus OSS/Pro	DevOps admin, CI write
monitoring	Shared	Prometheus, Grafana, Alertmanager	DevOps admin, team read

5.2 Key OpenShift Configuration Items

- ImagePullSecrets configured in each namespace pointing to Nexus Docker registry
- ResourceQuotas and LimitRanges set per namespace (stricter in PROD)
- NetworkPolicies enforcing namespace isolation with explicit ingress/egress allow rules
- PersistentVolumeClaims for Camunda Zeebe data, Elasticsearch, and Nexus storage
- Horizontal Pod Autoscaler (HPA) configured for frontend and API deployments
- OpenShift Routes with TLS edge termination for all external-facing services
- RBAC: RoleBindings mapping GitHub Actions service accounts to namespace roles
- OpenShift Secrets linked to GitHub Actions environment secrets via External Secrets Operator (recommended) or manual sync

6. Sonatype Nexus Repository Configuration

Repository Name	Type	Format	Purpose
docker-hosted	Hosted	Docker	Store all built Docker images
docker-proxy	Proxy	Docker	Proxy Docker Hub to reduce external pulls
docker-group	Group	Docker	Single pull endpoint combining hosted + proxy
maven-hosted	Hosted	Maven2	Spring Boot JARs, internal libraries
maven-central-proxy	Proxy	Maven2	Proxy Maven Central for offline builds
maven-group	Group	Maven2	Combined Maven dependency resolution
npm-hosted	Hosted	npm	Internal npm packages
npm-proxy	Proxy	npm	Proxy npmjs.com for frontend dependencies
helm-hosted	Hosted	Helm	Internal Helm chart repository

7. Project Timeline & Task Breakdown

Total Duration: 7 Months | Team: 2-3 DevOps Engineers | Start: Month 1

7.1 Phase Overview

Phase	Months	Focus	Key Deliverables
Phase 1	M1 – M2	Foundation & Setup	GitHub Org, repos, Nexus install, OCP namespaces, service accounts, base Helm charts
Phase 2	M2 – M3	CI Pipelines (DEV)	All 4 CI/CD workflows operational, DEV environment fully automated
Phase 3	M3 – M4	UAT Promotion	UAT pipeline gates, approval workflows, regression test integration
Phase 4	M4 – M5	SIT Hardening	SIT pipelines, performance test hooks, security scanning, DR runbooks
Phase 5	M5 – M6	PROD Readiness	PROD pipeline with CAB gate, rollback automation, monitoring dashboards
Phase 6	M6 – M7	Go-Live & Handover	PROD go-live, runbook documentation, team knowledge transfer, post-launch support

7.2 Detailed Task Breakdown

Phase 1: Foundation & Setup (Months 1–2)

#	Task	Owner	Duration	Acceptance Criteria
1.1	Create GitHub Organization, configure SSO/SAML with enterprise IdP	DevOps Lead	2 days	GitHub Org accessible via SSO, MFA enforced
1.2	Create all 6 repositories with branch protection rules and .gitignore templates	DevOps Engineer	2 days	All repos visible, default branch protected, PRs required
1.3	Install Sonatype Nexus on OpenShift (nexus namespace), configure persistent storage	DevOps Engineer	3 days	Nexus UI accessible, storage mounted, admin password rotated
1.4	Configure Nexus repositories (Docker, Maven, npm, Helm) with cleanup policies	DevOps Engineer	2 days	All repo types operational, CI user can push/pull
1.5	Create OpenShift namespaces (dev/uat/sit/prod/nexus/monitoring) with ResourceQuotas	DevOps Lead	2 days	6 namespaces active, quotas enforced, no default SA leaks
1.6	Create and configure all service accounts (GitHub SA, Nexus CI SA, OCP SAs per env)	DevOps Lead	3 days	All SAs created, RBAC applied, secrets stored in GitHub

1.7	Configure GitHub Organization secrets and Environment secrets per environment	DevOps Engineer	1 day	Secrets accessible from test workflow, masked in logs
1.8	Configure OpenShift ImagePullSecrets for Nexus registry in all namespaces	DevOps Engineer	1 day	Test pod can pull from Nexus Docker registry
1.9	Create base Helm charts for frontend, API, and Camunda with per-env values files	DevOps Engineer	5 days	Helm charts lint successfully, values.yaml files for all 4 envs
1.10	Set up shared GitHub Actions reusable workflows in org/cicd-shared repository	DevOps Lead	3 days	Reusable workflows callable from any repo in the org
1.11	Install and configure Camunda 8.5 on DEV namespace via Helm	DevOps Engineer	4 days	Zeebe, Operate, Tasklist accessible, health checks green
1.12	Install Prometheus + Grafana in monitoring namespace, configure Camunda & app dashboards	DevOps Engineer	3 days	Grafana dashboards showing pod metrics, Zeebe metrics visible

