CONTINUOUS INTEGRATION AND CONTINUOUS DEPLOYMENT (CI/CD) PROCEDURE

Document Reference: QMS-DEV-PRO-005

Version: 1.0

Classification: RESTRICTED - INTERNAL ENGINEERING ONLY

Role	Name & Signature	Position	Date
Prepared by			
Reviewed by			
Approved by			

TABLE OF CONTENTS

TABLE OF CONTENTS 2
1. AMENDMENT RECORD 4
2. GENERAL 5
2.1 Scope 5
2.2 Purpose 5
2.3 Guiding Principles 5
2.4 Responsibility for Implementation 6
2.5 References 6
2.6 Roles and Responsibilities 7
2.7 RACI Matrix 7
3. CI/CD PROCESS 8
3.1 Objectives 8
3.2 Key Performance Indicators 8
3.3 Resources 11
3.4 Pipeline Stages 12
Stage 1: Pre-commit Checks 12
Stage 2: Build and Dependency Resolution 12
Stage 3: Static Analysis and Secret Scanning 13
Stage 4: Unit Tests and Coverage Gate 13
Stage 5: Container Image Build, Scan, and Signing 13
Stage 6: Integration, Contract, and Performance Testing 14
Stage 7: Staging Deployment and Validation 14
Stage 8: Production Deployment 15
3.5 Supply Chain Security 15
3.5.1 Dependency Pinning 15
3.5.2 Internal Artefact Registry 15
3.5.3 Provenance and Attestation 16
3.5.4 Automated Dependency Update Policy 16
3.6 Emergency Pipeline Procedure 16
3.7 Quality Gates Summary 17
3.8 Environment Management 18
3.9 Version Control and Branching Standards 18
3.10 Process Map 19

3.11 Retained Documentation 19
4. RISKS AND OPPORTUNITIES 20
4.1 Risk Register 20
4.2 Opportunities 21
5. CONTINUOUS IMPROVEMENT 22
5.1 Improvement Cadences 22
5.2 Non-Conformance Handling 23
5.3 Document Control 23
APPENDIX A: SECURITY ADVISORY - SUPPLY CHAIN ATTACK PATTERNS AND MITIGATIONS 24
A.1 Attack Patterns Observed 24
A.2 Additional Controls 24
A.2.1 Publish-time Cooling-off Period 25
A.2.2 Post-install Script Controls 25
A.2.3 Dependency PR Review Protocol 25
A.2.4 CI Credential Isolation 26
A.2.5 Abandoned Dependency Monitoring 26
A.2.6 SBOM Differential Monitoring 26

1. AMENDMENT RECORD

Reviewed at minimum annually or following a significant pipeline failure, tooling change, security incident, or audit finding. All revisions are subject to the same approval requirements as the original document.

Date	Issue	Rev	Page	Subject	Revised By	Approved By

2. GENERAL

2.1 Scope

This procedure governs the design, operation, and maintenance of CI/CD pipelines for all software delivered to shared or production environments. It applies to:

- Application code: new features, bug fixes, security patches, and refactoring
- Infrastructure as Code: Terraform, Pulumi, Ansible, Crossplane, Helm charts, and equivalents
- Container images: base image builds, application image builds, and image promotion workflows
- CI/CD pipeline definitions: changes to pipeline configuration are subject to the same review and gate requirements as application code
- Database migration scripts and schema changes promoted through the pipeline
- Third-party dependency updates, including output from automated dependency management tooling
- Platform and developer tooling maintained by the DevOps or Platform Engineering function

Changes that affect only a developer's local environment and have not been pushed to a shared branch are out of scope. Manual deployments to any shared environment are prohibited except under the emergency procedure defined in Section 3.6.

2.2 Purpose

- Automate build, test, and deployment processes to eliminate manual error and reduce cycle time
- Enforce quality, security, and compliance gates consistently across all changes
- Maintain a complete, signed, and auditable artefact chain from source commit to production deployment
- Detect and block security vulnerabilities, supply chain risks, and policy violations before they reach shared environments
- Produce fast, actionable feedback to developers on gate failures
- Support reproducible, rollback-capable deployments across all environment tiers

2.3 Guiding Principles

- The pipeline is the only authorised path to any shared or production environment. Manual deployments require emergency procedure activation and produce a non-conformance record.
- Pipeline configuration is subject to version control, peer review, and the same change controls as application code.
- A single signed artefact is built from a tagged commit and promoted unchanged through all environments. Separate builds per environment are prohibited.
- Quality gates are defined per stage in Section 3.7. A gate that does not produce a blocking result on failure is classified as a monitoring control and is documented separately from pipeline gates.

- Pipeline stage ordering is determined by execution cost and dependency. Cheap, fast checks run first. Stages that have no inter-stage dependency are parallelised.
- Pipeline health is a tracked operational metric. Degraded pipeline reliability is treated as a defect and escalated accordingly.

2.4 Responsibility for Implementation

The DevOps or Platform Engineering Lead is accountable for this procedure. Specific responsibilities are distributed as follows:

- DevOps / Platform Engineer: pipeline design, configuration, and maintenance; environment management; artefact registry operations; pipeline health KPIs
- Engineering Lead: gate threshold ownership; code review policy; branch protection rule configuration; pipeline configuration change approvals
- Security Engineer: SAST/DAST tooling configuration; CVE threshold policy; secret scanning; supply chain security controls
- QA Lead: test suite pipeline integration; coverage threshold definition; integration and E2E test ownership
- Developers: conformance with branching and commit standards; resolution of pipeline failures on their branches within one business day

The engineer who configures the pipeline must not be the sole approver of their own pipeline configuration changes.

