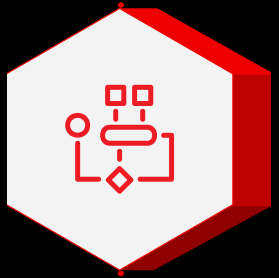
Red Hat

A practical guide to software supply chain security

Contents



Introduction

Security remains a top concern for organizations of all types and sizes.

Across industries, organizations rely on IT infrastructure and applications to manage operations, deliver services and products, and gain insight into their business. Security is a critical consideration for these systems and workloads. Data breaches and attacks can result in severe consequences for both businesses and their customers. In fact, the average cost of a data breach in 2023 reached an all-time high of US$4.45 million.[1]

Software supply chain attacks are of particular concern, as they take nearly 9% longer to identify and contain and result in a higher average cost of US$4.63 million.[1] And software supply chain attacks have increased by 742% annually, on average, over the past 3 years.[2]

It's no surprise, then, that 76% of CEOs say that protecting their partner ecosystem and supply chain is just as important as building their organization's cyber defenses.[3]

This e-book provides a practical guide for understanding and implementing software supply chain security in containerized and Kubernetes environments. We'll review the components and architecture of software supply chains, identify areas where vulnerabilities can be exploited, and provide best practices and guidelines for protecting your software supply chain.

The high cost of software supply chain attacks

Software supply chain security is critical for organizations that depend on applications and digital services to operate.

Average cost of a software supply chain attack:

US$4.63 million[1]

Average time to identify and contain a software supply chain attack:

294 days[1]

Share of data breaches originating from software supply chain attacks in 2023:

12%[1]

Annual increase in software supply chain attacks over the past 3 years:

742%[2]

1 *IBM Security. "**Cost of a Data Breach Report 2023**," July 2023.*
2 *Sonatype. "**8th Annual State of the Software Supply Chain**," October 2022.*
3 *KPMG. "**KPMG 2022 CEO Outlook**," 2022.*

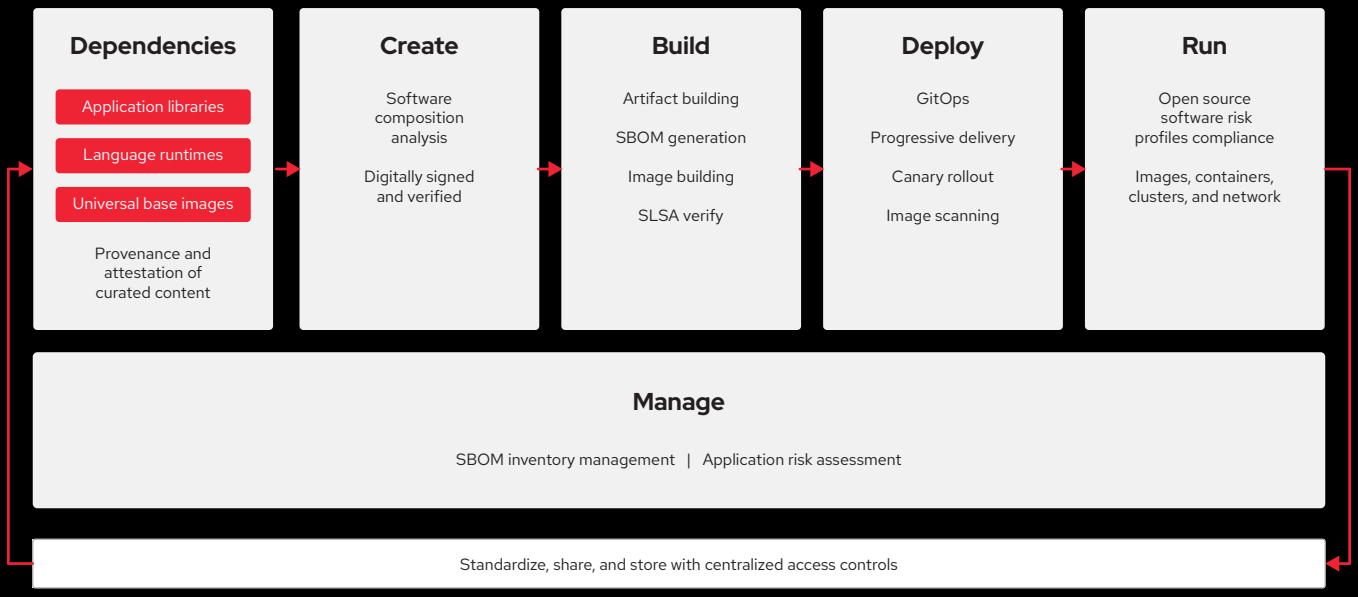
Understanding software supply chains

Software supply chains are the ecosystems in which software is developed, delivered, and deployed. They include everyone and everything that acts upon or impacts software during its life cycle. All people, components, libraries, tools, processes, and systems that create, build, deploy, and run software are part of the software supply chain.