Phase 2: CI Pipelines – DEV Environment (Months 2–3)

#	Task	Owner	Duration	Acceptance Criteria
2.1	Develop and test Frontend React JS CI pipeline (lint, test, build, containerize, push to Nexus)	DevOps Engineer	4 days	Pipeline runs end-to-end on develop branch merge
2.2	Develop and test Spring Boot Cloud API CI pipeline (Maven build, test, Jib containerize, push)	DevOps Engineer	4 days	Pipeline runs, JaCoCo report published, image in Nexus
2.3	Develop Camunda installation pipeline for DEV – parameterized Helm install workflow	DevOps Engineer	3 days	Pipeline can install/upgrade Camunda in DEV on demand
2.4	Develop BPMN/DMN deployment pipeline with lint validation and zbctl deploy	DevOps Engineer	3 days	BPMN deployed to DEV Camunda, visible in Operate
2.5	Configure frontend and API deployment to DEV namespace via Helm (rolling update)	DevOps Engineer	3 days	Applications reachable via OCP Route in DEV
2.6	Add smoke test and post-deployment health check steps to all pipelines	DevOps Engineer	2 days	Pipeline fails and rolls back on failed health check
2.7	Integrate OWASP Dependency Check and container image vulnerability scanning (Trivy)	DevOps Lead	2 days	Scan reports published, CRITICAL CVEs fail the pipeline
2.8	Set up Slack/Teams webhook notifications for pipeline success/failure	DevOps Engineer	1 day	Team channel receives pipeline outcomes in real-time
2.9	DEV environment end-to-end integration test: frontend → API → Camunda workflow execution	DevOps Lead + Dev Team	3 days	Full user journey completes in DEV end-to-end

Phase 3: UAT Environment (Months 3–4)

#	Task	Owner	Duration	Acceptance Criteria
3.1	Install Camunda 8.5 on UAT namespace with UAT-specific Helm values (sizing, config)	DevOps Engineer	2 days	Camunda UAT cluster healthy, separate from DEV
3.2	Extend all pipelines with UAT stage: add GitHub Environment approval gate (QA sign-off)	DevOps Lead	3 days	Deployments to UAT require manual approval in GitHub Actions
3.3	Configure UAT-specific Helm values (resource limits, replica counts, env variables)	DevOps Engineer	2 days	UAT deployments use correct config, isolated secrets
3.4	Integrate automated regression test suite execution post-UAT deployment	DevOps + QA Team	4 days	Test results published to pipeline, failures block promotion
3.5	Document UAT pipeline runbooks and environment access guide for QA team	DevOps Lead	2 days	QA team can independently trigger UAT deployments and view logs

Phase 4: SIT Environment & Hardening (Months 4–5)

#	Task	Owner	Duration	Acceptance Criteria
4.1	Install Camunda 8.5 on SIT namespace with production-like Helm values	DevOps Engineer	2 days	Camunda SIT mirrors PROD sizing and config
4.2	Add SIT stage with dual-approver gate (Release Manager + DevOps) to all pipelines	DevOps Lead	2 days	SIT deployments require 2 approvals in GitHub Environments
4.3	Integrate performance/load testing pipeline step (k6 or Gatling) on SIT	DevOps + QA Team	4 days	Load test reports published, SLO thresholds enforced
4.4	Implement automated rollback pipeline: on failed deploy, roll back to previous Helm revision	DevOps Lead	3 days	Pipeline auto-rolls back and notifies team on failure
4.5	Conduct security hardening review: RBAC, NetworkPolicies, secret rotation, SCC audit	DevOps Lead	3 days	Security review checklist signed off, no critical findings open
4.6	Configure Alertmanager rules for Camunda, API, and frontend SLOs	DevOps Engineer	2 days	Alerts fire to Slack/email on SLO breach in SIT
4.7	DR simulation: test namespace restore from backup, Camunda state recovery from Elasticsearch snapshot	DevOps Lead	3 days	DR runbook validated, RTO/RPO objectives met

Phase 5: Production Readiness (Months 5–6)