2.5 References

- ISO 9001:2015, Quality Management Systems
- ISO 27001:2022, Annex A controls 8.25 (Secure development life cycle) and 8.30 (Outsourced development)
- NIST SP 800-218, Secure Software Development Framework (SSDF)
- SLSA Framework (Supply-chain Levels for Software Artefacts), slsa.dev
- OpenSSF Scorecard, Best Practices for Open Source Projects
- OWASP CI/CD Security Top 10
- Internal: SDLC Procedure
- Internal: Code Review Procedure
- Internal: Release Management Procedure
- Internal: Incident Response Procedure
- Internal: Developer Guide / Engineering Handbook (where maintained)

2.6 Roles and Responsibilities

Role	Primary Responsibilities	Backup	Notes
DevOps / Platform Engineer	Pipeline design and maintenance; environment management; artefact registry; deployment tooling; pipeline health monitoring	Cross-trained team member	Pipeline pass rate and build success rate are KPIs owned by this role
Engineering Lead	Branch protection rules; gate threshold policy; code review requirements; pipeline configuration change approvals	Senior Engineer	Owns gate threshold definitions in Section 3.7
Security Engineer	SAST/DAST tooling configuration; CVE threshold policy; secret scanning; supply chain controls per Section 3.5	Security team rotation	Required approver for changes to security gate thresholds or scanning tool configuration
QA Lead	Test suite pipeline integration; coverage threshold definition; integration and E2E test ownership	Senior QA Engineer	Test failures that are not automatically blocking require explicit documented justification
Developers	Conformance with branching and commit standards; prompt resolution of pipeline failures	Peer developer	Pipeline failures on a developer's branch are that developer's responsibility to resolve within one business day

2.7 RACI Matrix

Activity	DevOps Eng	Developer	QA Engineer	Security Eng	Eng Lead
Pipeline design and config change	A/R	C	C	C	A
Branch protection rule configuration	C	I	I	I	A/R
Code commit and branch management	I	R	I	I	I
Code review	C	R	C	I	A
Automated test execution	A/R	C	R	C	I
SAST / DAST / secret scanning	C	I	C	A/R	I

Dependency and SCA scanning	A/R	C	C	A	I
Artefact signing and registry push	A/R	I	I	C	I
Staging deployment	A/R	C	C	I	I
Deployment approval (production)	A	C	R	C	R
Production deployment	A/R	I	I	I	I
Post-deployment monitoring	A/R	C	C	I	I
Pipeline health KPI review	A/R	I	C	C	R

R: Responsible. A: Accountable. C: Consulted. I: Informed.

3. CI/CD PROCESS

3.1 Objectives

- All production-bound changes pass through the gate sequence defined in Section 3.7
- The pipeline produces a single signed artefact built from a tagged, reviewed commit that is promoted unchanged through all environments
- Security and supply chain checks run on every pipeline execution
- Pipeline health metrics are collected from tooling output and reviewed at the cadences defined in Section 5
- Most gate failures are surfaced to the developer within 15 minutes of commit
- The pipeline configuration is version-controlled, peer-reviewed, and treated as production code

3.2 Key Performance Indicators

All metrics below are sourced from CI/CD platform, monitoring system, and incident tracker output. Manual collection of any listed metric indicates a tooling gap requiring remediation.

KPI	What it measures	Target	How collected	Breach response
Pipeline Pass Rate (first run)	Percentage of pipeline runs that pass all gates on the first execution without a retry or re-push	>=90%	CI platform build history	Breach triggers a review of pre-commit hook adoption and local test execution compliance across the team
Build Success Rate	Percentage of pipeline runs that complete a	>=98%	CI platform build history	Persistent breach indicates broken branch hygiene or dependency

	successful build (compilation, dependency resolution)			resolution issues; Engineering Lead reviews within 5 business days
Pipeline Execution Time (P95)	95th percentile elapsed time from commit trigger to pipeline completion across all stages	<15 min for application pipelines; <30 min for infrastructure pipelines	CI platform run timestamps	Breach triggers a stage-by-stage timing audit to identify parallelisation or caching opportunities
Deployment Frequency	Number of successful deployments to production per week	Weekly minimum for all teams; daily for teams with automated deployment approval	CI/CD deployment event log	Frequency below the weekly minimum for two consecutive sprints is escalated to the Engineering Lead for pipeline or process review
Lead Time for Changes	Elapsed time from code commit merged to trunk to production deployment	<1 day	Git commit timestamp to production deployment event in CI/CD log	Breach triggers a review of release scheduling, gate sequencing, and test suite execution time
Change Failure Rate	Percentage of production deployments that result in a degraded service, rollback, or attributed production incident	<5%	Production incidents and rollback events correlated with deployment records in the change management system	CFR above 10% in any rolling 30-day period triggers a mandatory gate adequacy review, documented as a non-conformance
Mean Time to Restore	Time from detection of a deployment-attributed production incident to confirmed service restoration	<1 hour	Incident tracker, filtered to deployment-attributed incidents	MTTR breach on a deployment-caused incident triggers a post-incident review of rollback procedure and pipeline design
Gate Bypass Rate	Number of times a defined pipeline gate was suppressed, overridden, or skipped, with or without approval,	Zero without Emergency Procedure activation; all activations reported in monthly KPI review	Pipeline execution logs, gate suppression audit trail	Any bypass without a documented approval and justification is an immediate non-conformance. Approved bypass events exceeding two per month for a