The 2 main groups of actors in software supply chains are producers and consumers. Producers create and distribute software. Producers include software development companies, open source projects, and development teams in government and public sector organizations. Consumers use software. Consumers can be operations teams within the producer's organization, external software development organizations, government and public sector entities, and other businesses. In today's software ecosystem, all producers are also consumers, as anyone who builds software either uses or incorporates third-party tools or components when creating their own products.

While every software supply chain is unique, most follow a similar foundational model. Divided into 4 phases—create, build, deploy, and run—this model shows how sources and dependencies are transformed into artifacts that are integrated into other software or deployed and run as applications.

The following sections discuss the details of each phase in the software supply chain model.



Create phase

In the create phase, producers perform development- and operations-related tasks that add value to their software products.

First, developers and IT operations staff write and review sources. Sources are content that is written or thoroughly reviewed by the producer, stored in repositories, and used to build software products. Common sources include:

- Source code written by developers within the producer organization.
- Source code written by a third party that is thoroughly reviewed by the producer.
- Container images created by the producer's operations teams.
- Build tools and configurations written by producers to define how artifacts are transformed.
- Infrastructure as code (IaC) used to provision build resources.

Software architects and developers also evaluate and integrate dependencies during the create phase. Like sources, dependencies are also used to build software, but they are not written or reviewed by the producer. Common dependencies include:

- Open source libraries, modules, and components.
- Industry-standard software development frameworks.
- Third-party commercial middleware.
- Publicly available container images.

When evaluating dependencies, producers should consider the trustworthiness of the supplier and verify the authenticity of each dependency.

Finally, developers use security scan and test tools to detect potential vulnerabilities in both sources and dependencies. Common techniques and tools include software composition analysis (SCA), threat modeling, static application security testing (SAST), interactive application security testing (IAST), and dynamic application security testing (DAST).

Build phase

In the build phase, sources and dependencies are transformed into artifacts using build tools and platforms. Artifacts are the outputs of this phase and are published for use by others. Common artifacts include:

- Compiled binaries.
- Container images.
- Documentation.
- Software bills of materials (SBOMs).
- Vulnerability Exploitability eXchange (VEX) documents.
- Other attestations.

Build platform implementation varies between organizations. Common elements include:

- Compilers and related tools for transforming sources and dependencies into artifacts.
- Attestation tools that generate provenance on newly created artifacts.
- Functional test suites created within the software development organization.
- Continuous integration and continuous deployment (CI/CD) pipelines.
- Security scanning tools that may be integrated into CI/CD pipelines.

Finally, artifacts created during the build phase are stored in artifact repositories and published in package registries to make them available for use in the deploy and run phases.

The create and build phases are iterative. Software development and IT operations teams often use artifacts from the build phase to test and debug issues in sources and dependencies. Once issues are resolved, the build phase begins again.

Deploy phase

In the deploy phase, consumers access published artifacts and either use them as dependencies in software development projects or deploy them as workloads.

IT operations teams often use **GitOps** approaches to automate version-controlled software deployment, speed delivery of new features to users, promote collaboration between developer and operations teams, and increase consistency. Teams define deployment configurations using IaC and commit them to source repositories. Then consumers' CI/CD pipelines continuously and automatically download, test, and deploy updated versions of applications and infrastructure, based on these configurations.

Run phase

In the run phase, consumers run the resulting applications in hybrid cloud environments.

A secure runtime environment is essential and IT operations teams should configure the security features of the underlying software infrastructure—including the operating system and container management platform—accordingly. They should also ensure the runtime environment is maintained with the latest operating system and platform security updates. Finally, IT operations teams should monitor applications for reported vulnerabilities and active threats from both external and internal sources.

