SECURE SOFTWARE DEVELOPMENT LIFE CYCLE POLICY

Version Number:	2.1
Version Date:	10/08/2025
Owned by:	Security, GRC
Approved by:	Nate Fitch, Sr. V.P., Chief of Staff
Confidentiality Level:	Internal

Table of Contents

1. Purpose

This policy establishes a formal framework for embedding security into every stage of the software development lifecycle, from initial requirements through retirement. It ensures all software developed by or on behalf of Docker follows a structured Secure Software Development Lifecycle (SSDLC) that minimizes vulnerabilities and addresses risks before software reaches end users.

The policy mandates:

- Defined stages incorporating security controls: planning, design, implementation, testing, deployment, and decommissioning
- Threat modeling, secure coding practices, and validation against known vulnerabilities
- Static and dynamic code testing, with container scanning before release
- Embedded security tools (GitHub Advanced Security, SAST/DAST platforms, CI/CD enforcement)
- Mandatory secure coding training for all development personnel
- Production data protection, environment isolation, and third-party security assessments
- Traceable documentation and artifact retention for audits and incident response

This framework ensures collaboration between engineering, product, and security teams to build reliable, secure software while supporting compliance with ISO/IEC 27001, SOC 2, customer expectations, and regulatory requirements.

2. Scope

This policy applies to all software developed or maintained by Docker, whether it is intended for internal use or customer delivery. It includes source code repositories, APIs, infrastructure-as-code, container images, backend services, and front-end applications across all business units and product teams.

The policy governs development activities carried out in all environments, including development, testing, staging, and production. It applies to all personnel involved in the software development process, including full-time engineers, contractors, consultants, third-party development partners, and vendors who contribute code or participate in design and testing.

Whether software is built from scratch, derived from open-source components, or developed in collaboration with external partners, the requirements of this policy apply equally. Development projects that operate within Docker's CI/CD pipelines, use Docker-managed cloud environments, or deploy to Docker customers are all within the scope of this policy.

3. Roles and Responsibilities

Role	Responsibilities
Security Engineering Team	Owns the secure development policy and SSDLC framework. Conducts threat modeling sessions, reviews secure architecture, validates testing coverage, and delivers secure coding training. Leads triage and verification of security findings.
Software Engineering Teams	Follow SSDLC practices as part of their daily development activities. Perform unit and integration testing, apply secure coding standards, and remediate vulnerabilities identified through scanning or external testing. Document design and testing decisions in Jira.
DevOps / Platform Engineering	Integrate secure development tooling into CI/CD pipelines. Enforce code signing, scan container images, manage branch protection rules, and support secure infrastructure deployment.
Product Managers	Ensure security requirements are considered in product planning and included in functional specifications. Participate in risk discussions and coordinate remediation timelines for findings.
QA and Test Engineers	Execute automated and manual testing, including functional, regression, and UAT. Support SAST and DAST processes and validate fix effectiveness for reported vulnerabilities.
Third-Party Developers and Vendors	Must comply with Docker’s SSDLC policy and provide documented evidence of secure development practices. Complete security training and undergo code reviews as required.
Compliance and Legal	Review development documentation and policies for alignment with audit requirements. Coordinate with Security and Engineering on customer inquiries, attestations, and third-party assessments.

4. Definitions

Secure Software Development Lifecycle (SSDLC) - A structured framework that integrates security practices into each phase of software development, from initial requirements gathering through deployment and retirement, ensuring vulnerabilities are identified and mitigated before software reaches production.

Threat Modeling - A systematic process of identifying, analyzing, and documenting potential security threats to a system or application, evaluating attack vectors and vulnerabilities to inform design decisions and security controls.

Static Application Security Testing (SAST) - Automated security testing that analyzes source code, bytecode, or binary code for vulnerabilities without executing the program, identifying security flaws during the development phase.

Dynamic Application Security Testing (DAST) - Security testing methodology that evaluates running applications by simulating attacks to identify vulnerabilities in the deployed state, testing from an external perspective without access to source code.

Code Owners - Designated engineers or teams with authority and responsibility to review and approve code changes within specific repositories, modules, or components based on technical expertise and business domain knowledge.

5. SDLC Governance

All software development at Docker must adhere to the Secure Software Development Lifecycle (SSDLC) framework. This framework outlines structured stages including requirements gathering, secure design and threat modeling, secure coding practices, verification and validation, secure deployment, and formal retirement.