	per calendar month			single service trigger a root cause review
Test Coverage Delta	Change in test coverage percentage introduced by the merged change	No net decrease; new code >=80% covered	Coverage tooling output per pipeline run	Net decrease in coverage blocks the pipeline. Persistent coverage stagnation below threshold is flagged in the monthly QA review
SAST Finding Introduction Rate	New Critical or High severity static analysis findings introduced per 100 pipeline runs	Zero Critical; <2 High per 100 runs	SAST tool output per pipeline run	Critical findings are an immediate hard block. High finding rate above threshold triggers a quarterly SAST rule adequacy review
Dependency Vulnerability Introduction Rate	New Critical or High CVE findings in project dependencies introduced per 100 pipeline runs	Zero Critical; <2 High per 100 runs	SCA tool output per pipeline run	Critical CVE findings block the pipeline. Persistent High finding rate triggers a review of the automated update cadence and internal registry scan configuration
Secret Detection Rate	Secrets or credentials detected by secret scanning per month across all repositories	Zero secrets reaching any shared branch	Secret scanning tool output	Any secret detected on a shared branch requires immediate rotation of the exposed credential regardless of whether the commit has been reverted
Artefact Signing Coverage	Percentage of production-bound artefacts carrying a valid cryptographic signature and provenance attestation	100%	Artefact registry metadata	Coverage below 100% for any production-path repository is a non-conformance under this procedure
Flaky Test Rate	Percentage of pipeline test failures attributable to test non-determinism rather than code defects	<2% of pipeline runs	CI platform test result history; quarantine tag count	Flaky tests are quarantined within 48 hours of identification. Flaky test rate above 2% triggers a test suite reliability review

3.3 Resources

Tool selection within each category is a team decision. The categories are mandatory requirements.

Category	Reference Tools
Version Control	Git (GitHub, GitLab, Bitbucket, Azure DevOps). Branch protection rules enforced at infrastructure level.
CI Platform	GitHub Actions, GitLab CI, Jenkins, CircleCI, Tekton, Buildkite. Pipeline definitions version-controlled and reviewed on change.
Container Build	Docker, Podman, Buildah. Rootless builds preferred. Privileged container builds in CI are prohibited.
Container Orchestration	Kubernetes, OpenShift, Nomad. Deployment manifests are IaC, version-controlled, and peer-reviewed.
Artefact Registry	Harbor, JFrog Artifactory, AWS ECR, GCP GAR, Azure ACR. Internal registry is the sole source for shared environment deployments.
SAST	SonarQube/SonarCloud, Semgrep, CodeQL, Checkmarx. Rules reviewed quarterly; suppression audit monthly.
SCA / Dependency Scanning	Trivy, Snyk, Grype, OWASP Dependency-Check, Renovate/Dependabot for automated update PR generation.
Secret Scanning	GitLeaks, TruffleHog, GitHub Secret Scanning, GitGuardian. Enforced at pre-commit hook and CI gate.
Artefact Signing	cosign (Sigstore), Notary v2. Admission controllers reject unsigned artefacts at deployment.
SBOM Generation	Syft, CycloneDX tooling. SBOM attached to every release artefact; SBOM diff run on every build.
IaC Scanning	Checkov, tfsec, Terrascan, OPA/Conftest, kube-score.
DAST	OWASP ZAP, Nuclei. Run against the ephemeral environment in Stage 6 on every merge to trunk.
Test Frameworks	Jest, PyTest, JUnit 5, Go Test, RSpec, Playwright, Cypress. Coverage reporting integrated into pipeline.
Performance Testing	k6, Gatling, Locust, JMeter. Baseline recorded per release; regression gated in Stage 6.
IaC / Config Management	Terraform, Pulumi, Ansible, Crossplane. All shared environment configuration managed as code.
Monitoring and Observability	Prometheus, Grafana, Datadog, New Relic, OpenTelemetry. SLO dashboards confirmed active before every deployment window.
Feature Flags	LaunchDarkly, Unleash, Flagsmith, Flipt. Used for progressive delivery and emergency kill-switch capability.

Mutation Testing	PiTest (Java), Mutmut (Python), Stryker (JS/TS). Runs nightly as a scheduled async job; results surfaced as PR comment.

3.4 Pipeline Stages

Eight sequential stages. Stages with no inter-stage dependency are parallelised where pipeline runner capacity permits. The sequence below is the logical dependency order.

Stage 1: Pre-commit Checks

Trigger	Developer executes pre-commit hook before pushing to any remote branch
Steps	• Linting and code style enforcement (ESLint, Pylint/Ruff, gofmt, RuboCop, ktlint, or language-equivalent). • Secret and credential scanning (GitLeaks or equivalent). Commit is rejected if any credential pattern is detected. • Conventional commit message format validation. • File size check; binary file detection.
Gate	Commit rejected if any check fails. Developer resolves locally before pushing.
Note	Pre-commit hooks are configured and enforced in every repository. Repositories without pre-commit hooks are a non-conformance flagged in the monthly pipeline health review.