Understanding software supply chain attacks

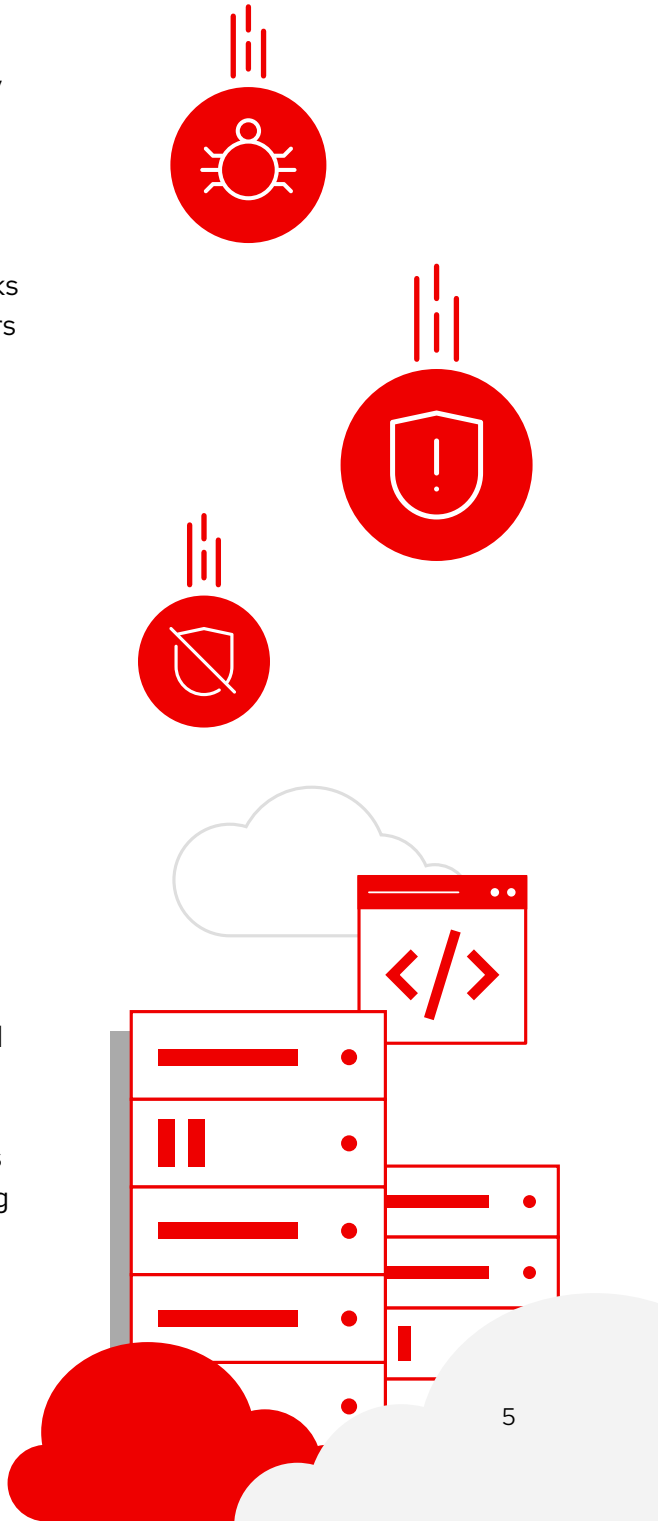
Software supply chain attacks encompass a range of malicious activities. Their primary objective is to compromise the security and integrity of software. These attacks can occur across all stages of the software supply chain, from development and build to deployment and runtime. They seek to inject unofficial or unauthorized behavior into software artifacts while avoiding detection. The intended targets of software supply chain attacks are both software vendors that produce artifacts and consumers of those artifacts.

An attack begins when an adversary gains access to systems used in a software supply chain, including repositories, build platforms, registries, and deployment pipelines. Common methods of access include phishing attacks, exploitation of security flaws in internet-facing applications, and malicious use of existing account credentials.

Once they have access, the adversary modifies the software product to include unauthorized behavior and waits for the target to deploy the compromised software. Adversaries can compromise software directly or via subsequent patches or hot fixes. And because artifacts are often used by many different consumers, compromising a single software vendor can result in multiple potential targets.

After the target deploys the compromised application, adversaries use their modifications to gain persistent privileged access to the target's systems, bypassing traditional perimeter security measures like border routers and firewalls. They can then inject additional malware and carry out malicious activities like data or financial theft, unauthorized monitoring, or disabling of networks and systems.

The following sections discuss how attacks can occur in different parts of the software supply chain.



Attacks involving dependencies

Software organizations often use a variety of third-party commercial and open source libraries, components, and services—each of which has its own dependencies. On average, commonly used libraries contain 5.7 dependencies and some libraries have more than 35 dependencies.[4] The number and transitive nature of dependencies results in a large attack area for adversaries to exploit.

Key attack vectors

- **Compromised dependencies:** Adversaries often try to introduce malicious behavior into commonly used third-party dependencies before producers download and use them in build processes.

Attacks during the create phase

Modern software development organizations store many types of content—including source code, infrastructure as code, and tool configurations—in source repositories. This variety makes source repositories attractive targets. Adversaries that attack repositories can potentially modify not only code but also infrastructure, CI/CD pipelines, and build platforms.

Key attack vectors

- **Unauthorized changes to sources:** Adversaries can attack sources before they are submitted to official repositories. They may also use integrated development environments (IDEs) or other developer tools to introduce malicious behavior into sources. Adversaries within your organization may use developer roles and access to directly add malicious behavior to sources. Additionally, any developer can unintentionally introduce vulnerabilities into sources that adversaries can exploit later.
- **Compromised source repositories:** Adversaries may use administrative interfaces or compromised underlying infrastructure to directly attack sources that are already stored in repositories.