#	Task	Owner	Duration	Acceptance Criteria

5.1	Install Camunda 8.5 on PROD with HA configuration (3x Zeebe brokers, clustered ES)	DevOps Lead	3 days	PROD Camunda cluster HA, no single point of failure
5.2	Build PROD deployment pipeline with CAB approval gate and release note requirement	DevOps Lead	3 days	PROD pipeline waits for CAB-designated approver before proceeding
5.3	Implement blue-green or canary deployment strategy for frontend and API on PROD	DevOps Lead	4 days	Zero-downtime deployment verified with traffic splitting
5.4	Set up backup jobs: Elasticsearch snapshot schedule, Nexus blob store backup, OCP etcd backup	DevOps Engineer	2 days	Backup jobs scheduled, snapshots verified restorable
5.5	Complete PROD Grafana dashboards: SLA tracking, Camunda process metrics, business KPIs	DevOps Engineer	3 days	Dashboards show real-time PROD metrics and historical trends
5.6	PROD readiness review: pipeline walkthrough, security sign-off, ops runbook review	DevOps Lead + Stakeholders	2 days	All go/no-go checklist items signed off by stakeholders

Phase 6: Go-Live & Handover (Months 6–7)

#	Task	Owner	Duration	Acceptance Criteria
6.1	Execute PROD go-live: deploy all components with CAB approval, monitor for 48hrs	DevOps Lead	3 days	All PROD components live, no P1/P2 incidents in first 48hrs
6.2	Write comprehensive pipeline and architecture documentation in Confluence/Wiki	DevOps Lead	4 days	All pipelines, configs, and runbooks documented and reviewed
6.3	Conduct hands-on knowledge transfer sessions for ops and development teams	DevOps Lead + Engineers	3 days	Team can independently trigger pipelines, debug failures, and deploy BPMN
6.4	Post-launch monitoring, incident response, and pipeline fine-tuning period	DevOps Team	2 weeks	SLO compliance maintained, on-call runbook validated
6.5	Project retrospective, lessons learned documentation, and backlog for future improvements	DevOps Lead + PM	1 day	Retro completed, improvement backlog created and prioritized

8. Risks & Mitigations

Risk	Likelihood	Impact	Mitigation
Camunda 8.5 SCC compatibility issues on OpenShift	High	High	Pre-validate in DEV early (Month 1), use nonroot SCC template, engage Camunda support
GitHub Actions runner hitting OCP API rate limits	Medium	Medium	Deploy self-hosted GitHub Actions runners inside OCP cluster for internal API calls
Nexus storage exhaustion blocking CI pipeline artifact pushes	Medium	High	Implement cleanup policies from day 1, set storage alerts at 70% capacity
BPMN versioning conflicts during concurrent deployments	Medium	Medium	Use Zeebe resource-based versioning, enforce single deployer service account per env
Key team member unavailability during PROD go-live	Low	High	Cross-train at least 2 engineers on all pipeline components, maintain runbooks
Environment config drift between SIT and PROD	Medium	High	Use Helm value inheritance, automated config diff check as pre-PROD gate

9. Recommended Team Structure & Tools

9.1 Team Roles

Role	Responsibility	Key Skills Required
DevOps Lead (x1)	Architecture decisions, OCP RBAC, pipeline design, stakeholder comms	OpenShift, GitHub Actions, Helm, Camunda administration, security
DevOps Engineer (x2)	Pipeline development, Nexus config, Camunda Helm, monitoring setup	Docker, Kubernetes/OCP, CI/CD, Nexus, Prometheus/Grafana
Dev Team Liaison (x1)	Provide Dockerfiles, assist with app-specific pipeline configs, BPMN testing	React, Spring Boot, Camunda worker development
Release Manager (x1)	Coordinate UAT/SIT/PROD gates, CAB submissions, go/no-go decisions	Change management, release coordination, ITIL basics

9.2 Additional Recommended Tooling

- bpmnlint – BPMN file linting in CI pipeline (npm package)
- zbctl – Camunda CLI for Zeebe gateway operations and BPMN deployment
- Trivy – Container image and filesystem vulnerability scanning in CI
- OWASP Dependency Check – Maven plugin for Java dependency CVE scanning
- k6 or Gatling – Load and performance testing integrated into SIT pipeline
- External Secrets Operator – Sync secrets from vault into OpenShift Secrets automatically
- Renovate or Dependabot – Automated dependency update PRs for keeping images current
- GitHub Actions self-hosted runners on OCP – For faster, internal network access to cluster resources