Stage 2: Build and Dependency Resolution

Trigger	Push to any branch triggers the CI pipeline
Steps	• Checkout from the exact commit SHA. • Dependency resolution from the internal artefact registry proxy. Direct public registry resolution is disabled in all CI environments. See Section 3.5. • Compilation or build verification. Failure blocks all subsequent stages. • SCA vulnerability scan (Trivy, Snyk, Grype, or OWASP Dependency-Check). Critical CVE findings block progression; High findings trigger automatic issue creation and Security Engineer notification. • Licence compliance check. Dependencies with prohibited licences are flagged and block progression. • Build artefact produced as an immutable, content-addressed object. Artefact hash recorded in the pipeline run log.
Gate	Build succeeds; no Critical CVE findings; no prohibited licences; artefact hash recorded in pipeline log.
Note	Dependency resolution uses pinned lock files (package-lock.json, poetry.lock, go.sum, Gemfile.lock, Cargo.lock). Lock files are committed to version control. CI fails if the lock file is out of sync with the dependency manifest. See Section 3.5.1.

Stage 3: Static Analysis and Secret Scanning

Trigger	Stage 2 success
Steps	• SAST scan on source code (SonarQube/SonarCloud, Semgrep, or CodeQL). Critical findings block; High findings require triage within 4 hours. • IaC policy scan where applicable (Checkov, tfsec, Terrascan, OPA/Conftest). Misconfigured IAM, open security groups, and missing encryption are classified as Critical. • Full secret and credential scan across the commit delta (TruffleHog, GitLeaks, or platform-native). • Code coverage report generated and recorded. Coverage delta gate is applied at Stage 4.
Gate	Zero Critical SAST findings; zero secrets detected; IaC policy violations within accepted threshold.
Note	SAST suppression comments require a justification string. Suppressions are audited monthly. Suppression rate is a tracked metric.

Stage 4: Unit Tests and Coverage Gate

Trigger	Stage 3 success
Steps	• Unit test suite executed against source. 100% pass required; any failure blocks. • Test coverage delta computed against the pre-merge baseline. Net decrease blocks the pipeline. New code must meet the coverage threshold defined by the QA Lead (default: 80%).
Gate	All unit tests pass; coverage delta non-negative; new code coverage at or above threshold.
Note	Unit tests and coverage run against source at this stage. Integration and contract tests run against the built container image in Stage 6, after the artefact is produced. Mutation testing is not part of the synchronous pipeline; it runs as a scheduled nightly job. Results are surfaced as a PR comment and tracked as a quality maturity indicator but do not block merge.

Stage 5: Container Image Build, Scan, and Signing

Trigger	Stage 4 success
Steps	• Container image built from the verified build artefact using a minimal, pinned base image. • Base image version pinned to a digest, not a mutable tag. • Image vulnerability scan (Trivy or equivalent) across OS packages and application layers. Critical CVE blocks; High triggers notification. • Image signed using cosign (Sigstore) or Notary v2. Unsigned images are rejected at deployment by admission controller. • SBOM generated and attached to the image manifest (Syft, CycloneDX, or SPDX tooling).

	• SBOM diff computed against the previous build's SBOM. New transitive dependencies not present in the previous SBOM are flagged for review. • Image pushed to the internal artefact registry with provenance attestation.
Gate	Image scan passes threshold; image signed; SBOM attached and diff reviewed; image in internal registry with provenance.
Note	All base images are mirrored to the internal registry and scanned before use. Images are not pulled directly from public registries in any shared environment.

Stage 6: Integration, Contract, and Performance Testing

Trigger	Stage 5 success; applies to merges to trunk or release branches
Steps	• Integration tests executed against the built container image in an ephemeral environment (docker-compose, Kind cluster, or equivalent). Environment is provisioned and torn down within the pipeline run. • Contract tests (Pact or equivalent) executed concurrently with integration tests where pipeline runner capacity permits. Integration tests and contract tests are parallelised; neither blocks the other's start. • DAST scan (OWASP ZAP or Nuclei) executed against the deployed application in the ephemeral environment. • Performance tests executed against the staged application; results compared against the recorded baseline. Regression above 10% on any defined SLO blocks progression. • Flaky test detection: any test that fails and passes on retry without a code change is flagged for quarantine.
Gate	Integration and contract tests pass; DAST within threshold; performance within 10% of baseline; no new flaky tests introduced.
Note	Integration and contract tests run against the container image produced in Stage 5, not against raw source. The ephemeral test environment is provisioned from IaC and has a defined TTL.

Stage 7: Staging Deployment and Validation

Trigger	Stage 6 success
Steps	• Signed artefact deployed to the staging environment via the pipeline. No manual deployment steps. • IaC diff executed to verify staging configuration matches production. Non-zero drift blocks promotion. • Smoke test suite executed against staging. • For major releases: UAT executed by the product owner or designated business stakeholders. Written sign-off required before Stage 8 proceeds.
Gate	Smoke tests pass; environment parity confirmed; UAT sign-off obtained for major releases.

Note	Staging environment configuration must be identical to production. Parity is enforced by IaC diff on every pipeline run to this stage.

Stage 8: Production Deployment

Trigger	Stage 7 success; release manager or designated approver sign-off recorded in the pipeline
Steps	• Deployment approval recorded as a pipeline gate. Approval is named and timestamped. • Production deployment executed via the pipeline using the signed artefact promoted from staging. • Rollout strategy applied per the release classification in QMS-DEV-PRO-004 (blue/green, canary, rolling, or feature-flag-gated). • Post-deployment smoke tests executed against production. • Observability stack confirms error rate, latency, and saturation metrics are within SLO thresholds. • Stabilisation window monitoring active. Duration per release classification in QMS-DEV-PRO-004. • Rollback pipeline confirmed operable before the deployment window opens. Confirmation is recorded in the release record.
Gate	Approval recorded; smoke tests pass; SLOs within threshold at end of stabilisation window.
Note	The rollback pipeline must be executed in staging and the result recorded in the release package before any major release deployment proceeds.