4 Sonatype. "8th Annual State of the Software Supply Chain," October 2022.

Attacks during the build phase

Build platforms are typically multitenant systems that are tightly integrated with artifact repositories and package registries. They often provide self-service capabilities to multiple untrusted users simultaneously. The variety and complexity of systems integrated into build platforms make them attractive targets for adversaries.

Key attack vectors

- **Modified build sources:** Adversaries may attack build platform inputs—like source versions and locations and build steps and parameters—directing the platform to use compromised sources or dependencies.
- **Compromised build platforms:** Adversaries can attack any part of the build platform, including services, tools, and underlying infrastructure. They may perform builds that alter other builds or compromise artifacts in build caches. Finally, adversaries may modify the build platforms themselves to inject malicious behavior.
- **Compromised artifact repositories:** Adversaries can directly attack artifact repositories by uploading and publishing altered artifacts via an administrative interface or by compromising underlying infrastructure.

Attacks during the deploy phase

Organizations rely on automated processes to download artifacts and software that are deployed and run in their environments. Because installation, deployment, and update processes often retrieve software from external sources over the internet, they are attractive targets for supply chain attacks.

Key attack vectors

- **Altered artifacts:** Adversaries may attack artifacts that consumers download during installation or update processes. They may replace downloaded artifacts with ones containing malicious behavior or trick consumers into downloading or using compromised artifacts. Adversaries may also intentionally prevent consumers from downloading updates to ensure existing security vulnerabilities remain in place.
- **Compromised deployment processes:** Adversaries may modify consumers' deployment processes—including services, tools, and software infrastructure—to introduce compromised artifacts.

Attacks during the run phase

Nearly all aspects of modern business are dependent on software. Accordingly, applications and workloads running in cloud environments are attractive targets for adversaries.

Key attack vectors

Adversaries may attack targets' systems by exploiting vulnerabilities in published software due to:

- **Misconfigured software:** Overly permissive configurations for network or privileged access, or a lack of role-based access control (RBAC), can leave software vulnerable to attack.
- **Known vulnerabilities:** Insufficient vulnerability scanning during the previous phases can leave known vulnerabilities in new software.
- **Zero day vulnerabilities:** Unknown, or *zero day*, vulnerabilities are more likely to make it to the run phase without robust scanning during the previous phases.

Global responses to software supply chain attacks

These scenarios are not just theoretical. Software supply chain attacks like these have resulted in real data breaches. As a result, regulators worldwide are designing rules and recommendations to protect the global software supply chain. Here are some of the responses:

- The U.S. National Institute of Standards and Technology (NIST) developed **standards and guidelines to enhance software supply chain security** and govern the federal government's software procurement activities.
- The U.S. Cybersecurity and Infrastructure Security Agency (CISA) issued a **recommended practices guide for developers**, targeted at creating more secure software supply chains.
- The European Commission proposed the **Cyber Resilience Act** in September 2022, requiring software producers to have, among other things, a detailed understanding of all components within their product.
- The Association of Southeast Asian Nations (ASEAN) has **several initiatives** to improve international cooperation in the area of software supply chain security.

Protecting your software supply chain

Safeguarding your software supply chain requires a multifaceted approach. There are many things you can do to improve software supply chain security, and each will add another layer of protection for your organization and customers. The following sections discuss key concepts for building security into your software supply chain, along with best practices for implementation.

Software bill of materials

A software bill of materials (SBOM) is a nested inventory of all sources and dependencies—including source code, open source software and libraries, middleware, and development frameworks—that are part of an artifact. There are several industry-wide standards for SBOMs—**CycloneDX**, **Software Package Data Exchange (SPDX)**, and **Software Identification (SWID) tagging**, for example—that describe software composition using a uniform language that other tools can understand. Although SBOM contents may vary by company and standard, most contain a manifest, pedigree, and provenance.

- The **manifest** is a list of the components used to build an artifact. It includes baseline information like the author name, supplier name, version string, component hash, and a unique identifier for each component.
- The **pedigree** describes how the artifact was produced and descendant, variant, and ancestor relationships. It can also specify modifications made to open source components.
- The **provenance** describes the chain of custody or path of the software between organizations. It describes where and from whom each component was retrieved.

Industry best practices require SBOMs to be generated as part of the build process, stored in artifact repositories, delivered with associated artifacts, and reviewed regularly by consumers. Creating and maintaining SBOMs can help organizations avoid using and distributing potentially harmful software. Producers can ensure that all components are up to date and respond quickly to new vulnerabilities while consumers can perform vulnerability or license analyses on SBOMs to evaluate the risk of using or deploying an artifact.

