Rewards Service — Twelve-Factor Compliance Guide

A practical checklist to ensure the Rewards microservice adheres to the Twelve-Factor App methodology. Tailored to our Spring Boot + OAuth2 + Redis + AWS (ECR/ECS or K8s) stack.

I. Codebase

One codebase tracked in Git, many deploys - Single Git repo (`rewards-service`) with trunk-based or short-lived branches. - Environments (dev/stage/prod) are separate **deploys** of the same commit. **Do:** Tag releases (e.g., `v0.1.3`), GitHub Actions build on every push, immutable image per commit. **Avoid:** Divergent forks per environment. **Acceptance:** `git rev-parse HEAD` matches image label/annotation in runtime.

II. Dependencies

Explicitly declare & isolate - Declare in `pom.xml` only; no transitive reliance by accident (use `mvn dependency:tree`). - Container isolates runtime — no reliance on host tools. **Do:** Pin critical versions; use Maven Enforcer to ban `SNAPSHOT` in prod. **Avoid:** Installing JDK/Redis on host as an app dependency. **Acceptance:** Build passes in clean container from scratch.

III. Config

Store config in the environment - All secrets/URLs via env vars or a secrets manager; no hard-coded values in repo. - Spring Boot reads via `SPRING_*`, `REWARDS_*`, or profile-specific env. - Example: `SPRING_REDIS_HOST`, `REWARDS_JWT_ISSUER`, `REWARDS_JWKS_URI`. **Do:** Use AWS Secrets Manager/Parameter Store and inject as env at deploy time. **Avoid:** Committing `application-prod.yml` with secrets. **Acceptance:** Container can start with only env vars; repo contains **no** secrets.

IV. Backing Services

Treat backing services as attached resources - Redis (ElastiCache), Postgres (RDS), Auth (OIDC/Keycloak/Okta), S3 are swap-able by configuration. - Access via URLs/hosts provided by env; use interfaces for clients. **Do:** Externalize Redis/Postgres endpoints; health checks via Actuator. **Avoid:** Embedding service credentials or assuming single vendor. **Acceptance:** Can switch Redis host from local to ElastiCache without code change.

V. Build, Release, Run

Strictly separate - **Build:** CI builds immutable Docker image per commit (`<repo>:<git-sha>`). - **Release:** Attach config to image producing a deployable release (K8s/ECS task def + env). - **Run:** Scheduler (K8s/ECS) runs release; no mutation at runtime. **Do:** Promote same image from dev→stage→prod; use provenance labels. **Avoid:** `docker exec` to patch live containers. **Acceptance:** SBOM + image digest logged in deployment metadata.

VI. Processes

Execute as one or more stateless processes - Horizontal scale via more replicas; no local session state. - Caching via Redis; uploads to S3; do not write to container FS (except `/tmp`). **Do:** Idempotent services; store correlation IDs in headers (MDC). **Avoid:** Sticky sessions or in-memory user sessions. **Acceptance:** Pod restart doesn't lose customer state.

VII. Port Binding

Export services via port binding - Spring Boot exposes HTTP on `:8080`; container `EXPOSE 8080`. - Use ALB/API Gateway/Ingress to route. **Do:** Health endpoints on `/actuator/health`. **Avoid:** Requiring an external app server on host. **Acceptance:** `curl http://localhost:8080/actuator/health` inside container succeeds.

VIII. Concurrency

Scale out via the process model - Configure replicas (K8s `Deployment.replicas` / ECS `desiredCount`). - Use connection pools; Resilience4j bulkheads for remote calls. **Do:** Tune JVM, thread pools, and DB pool via env (e.g., `JAVA_OPTS`, HikariCP sizes). **Avoid:** Single huge instance scaling vertically only. **Acceptance:** Load test shows linear(ish) throughput with replicas.

IX. Disposability

Fast startup/shutdown - Graceful shutdown: `server.shutdown=graceful`, preStop hook, `terminationGracePeriodSeconds`. - Idempotent retries; in-flight requests drained. **Do:** Spring `Lifecycle` hooks, health readiness probes. **Avoid:** Long non-interruptible startup tasks. **Acceptance:** Rolling updates complete with zero errors and minimal 5xx.

X. Dev/Prod Parity

Keep development, staging, and production as similar as possible - Same Docker image everywhere; config differs via env. - For local: in-memory cache (dev) but parity tests run with Redis (test docker compose). **Do:** Contract tests against mock/real upstreams; WireMock. **Avoid:** Local Windows-only scripts diverging from CI. **Acceptance:** Integration tests run in CI with dockerized Redis/Postgres.

XI. Logs

Treat logs as event streams - App logs to stdout/err with JSON or structured pattern; no log files in container. - Aggregated by CloudWatch/ELK; include `requestId`, `customerId` (when safe), and trace IDs. **Do:** Logback JSON encoder or MDC pattern; avoid PII. **Avoid:** Rotating files inside container. **Acceptance:** One click trace from API GW → service logs with correlation.

XII. Admin Processes

Run admin/one-off tasks as one-off processes - Use K8s `Job`/ECS one-off task for backfills, migrations, reindexing. - Same image, separate command (`java -jar ... --task=backfill`), read-only prod creds. **Do:** `@CommandLineRunner` guarded by profile/flag; RBAC-controlled. **Avoid:** SSH

into nodes and run ad-hoc scripts. **Acceptance:** Runbook documents repeatable commands and rollback.

Implementation Checklist (Rewards Service)

- [ ] **Git**: main + PR checks; semantic tags; CODEOWNERS.
- [ ] **CI**: Build image, run unit/integration tests, publish to ECR with tag `git-sha` + `semver`.
- [ ] **Config**: Remove secrets from repo; wire env via Secrets Manager; profile-free prod.
- [ ] **Health**: `/actuator/health`, readiness/liveness probes; Resilience4j metrics enabled.
- [ ] **Observability**: Micrometer + Prometheus (or CloudWatch), request/trace IDs, structured logs.
- [ ] **Security**: OAuth2 resource server; JWKS cache; mTLS optional for east-west.
- [ ] **State**: No local disk persistence; Redis for cache; RDS/S3 for durable data.
- [ ] **Release**: Immutable images; K8s/ECS manifests kept in infra repo; Helm/Terraform IaC.
- [ ] **Disposability**: Fast startup (<5s target), graceful shutdown, idempotent retries.

Example Env Vars (non-secret)

```
SERVER_PORT=8080
SPRING_PROFILES_ACTIVE=prod
SPRING_REDIS_HOST=redis.example.cache.amazonaws.com
SPRING_REDIS_PORT=6379
REWARDS_OIDC_ISSUER_URI=https://auth.example.com/realms/core
REWARDS_OIDC_JWKS_URI=https://auth.example.com/realms/core/protocol/openid-connect/certs
MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE=health,info,metrics,prometheus
RESILIENCE4J_RETRY_INSTANCES_UPSTREAMRETRY_MAX-ATTEMPTS=3
```

> Secrets (DB passwords, client secrets) must come from Secrets Manager/SSM and be injected as env or mounted files at deploy time.

Runbook Links (to create)

- Build & push image to ECR
- Deploy to ECS/K8s
- Rotate secrets (JWT issuer keys, DB creds)
- Rollback procedure
- Disaster recovery (Redis/RDS failover)

Owner: Platform Engineering · **Service:** rewards-service · **Last updated:** <set date>