3.5 Supply Chain Security

All pipeline stages that resolve, fetch, build, or execute third-party dependencies must conform to the controls in this section. Appendix A contains the current security advisory on supply chain attack patterns and additional mitigations, updated March 2026.

3.5.1 Dependency Pinning

- All dependencies are pinned to exact versions in committed lock files (package-lock.json, yarn.lock, poetry.lock, go.sum, Gemfile.lock, Cargo.lock, requirements.txt with hashes).
- Lock files are committed to version control and are not regenerated in CI without a corresponding reviewed diff.
- Floating version ranges (^, ~, *, latest) in production dependency manifests are a non-conformance. They are permitted in local development tooling only.
- Base container images are pinned to a digest, not a tag.

3.5.2 Internal Artefact Registry

- CI environments resolve all dependencies from the internal registry proxy (Nexus, JFrog, Verdaccio, Athens, or equivalent). Direct public registry resolution is disabled in CI.

- The internal registry operates as a pull-through cache with scan-on-pull enabled. New packages are scanned before being cached.
- Packages may be quarantined in the registry following a post-cache vulnerability disclosure. Quarantined packages block all downstream builds until remediation is confirmed.
- Write access to production-path namespaces in the internal registry is restricted. Developers do not hold direct write permissions.

3.5.3 Provenance and Attestation

- Every artefact produced by the pipeline carries a provenance attestation recording: build system identity, source repository, commit SHA, build timestamp, and pipeline run ID.
- Provenance conforms to the SLSA specification at minimum Level 2. Tier-1 services target SLSA Level 3.
- Artefacts are signed using cosign with Sigstore's Rekor transparency log, or an equivalent in-house signing infrastructure.
- Admission controllers (Kyverno, OPA Gatekeeper, or Connaisseur) are configured to reject unsigned artefacts or artefacts without valid provenance from all production Kubernetes clusters.

3.5.4 Automated Dependency Update Policy

- Automated dependency update tooling (Renovate or Dependabot) is configured for all repositories. Update PRs are generated for human review; auto-merge is disabled.
- Dependency update PRs execute the full pipeline before any human review. A pipeline failure on an automated update PR is treated identically to a failure on a human-authored PR.
- Critical CVE remediation PRs generated by automated tooling are subject to a 4-hour review SLA. High CVE PRs: 24-hour SLA. All other updates: normal sprint cycle.
- Dependency update PRs are not batched with feature or fix changes. They are merged independently.

3.6 Emergency Pipeline Procedure

An emergency deployment bypasses specific pipeline stages under the controls defined below. Activation requires explicit authorisation and produces a non-conformance record.

- Emergency Procedure activation (EMAUTH) is restricted to gate failures in Stage 6 (integration, contract, and performance testing) and Stage 7 (staging deployment and validation). Stages 1 through 4 are non-bypassable under all circumstances. A failing unit test suite must be resolved in the branch before the pipeline can proceed.
- The Emergency Procedure is activated by the Release Manager or Engineering Lead and logged in the change management system within 30 minutes of activation.
- Stages eligible for scope reduction under an emergency: integration and contract tests (Stage 6), performance tests (Stage 6), UAT (Stage 7). Any reduction requires documented justification. Omitted stages are completed retrospectively within 24 hours.

- Each emergency activation produces an automatic non-conformance record. The record is reviewed at the next engineering review.
- More than two emergency activations in a rolling 30-day period for the same service require a mandatory root cause review before any further activations are permitted.

3.7 Quality Gates Summary

All gates below are hard blocks unless the block type column states otherwise. A hard block means the pipeline does not proceed. Proceeding past a hard block requires Emergency Procedure activation and produces a non-conformance record.

Stage	Gate Criteria	Block Type	Validation Method
Pre-commit	Lint passes; no secrets detected	Hard block -- commit rejected	Pre-commit hook (GitLeaks, language linter)
Build	Compilation succeeds; all dependencies resolved from internal registry; no Critical CVE in dependencies; no prohibited licences	Hard block	CI pipeline build job
SAST	Zero Critical findings; High findings triaged within 4 hours	Critical: hard block; High: triage gate	SonarQube / Semgrep / CodeQL
IaC Policy	No Critical misconfigurations	Hard block	Checkov / tfsec / OPA
Secret Scan	Zero secrets detected across commit delta	Hard block	TruffleHog / GitLeaks
Unit Tests	100% pass; coverage delta non-negative; new code at coverage threshold	Hard block	CI test runner with coverage tooling
Container Image Scan	No Critical CVE; base image pinned to digest; image signed; SBOM attached	Critical: hard block; High: triage gate	Trivy + cosign
Integration Tests	All critical path tests pass	Hard block	CI test runner, ephemeral environment
Contract Tests	All consumer contracts verified	Hard block	Pact or equivalent
Performance	No regression above 10% on defined SLOs vs recorded baseline	Hard block on regression	k6 / Gatling vs recorded baseline
DAST	No Critical findings; High findings triaged	Critical: hard block; High: triage gate	OWASP ZAP / Nuclei

Staging Smoke Tests	All smoke tests pass; IaC parity confirmed vs production	Hard block	Pipeline smoke test suite + IaC diff
Deployment Approval	Named approver sign-off recorded with timestamp	Hard block	Manual gate in pipeline platform