Vulnerability Exploitability eXchange

Published by producers, a Vulnerability Exploitability eXchange (VEX) is a machine-readable document that tells consumers if artifacts are impacted by vulnerabilities in upstream components and, if affected, the recommended remediation actions to take. For any combination of artifact and vulnerability, the VEX status can be:

- **Not affected.** No remediation is required.
- **Affected.** Actions are recommended to remediate the vulnerability.
- **Fixed.** The product version contains a fix for the vulnerability.
- **Under investigation.** It is not yet known whether the artifact version is affected by the vulnerability.

Producers and consumers can integrate SBOM and VEX information for a current view of the status of vulnerabilities in software products. You can take a targeted approach to finding and remediating vulnerabilities and reduce the number of investigations into nonexploitable vulnerabilities.

Attestation

Used by producers to make statements about the artifacts they deliver, software attestations are trust mechanisms based on authenticated, machine-readable metadata. Whether included in an SBOM or delivered separately, attestations allow consumers to independently validate the integrity of these statements from producers.

While implementation details vary across organizations, all attestations include a subject, content or statement, and a signature.

- The **subject** is the list of artifacts to which the attestation applies.
- The **content** is 1 or more statements about the artifacts that the producer knows to be true.
- The **signature** is a tamper-resistant mechanism that denotes and authenticates the producer that created the attestation.

Although producers can attest any fact related to an artifact, there are several common types of attestations in a typical software supply chain. Attestations that inventory resources used by the build platform like server operating systems, cloud regions, and security groups, along with their current state, affirm the security of the build environment. Attesting build platform components like compilers and vulnerability scanning tools check that the appropriate tools are used to create and test artifacts. Identifying the provenance and pedigree of sources and dependencies in attestations helps consumers verify that the correct files were used to build artifacts. Since multiple artifacts are often created in a single build process, attesting relationships between artifacts helps producers and consumers track artifacts and ensure consistency.

Secure Software Development Framework

Published by NIST, the **Secure Software Development Framework (SSDF)** is a set of fundamental, security-focused practices that can be applied to software development life cycle models. Based on established standards, guidance, and documentation, the SSDF includes practices that both directly and indirectly impact software artifacts to help producers increase the security of their software development processes.

The SSDF groups best practices into 4 categories of recommendations:

- **Prepare the organization.** Ensure that people, processes, and technology are ready for security-focused software development.
- **Protect the software.** Safeguard software components from tampering and unauthorized access.
- **Produce well-secured software.** Produce security-focused software with minimal vulnerabilities.
- **Respond to vulnerabilities.** Identify vulnerabilities in software releases, address them as needed, and act to prevent similar occurrences in the future.

Producers and consumers can use these recommendations to strengthen their existing software development practices, establish requirements for third-party suppliers, and acquire software that meets fundamental best practices.

Supply-chain Levels for Software Artifacts

Supply-chain Levels for Software Artifacts (SLSA) is a set of security guidelines established by industry consensus to help producers protect their software supply chains and build confidence and trust with consumers.

A key component of these guidelines, SLSA provenance is an attestation that is created and signed by build platforms. It describes how artifacts are built, including the building entity, process, and inputs. Producers can follow SLSA **build requirements** to strengthen their software supply chain security. Consumers can follow SLSA **verification recommendations** to evaluate the trustworthiness of artifacts.

SLSA defines 4 build process levels that describe the trustworthiness and completeness of an artifact's SLSA provenance. While higher levels provide greater guarantees against supply chain threats, they incur higher implementation costs. For each artifact, producers can align with the build process level that meets their needs and implement higher levels as requirements change.

Software supply chain security best practices

There is no one best way to secure your software supply chain. Instead, there are many best practices—based on the concepts discussed in the previous sections—that you can use to incrementally improve your software supply chain security.

Best practices that apply to sources

- Educate developers on and enforce secure coding best practices.
- Perform security-focused peer code reviews, analysis, and testing.
- Require and automate source code scanning with software linting and SAST tools continuously throughout all phases of the software supply chain.
- Actively manage dependencies in sources.
- Reuse existing, security-tested sources whenever possible.
- Require signed commits for all source repository submissions.
- Scan source repositories to ensure that secrets are not committed.

Best practices that apply to dependencies from open source organizations

- Thoroughly research and investigate open source artifacts before using them.
- Download and use open source artifacts from trusted sources.
- Verify the provenance of all dependencies before building them.
- Obtain and thoroughly review SBOMs for all open source artifacts.
- Automatically scan all open source dependencies with SAST tools.
- Use SCA tools to detect transitive dependencies in open source artifacts.
- Keep open source artifacts up to date with tools that automatically monitor dependencies in your source repositories for new releases.