Each stage of the lifecycle incorporates specific security expectations. Requirements must document security objectives; designs must be reviewed for architectural weaknesses; implementations must be reviewed for secure coding adherence; and deployments must validate that risks are mitigated. Each phase must generate traceable artifacts and approvals logged in Docker’s ticketing systems.

Security Engineering must review and approve critical design decisions for business critical systems with output retained in a Docker approved ticket management system.

6. Security Reviews

Before any new product feature or service is deployed, it must undergo defined security reviews. This may include:

- Threat modeling based on business logic, data sensitivity, and architecture
- Architecture review for business critical systems or any externally exposed functionality
- Design approval by Security Engineering to ensure risks have been identified and mitigated

All documentation must be maintained in a Docker approved ticketing management system and linked to the corresponding release or deployment ticket prior to deployment.

7. Secure Development Controls

The following baseline controls must be enforced across all development activities:

- Source code repositories must have GitHub Advanced Security enabled for vulnerability detection, secret scanning, and dependency management
- All open-source software must have validated licenses and no known vulnerabilities at the time of integration; enforcement is provided through automated tooling.
- Secret scanning must be active on all pull requests, and embedded secrets must be remediated prior to merge
- Commit workflows must enforce branch protection, required approvals from code owners, and pass all CI checks before code is merged

8. Testing Controls

Comprehensive testing is required prior to deployment of any software component:

- Unit, integration, and user acceptance testing must be completed as part of each release cycle
- Static Application Security Testing (SAST) must run on every commit or pull request
- All container images must be scanned for known vulnerabilities before they are pushed to Docker registries

No deployment to production may occur without passing all required testing gates. Dynamic Application Security Testing (DAST) will be performed monthly on externally accessible applications and APIs.

9. Environment and Data Controls

Production data must never be copied into or used within development, test, or staging environments. All testing environments must use synthetic or anonymized datasets generated through automated data scrubbing tools. All test data must be reviewed and validated by Security Engineering prior to use.

Non-production environments must be segmented from production through network policies and role-based access controls. Monitoring tools must validate that these environments remain aligned with hardening standards and that unauthorized access attempts are detected.

10. Penetration Testing and Third-Party Review

Docker's customer-facing products are subject to independent penetration testing at least once per year. These tests are performed by qualified assessors and cover common security vulnerabilities such as those defined in the OWASP Top 10. The scope must include both frontend and backend services.

Findings from penetration tests must be triaged according to severity and remediated in accordance with Docker's Vulnerability Management Policy. Retesting is required for all high and critical findings.

Summary reports are reviewed internally by Security and made available to customers upon request through established trust and compliance channels.

11. Documentation and Auditability

To ensure traceability and audit readiness, Docker must retain documentation related to all stages of the development process. Required artifacts include:

- Jira tickets and commit histories associated with each change
- Threat models and associated risk assessments
- Security design reviews and architecture diagrams
- Testing reports, sign-offs, and validation results
- Records of secure development training participation

All documentation must be retained for a minimum of three years. These records support internal reviews, customer audits, and regulatory compliance requirements.

12. Enforcement and Exceptions

All Docker personnel are expected to comply with this policy in full. Failure to adhere may result in disciplinary action, up to and including termination of employment or contract. In cases involving willful misconduct, malicious activity, or violations of law, Docker may initiate legal proceedings. The organization reserves the right to monitor, audit, and inspect user activity on its systems to ensure compliance.

Any exceptions to this policy must be formally requested, documented, and approved by Docker's GRC or Legal team. Exception requests must clearly outline the business justification and associated risks, and should be submitted to Docker's Risk Submission Portal.

13. Change History

Date	**Version**	**Responsible Party**	**Description of Change**
10/08/2025	2.1	Nate Fitch, Sr. V.P., Chief of Staff	Revised policy approved
09/24/2025	2.0	Ian Wilson, Senior Security and Compliance Engineer	Policy Annual Review and Update
12/04/2024	1.4	Tushar Jain, Executive Vice President, Engineering	Revisions approval

11/14/2024	1.3	Karen Hajioannou, Senior Security & Compliance Engineer	Policy annual review and update
02/13/2023	1.2	Jean Laurent, SVP of Engineering	Policy Approved
12/20/2022	1.1	Rachel Taylor, Senior Manager, Information Security, Risk & Trust	Draft Submitted for review
12/13/2022	1.0	Rachel Taylor, Senior Manager, Information Security, Risk & Trust	Basic document outline