3.8 Environment Management

Environment	Purpose	Access Control	Data Handling	Configuration Source
Development (ephemeral)	Feature development; unit and integration test execution; pipeline stage validation	Developer who owns the branch; auto-destroyed after pipeline run or after 24-hour TTL	Synthetic data only; production data is prohibited	IaC, auto-provisioned per pipeline run; no persistent manual configuration
Integration / QA	Integration test execution; pipeline integration stage	DevOps, QA; no direct developer access to shared state	Synthetic data only; test fixtures version-controlled	IaC; drift alerts active; manual configuration changes are a non-conformance
Staging	Full pipeline validation; UAT; DAST; performance testing; release candidate validation	DevOps, QA, designated UAT participants	Masked or anonymised production data; re-identification prohibited; masking verified quarterly	IaC; configuration must be identical to production; parity enforced by IaC diff per pipeline run
Production	Live service delivery	DevOps on-call only; all access logged; privileged access via break-glass procedure with audit trail	Full production data governed by data classification and privacy policies	IaC only; manual configuration changes are prohibited; drift detection active with alerting on any deviation

3.9 Version Control and Branching Standards

Item	Standard	Example	Rationale
Branch naming	feature/{ticket-id}-short-description; bugfix/{ticket-id}; hotfix/{ticket-id}; release/{version}	feature/PLT-423-add-oauth-provider	Enables automated ticket linkage; pipeline applies gate profiles by branch type
Commit message format	Conventional Commits: type(scope): description. Types:	feat(auth): add PKCE flow for public clients	Enables automated changelog generation;

	feat, fix, chore, refactor, test, docs, ci, perf, security		provides clear intent for reviewers
Merge strategy	Squash merge to trunk for feature branches; merge commit for release branches	N/A	Squash keeps trunk history clean; merge commits on release branches preserve the release boundary
Branch lifetime	Feature branches: merged or closed within 2 days of opening. Release branches: maintained for the support lifetime of the release.	N/A	Branches open longer than 2 days accumulate merge risk and are flagged in the weekly pipeline health review
Release tagging	Semantic versioning: v{major}.{minor}.{patch}. Release tags are immutable; force-pushing to a release tag is prohibited.	v2.4.1	Immutable tags are required for reproducible builds and deployment audit trails
Lock file policy	Lock files committed and current. CI fails if the lock file is out of sync with the dependency manifest.	package-lock.json, poetry.lock, go.sum	Out-of-sync lock files result in unpinned dependency resolution in CI. See Section 3.5.1.

3.10 Process Map

The end-to-end pipeline flow is provided as a separate artefact in the link below. The map shows all eight pipeline stages, gate decision points, the emergency bypass path with non-conformance logging, and integration points with the Code Review, Release Management, and Incident Response procedures.

CI-CD Process map

3.11 Retained Documentation

Document Type	Content	Retention	Location	Access
Pipeline run logs	Stage-by-stage execution log; gate pass/fail status; timestamps	12 months	CI platform	Team
Build artefact records	Artefact hash, signing record, provenance attestation, SBOM, SBOM diff vs previous build	Product lifetime	Artefact registry	Team

SAST and SCA reports	Per-run scan results; finding history; suppression records with justifications	2 years	Security tooling platform	Security + DevOps
Secret detection logs	Detection events; rotation confirmation records	5 years	Security tooling / SIEM	Security only
Deployment records	Pipeline-to-production deployment events; approval records; rollout strategy	3 years	CI/CD platform + change management system	Team + Management
Gate bypass records	Emergency activations; approver; justification; retrospective completion record	5 years	QMS non-conformance register	Management
Dependency audit records	SCA scan history; CVE triage decisions; update PR merge records	2 years	SCA tooling + version control	Security + DevOps
Performance test baselines	Recorded SLO baselines per release; regression records	Product lifetime	Test management / analytics platform	QA + DevOps
Pipeline configuration history	Version-controlled pipeline definitions; PR review records	Lifetime of repository	Version control	Team
Mutation test reports	Nightly async job results; mutation score per module; trend history	6 months	CI platform / reporting tool	QA + Developers

4. RISKS AND OPPORTUNITIES

4.1 Risk Register

Reviewed quarterly. The DevOps / Platform Engineering Lead owns the risk register for this procedure. Risk status is a standing item at the quarterly engineering review.

Category	Risk	Mitigation
Technical	Pipeline configuration change disabling one or more gates	Pipeline configuration changes require peer review and a passing pipeline run on the configuration change itself before merge to trunk