Best practices that apply to dependencies from third-party software producers

- Establish security requirements and controls for producers that are at least as rigorous as those used internally.
- Work with producers that follow DevSecOps practices throughout their entire software development life cycle.
- Require and review SBOMs from producers for each release.
- Verify the provenance of all third-party artifacts before integrating or deploying them.
- Choose producers with product vulnerability response programs that identify, disclose, and quickly fix vulnerabilities in both their sources and artifacts.
- Ensure that producers certify their products to industry security standards and submit artifacts for external assessments.
- Require producers to verify that all third-party artifacts incorporated into their code comply with all security controls.

Best practices that apply to container images

- Use container images—including base and parent images—from trusted sources.
- Verify the provenance of all downloaded container images.
- Avoid images that are updated infrequently, especially if they do not contain relevant vulnerability disclosures.
- Opt for minimal base images with only the required operating system packages and frameworks.
- Create containers by installing the exact tools and libraries your application requires rather than removing components from an existing image.
- Scan images for vulnerabilities early, often, and in multiple places, including developer workstations, CI/CD platforms, image registries, and actively running containers.
- Keep containers up to date by rebuilding, testing, and redeploying with the latest software patches.
- Monitor containers for newly discovered vulnerabilities throughout their entire life cycle.
- Avoid deploying containers with known vulnerabilities by removing old images from registries.
- Manage any necessary secrets carefully and grant only minimal required permissions.

Best practices that apply to build processes and platforms

- Follow SLSA guidelines for reproducible builds on hosted and isolated platforms.
- Implement build platforms using CI/CD pipelines with automated security and developer guardrails.
- Use only security-focused components—like hardened container images and artifacts and checksum-verified tools—in your build processes.
- Ensure all steps of your build process are properly configured and pay close attention to changes in tools, scripts, and configuration files.
- Scan dependencies' SBOMs for security risks and generate SBOMs for newly built artifacts as part of automated CI/CD pipelines.
- Automate provenance verification for all dependencies, along with provenance generation and attestation for newly built artifacts.
- Scan dependencies automatically for vulnerabilities using SAST tools.

Best practices that apply to infrastructure used in your software supply chain

- Set and enforce software download and installation policies for developer workstations.
- Follow least privilege principles by restricting direct access to repositories, build platforms, and deployment services, and use multifactor authentication and dedicated nonhuman accounts where possible.
- Use tools to encrypt, store, and manage secrets and credentials and enforce access controls.
- Automate infrastructure deployment using IaC and GitOps techniques.
- Monitor infrastructure for configuration drift and automatically apply updates, based on your latest IaC definitions.
- Use dedicated security tools to scan IaC files and identify vulnerabilities and misconfigurations across your entire IT environment.
- Place CI/CD pipeline infrastructure within your network perimeter and pay attention to changes in pipeline configuration.

Best practices that apply to deployment and run processes in your software supply chain

- Manage application configuration using industry best practices like analyzing the security impacts of requested privileges, allowing only the minimum required services and permissions, and defining configurations as code stored in repositories.
- Test applications and infrastructure in isolated, preproduction environments before deploying to production.
- Scan running workloads automatically for vulnerabilities and misconfigurations and use defined processes for triaging and mitigating suspected issues.
- Monitor deployed applications and container images for security updates.
- Use expected software behavior and analytics—including artificial intelligence and machine learning (AI/ML) techniques—to implement data controls and detect potentially malicious activities.
- Employ deliberate network segmentation to confine vulnerabilities to a portion of your IT environment.

Red Hat's approach to security best practices

Red Hat has developed and delivered security-focused open source software for more than 30 years. Security is a critical priority in our own software supply chain practices.

In collaboration with our security partners, we implement security throughout every phase of our software life cycles and technology stack. We develop our infrastructure, application platform, and automation technologies using a robust software supply chain that implements end-to-end security in a layered approach, from the operating system to the application, across hybrid cloud environments.

As a result of this approach, our trusted open source technologies let you:

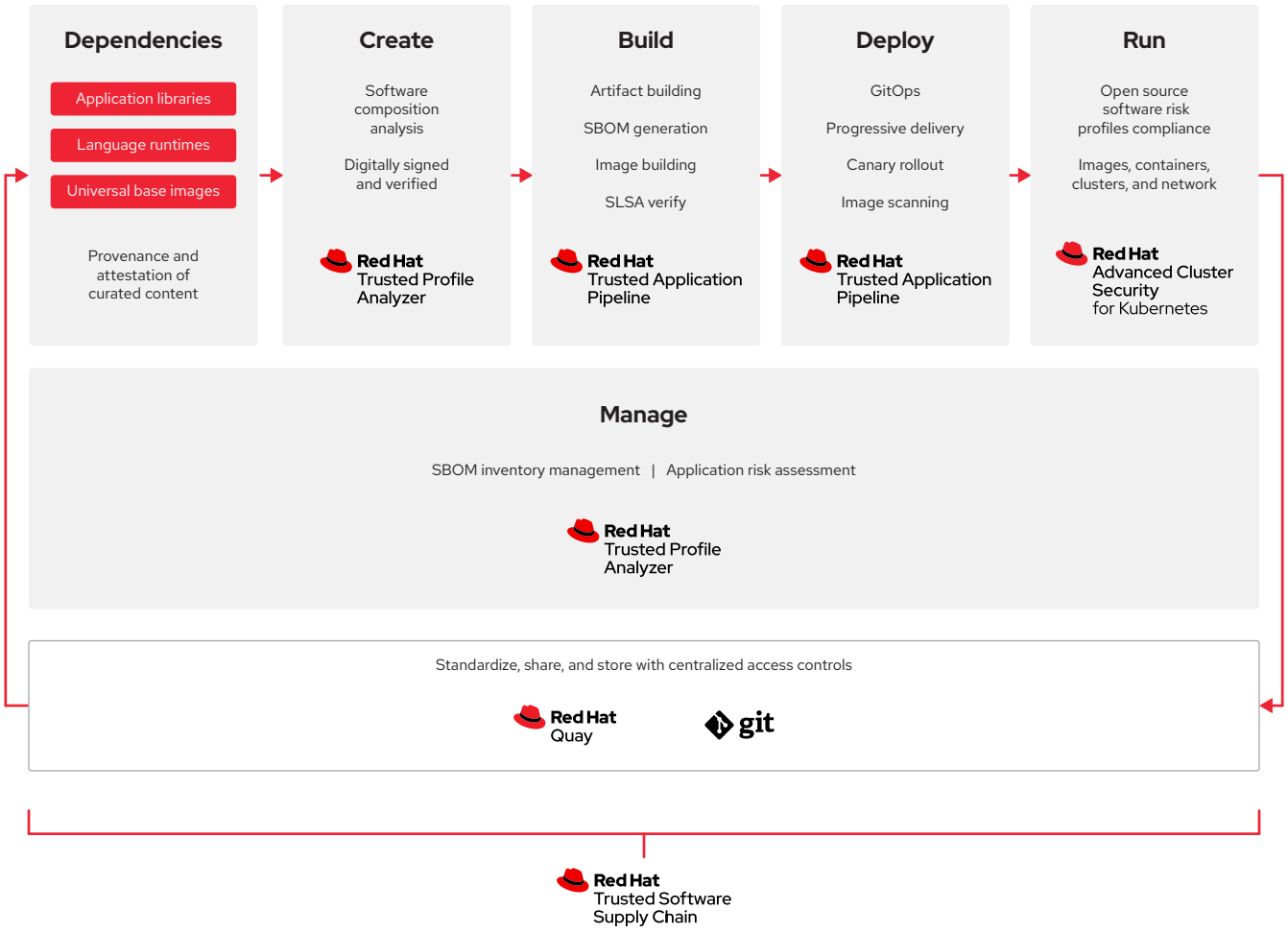
- Build a strong, security-focused IT foundation.
- Implement security throughout your application development life cycles via DevSecOps practices.
- Manage and automate your hybrid cloud environment to improve security and compliance.

Our software supply chain process includes:

- Static source code analysis.
- Software provenance.
- Quality assurance and regression testing.
- Product hardening.
- Distribution through secured channels.
- Continuous security updates for all packages contained in Red Hat® products, including backported fixes.

Boost software supply chain security with Red Hat

Red Hat's **trusted approach to software supply chain security** can help you successfully adopt DevSecOps practices, use open source code and third-party dependencies safely, and embed security into your software development life cycle. Our solutions let you code efficiently and build and monitor software using proven platforms, trusted content, and real-time security scanning and remediation—all without increasing operational complexity.



Code rapidly with integrated security checks and trusted content management

Consume content from trusted libraries that provide transparency and validate the provenance of open source and third-party source code. **Red Hat Trusted Profile Analyzer** (Service Preview) gives developers simple access to curated builds and hardened open source libraries that have been verified and attested with provenance checks. It also brings application security checks to local integrated development environments (IDEs). Developers can run dependency analyses and identify and remediate vulnerabilities while coding, helping to avoid deploying applications that contain security vulnerabilities.