Technical	Flaky test suite causing re-run behaviour rather than investigation	Flaky test rate tracked as KPI; any non-deterministic test failure is quarantined within 48 hours and treated as a defect
Technical	Internal artefact registry unavailability blocking all builds	Registry has a defined availability SLA and backup/restore procedure; CI fails fast with a diagnostic error and does not fall back to public registries
Technical	Performance test baseline drift masking genuine regressions	Baseline is re-recorded on every major release; gradual degradation above 5% over three consecutive releases triggers a baseline review rather than a silent update
Supply Chain	Malicious package introduced via compromised upstream dependency	Internal registry with scan-on-pull; dependency pinning enforced; publish-time cooling-off period; SBOM diff on every build. See Appendix A.
Supply Chain	CI credential exfiltration via malicious dependency post-install script	Short-lived OIDC tokens for cloud authentication; no long-lived static credentials in CI environment variables; post-install scripts disabled in CI where technically feasible. See Appendix A.
Supply Chain	Dependency confusion or typosquatting attack	Scoped registry configuration; internal packages resolved from internal registry only; no cross-fallback to public registry. See Appendix A.
Security	SAST suppression culture developing, masking real findings	Suppression requires justification; suppression rate tracked; wont-fix decisions re-evaluated quarterly; suppression audit monthly
Security	Deployed container image accumulating CVEs post-deployment	Continuous image scanning in registry post-push; alert on new CVEs against deployed image digests; base image rebuild cadence enforced
Process	Gate bypass becoming a routine workaround	Gate bypass rate tracked as KPI; each bypass creates a non-conformance record; root cause review triggered if pattern emerges
Process	Staging and production configuration diverging, causing staging-pass/production-fail failures	IaC diff is a hard gate before every production deployment; drift detection runs continuously with alerting on any manual change to shared environments
Compliance	Incomplete audit trail for regulated deployments	Pipeline run logs and gate results are immutable and retained per Section 3.11; provenance attestation provides a chain of custody from commit to production; bypass records are in the QMS non-conformance register

4.2 Opportunities

- SLSA Level 3 compliance: hermetic builds, isolated build environments, and two-party review. Increasingly required by enterprise procurement and regulated-sector RFPs.
- Pipeline-as-code templating: standardised, reusable pipeline definitions (GitHub Actions reusable workflows, GitLab CI includes, Tekton Pipelines) enforce gate standards across all repositories from a single source.

- Ephemeral environment tooling: on-demand environments per pipeline run (Crossplane, vCluster, Namespace-per-PR) eliminates shared test environment state as a reliability and data isolation risk.
- SBOM as a delivery artefact: SBOMs attached to every release enable downstream consumers in regulated industries to conduct their own vulnerability analysis against the organisation's released software.
- Progressive delivery as default rollout strategy: canary deployments and feature-flag-gated rollouts as the default for all production changes, not only major releases.

5. CONTINUOUS IMPROVEMENT

5.1 Improvement Cadences

Cadence	Trigger / Frequency	Scope	Required Output
Pipeline health review	Weekly	Pipeline pass rate; build success rate; execution time P95; flaky test count; gate bypass events	Status report; action item for any metric outside target with a named owner
Security tooling review	Monthly	SAST suppression audit; CVE triage backlog; dependency update SLA compliance; secret detection events	Updated suppression log; triage backlog cleared or escalated; SLA breach root cause recorded
KPI review	Monthly	All KPIs in Section 3.2; trend analysis	KPI report; action items for metrics outside target
Supply chain review	Quarterly	Dependency tree health; lock file currency; internal registry scan results; cooling-off policy effectiveness; SLSA posture assessment	Updated risk register; dependency remediation backlog; policy adjustments; Appendix A reviewed for new mitigations
Quarterly engineering review	Quarterly	Full KPI review; risk register update; tooling evaluation; gate threshold review; non-conformance review	Updated risk register; process change decisions; procedure amendment where required
Post-incident review	After any P1/P2 incident attributed to a pipeline failure or deployment	Pipeline gap analysis; gate miss analysis; supply chain incident timeline if applicable	Blameless post-mortem; pipeline action items; procedure update if a systemic gap is identified
Annual procedure review	Annually (minimum)	Full document review; compliance alignment; Appendix A updated with current threat intelligence; regulatory changes	Updated procedure version; amendment record; re-approval cycle

5.2 Non-Conformance Handling

The following conditions are non-conformances under this procedure and require a documented corrective action within 5 business days:

- A deployment to any shared environment executed outside the CI/CD pipeline without Emergency Procedure activation
- A production artefact that is unsigned or lacks provenance attestation
- A pipeline gate bypassed without a documented approver and justification
- A secret detected on a shared branch that was not immediately rotated
- A Critical SAST or dependency CVE finding that remained open for more than 24 hours without a documented triage decision
- A lock file out of sync with the dependency manifest in any repository running the pipeline
- Emergency Procedure activation more than twice in a rolling 30-day period for the same service
- A repository without pre-commit hooks configured, identified during the monthly pipeline health review

Non-conformances are logged in the QMS register and reviewed at the quarterly engineering review. Repeat non-conformances on the same item trigger a formal corrective action process.

5.3 Document Control

This procedure is reviewed whenever: a post-incident review identifies a pipeline gap; Appendix A is updated with new confirmed threat intelligence requiring a policy control change; a tooling change affects a defined stage; or the annual review cycle is due.

APPENDIX A: SECURITY ADVISORY - SUPPLY CHAIN ATTACK PATTERNS AND MITIGATIONS

Document reference: QMS-DEV-PRO-005-ADV-001

Advisory Status: Active. Last updated: March 2026.

This advisory documents supply chain attack patterns observed in production environments during Q1 2026, with particular concentration in the JavaScript/npm, Python/PyPI, and container base image ecosystems. Controls documented here are additions to the standing policy in Sections 3.5.1 to 3.5.4 and have been incorporated into the risk register in Section 4.1. This appendix is reviewed and updated on the quarterly supply chain review cycle.