Build and deploy software with security-focused workflows and policy-as-a-code

Integrate security guardrails into every phase of your software development life cycle. Red Hat OpenShift® helps you build security into containers and cloud-native applications. It incorporates many developer-friendly tools and features, including:

- **Red Hat Developer Hub** (Developer Preview)—an enterprise-grade, open platform for building developer portals.
- **Red Hat OpenShift Pipelines**—a cloud-native CI/CD solution for creating independently scalable pipelines from simple, repeatable steps.
- **Red Hat OpenShift GitOps**—an operational framework that uses Git as a single source of truth to manage and deploy application components as code.

Combining Red Hat OpenShift with DevSecOps best practices can simplify and speed application builds, testing, and deployment across on-site and cloud environments to support multicloud operational strategies. Declarative security tools let developers design, configure, and add security checks at every stage in your software supply chain.

You can build additional automation into your software supply chain using **Red Hat Trusted Application Pipeline** (Service Preview). This cloud service provides Tekton Chains pipeline definitions with automated security checks and security-focused release workflows for deploying container images across environments. It automatically generates SBOMs for container images and provides provenance and attestations in line with SLSA standards. Detailed RBACs are included, and deployment pipelines can be configured according to enterprise contracts to prevent suspicious build activity.

Extend security throughout your entire software supply chain

Continuously monitor applications and environments at runtime to identify risk profile changes caused by malicious components. **Red Hat Advanced Cluster Security for Kubernetes** provides Kubernetes-native security features to enhance infrastructure and workload protection and visibility throughout your entire application life cycle. It includes runtime detection and response capabilities.

Streamline security compliance by centralizing policy enforcement across clusters. **Red Hat Advanced Cluster Management for Kubernetes** delivers visibility into your entire Kubernetes domain with built-in governance and application life cycle management capabilities. You can immediately visualize your security posture based on defined standards and detailed application and cluster configuration audits.

Both solutions are included with **Red Hat OpenShift Platform Plus**—a complete set of powerful, optimized tools for hardening, protecting, and managing your applications.

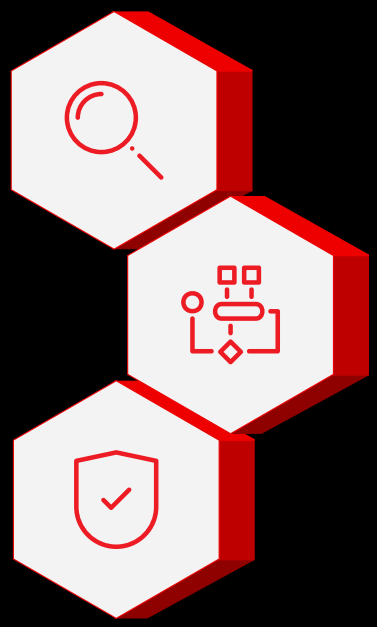
Verify and manage trusted content for your software assets

Provide secure locations for storing, managing, and finding trusted content, images, and artifacts like security metadata. Included with Red Hat OpenShift Platform Plus, **Red Hat Quay** is a security-focused, scalable private registry platform for delivering applications and container images across hybrid cloud environments. It automatically scans image containers for vulnerabilities before deploying to production to prevent last-minute introduction of threats.

Manage, analyze, and monitor SBOMs and VEX documentation with a centralized interface. Red Hat Trusted Profile Analyzer gives you visibility and control over software assets to identify security threats and vulnerabilities early in the development life cycle. It provides a unified interface for aggregating, managing, and analyzing security metadata and open source dependencies without delaying development or increasing operational complexity.

Build a trusted software supply chain with Red Hat

By following the best practices and recommendations in this e-book, you can begin your transition to a trusted software supply chain model. Red Hat OpenShift Platform Plus brings together a complete set of tested and trusted services—including Red Hat OpenShift, Red Hat Advanced Cluster Management, Red Hat Advanced Cluster Security, and Red Hat Quay—for building and deploying applications at scale. It lets you protect all artifacts, processes, tools, and infrastructure involved in your software life cycle. With this consistent foundation, you can incorporate multicluster security, compliance, and application and data management across teams and infrastructure to build a trusted software supply chain.



Ready to get started?

Protect your organization and applications from growing security threats.

Red Hat provides solutions to help you build a trusted software supply chain. Take advantage of our experience and expertise to code, build, and monitor your software using proven platforms, trusted content, and real-time security scanning and remediation.

Red Hat
OpenShift
Platform Plus

Learn how Red Hat OpenShift Platform Plus can help you build, deploy, and run security-focused applications at scale.

Learn more

Red Hat
Trusted Software
Supply Chain

See how Red Hat Trusted Software Supply Chain can help you build security into your software development life cycles.

Learn more

Red Hat
Advanced Cluster Security
for Kubernetes

Try container and Kubernetes security via a no-cost trial of Red Hat Advanced Cluster Security Cloud Service.

Try it

528000_1023_KVM

Red Hat