A.1 Attack Patterns Observed

Pattern	Description
Typosquatting and dependency confusion	Malicious packages published under names one character different from widely-used packages, or under names that shadow internal packages on public registries. Triggered when CI resolves from public registries before internal ones, or when lock files are absent or unpinned.
Compromised maintainer account, backdoored version	A legitimate package maintainer's account is compromised and a backdoored version is published. Automated update tooling generates a merge-ready PR. Without a changelog review and diff inspection, the malicious version is merged and deployed.
Post-install script abuse	Packages with post-install hook definitions (postinstall in package.json; setup.py install hooks; Cargo build.rs) execute arbitrary code at install time inside the CI environment. In 2026 incidents, this vector was used to exfiltrate CI environment variables including signing keys and deployment credentials.
CI credential exfiltration via dependency	Malicious dependencies installed in the CI runner read and transmit environment variables. Long-lived static credentials stored in CI secrets are the primary target.
Abandoned package takeover	Packages with no active maintainer are targeted for namespace takeover or account recovery. Packages with high download counts and no commits in 18 or more months are the primary targets.
Mutable tag repointing in container registries	Container images referenced by mutable tags (latest, alpine:3) are silently repointed to a different, potentially compromised digest. Pipelines that do not pin base image digests pull the repointed image on the next build.

A.2 Additional Controls

The following controls are validated mitigations for the patterns in Section A.1. They supplement the standing controls in Sections 3.5.1 to 3.5.4. Teams in regulated environments (PCI-DSS, SOC 2 Type II, ISO 27001) should treat these as mandatory additions rather than recommendations.

A.2.1 Publish-time Cooling-off Period

Configure the internal registry proxy to impose a minimum age requirement before a newly published package version becomes available for dependency resolution in CI. Newly published versions are held in a quarantine state and are not resolved until the hold period elapses.

Ecosystem	Implementation	Hold Period
npm / Node.js	Verdaccio custom middleware with package age policy; Nexus staging repository policy; JFrog Xray quarantine rule	24 to 72 hours recommended; 24 hours minimum for Critical dependencies
Python / PyPI	pip-audit with custom age policy pre-install; Nexus PyPI proxy with quarantine; combined with hash-verified requirements.txt	24 to 48 hours recommended
Go	Go module proxy (GOPROXY) with internal Athens proxy; checksum database (sum.golang.org) provides integrity verification independent of age gating	Integrity via checksum database; age gating via Athens policy where required
Rust / Cargo	cargo-vet provides an explicit audit workflow; crates.io versions must pass an audit entry before use in production builds	Audit required before crate version is approved for production
Container base images	Digest pinning (Section 3.5.1) prevents mutable tag repointing. Base image pipeline scans the mirrored digest before promotion to the internal registry.	24-hour minimum between external mirror and internal promotion

A.2.2 Post-install Script Controls

- JavaScript / npm: configure .npmrc with ignore-scripts=true in all CI environments. Apply --ignore-scripts to all npm install and npm ci invocations in pipeline scripts.
- Audit all direct dependencies that require a post-install script. For each: confirm the script's function is legitimate, pin the dependency version, and record the justification. Unreviewed post-install scripts in direct dependencies are a non-conformance.
- Python / PyPI: use pip install --no-build-isolation where feasible; audit setup.py install hooks in direct dependencies.
- For languages or package managers where post-install script disabling is not available, the dependency is reviewed manually before addition and re-reviewed on every version update.

A.2.3 Dependency PR Review Protocol

When automated tooling (Renovate, Dependabot) generates a dependency update PR, the reviewing engineer must:

- Review the package's published changelog or release notes for the version delta being updated.

- Inspect any changes to post-install scripts, build scripts, or new executable files in the diff (npm: package.json scripts block; Python: setup.py or pyproject.toml; Rust: build.rs).
- Run a SBOM diff (syft diff or equivalent) between the old and new dependency manifest to identify unexpected new transitive dependencies.
- Verify the package maintainer's recent activity and account status for direct dependencies of high criticality or high download frequency.

A dependency update PR that passes the automated pipeline but fails the manual review criteria above must not be merged until the reviewer's concerns are resolved.

A.2.4 CI Credential Isolation

- CI pipelines must not hold access to production signing keys, deployment credentials, or registry publish tokens beyond the minimum required for that specific pipeline job.
- Cloud provider authentication in CI uses short-lived OIDC tokens (GitHub Actions OIDC, GitLab CI JWT, Workload Identity Federation). Long-lived static credentials stored in CI secret stores for cloud provider access are a non-conformance.
- Signing key access is via hardware security module or KMS. Private signing key material does not exist as a file or environment variable in any CI runner.
- CI secret configuration files (e.g., .env files committed to version control, hardcoded tokens in pipeline YAML) are covered by secret scanning in Stage 2 and Stage 3.

A.2.5 Abandoned Dependency Monitoring

- Direct dependencies with no commits in 18 or more months and a significant fork count are flagged in the quarterly supply chain review as elevated takeover risk candidates.
- OpenSSF Scorecard is run quarterly against all direct dependencies of tier-1 services. Scores below 4.0 require a documented remediation plan: vendor, fork, or replace.
- Dependencies identified as abandoned are assessed for: vendoring into the internal repository, forking under the organisation's namespace, or replacement with an actively maintained equivalent.

A.2.6 SBOM Differential Monitoring

- An SBOM is generated on every pipeline run (Stage 5). The SBOM is diffed against the immediately preceding build's SBOM.
- Any new transitive dependency appearing in the diff that was not in the previous SBOM is flagged for review before the artefact is promoted past staging.
- SBOM diffs are retained per Section 3.11. The diff record provides an audit trail of when each transitive dependency entered the build.

This advisory will be reviewed and updated on the quarterly supply chain review cycle. Updates that introduce new mandatory controls are incorporated into the procedure body via the standard amendment process.