Safe Software Deployment: How Software Manufacturers Can Ensure Reliability for Customers

Publication: October 2024

Cybersecurity and Infrastructure Security Agency
Federal Bureau of Investigation
Australian Signals Directorate's Australian Cyber Security Centre

Table of Contents

Introduction

Many software manufacturers and service providers deploy software and configuration updates as part of their service offerings. These updates may enhance features and/or address security vulnerabilities to provide benefits and security to customers. However, software and the systems that deploy software are highly complex and continually evolving, making it challenging to deploy secure updates.

It is critical for all software manufacturers to implement a safe software deployment program supported by verified processes, including robust testing and measurements. The program should support and enhance both the security and quality of the product and deployment environment. This guide, authored by the Cybersecurity and Infrastructure Security Agency (CISA) and the following partners (hereafter referred to as the authoring agencies), encourages software manufacturers to establish a safe software deployment program as part of their software development lifecycle (SDLC).

- U.S. Federal Bureau of Investigation
- Australian Signals Directorate's (ASD's) Australian Cyber Security Centre (ACSC)

Software manufacturers should incorporate safe deployment practices early in the SDLC. Safe deployment processes do not begin with the first push of code; they start much earlier. To maintain product quality and reliability, technology leaders should ensure that all code and configuration changes pass through a series of well-defined phases that are supported by a robust testing strategy. These phases are designed to catch any new issues and regressions, reducing the risk of flawed software from impacting customers.

This guidance is part of CISA's Secure by Design campaign. The guide is primarily intended for software or service manufacturers deploying software to many types of customer systems, including mobile devices and laptops, as well as for cloud-based services (which may consist of thousands of physical and virtual systems organized into clusters). While the guide is not specifically focused on internal IT teams deploying to internal systems, many of the same phases will apply—albeit in a modified form. Other domains may require phases that are balanced differently. For example, organizations will have to contemplate complexities in some deployment scenarios, such as OSS software deployments, and the tradeoffs between automatic versus manual updates. This guide will not cover all scenarios but can be a useful tool for organizations looking to mature their deployment processes.

Objectives

A safe software deployment process aims to achieve several key objectives, contributing to the success of secure software deployment efforts. These objectives help ensure that software is not only reliable, safe, and secure for customers, but is deployed in a controlled, efficient manner. By incorporating mechanisms for timely issue detection, continuous improvement, and support for agile development, the safe software deployment process empowers teams to deliver high-quality software while adapting to changing requirements and technologies.

1. **Quality processes.** Robust quality assurance processes help ensure that software products function consistently and meet customer expectations without causing disruptions. This focus on reliability enhances customer safety by reducing the risk of failures that could compromise critical systems or data, resulting in increased customer trust.
2. **Cost and impact management.** For software manufacturers, technical and process defects cost less to remediate when such defects are caught early in the process and cause less damage to customer systems. Customers impacted by these defects can suffer downtime and data loss, adversely affecting both their financial and reputational positions.
3. **Controlled and measured deployments.** A well-defined deployment strategy includes phased rollouts, such as canary releases, allowing manufacturers to monitor performance in real-world environments without impacting all customers at once. This reduces the risk of widespread failures and provides valuable data to refine subsequent deployments.
4. **Comprehensive testing**. Early detection through comprehensive testing strategies, including automated testing and monitoring tools, allows teams to identify and address defects before they escalate into larger problems. Proactively catching issues early improves the quality of a product and reduces the likelihood of expensive fixes later.
5. **Continuous improvement.** Feedback loops integrated into the development and deployment processes enable teams to learn from each iteration and apply those lessons to future releases. Using feedback loops helps evolve the safe software deployment process and improves maintenance practices of security standards, as well as improves performance, and better meets the needs of users. It is important to include “near misses” in the feedback loop. “Near misses” provide an opportunity to enhance the program without the software manufacturer or their customers experiencing the full negative impact of an actual incident.
6. **Optimize for agility.** Teams should be able to adapt quickly to changes in requirements, emerging technologies, and security threats. Shorter development cycles, collaboration, and iterative improvements help deliver high-quality software that meets evolving customer expectations.
7. **Secure development ecosystem**. A secure development ecosystem is designed to support developers while reducing the likelihood of defects entering the code base. Software manufacturers have a greater chance of detecting and blocking issues earlier by implementing automated technical controls throughout the development ecosystem.

Key Phases of a Safe Software Deployment Process

Successful safe software deployments are based on an established secure software framework such as NIST Secure Software Development Framework (SSDF). Moreover, safe deployments hinge on a structured, well thought out process incorporated into the SDLC that minimizes risk and promotes reliability. Each phase of a safe software deployment process plays a critical role in guiding software from initial planning to release. Strongly recommended practices for safely deploying software are rigorous testing during the planning phase, controlled deployments, and continuous feedback. By following these key phases, software manufacturers can enhance product quality, reduce deployment risks, and provide a better experience for their customers.

The subsections below include common phases found in successful safe software deployment programs:

Planning

Before writing any code, a clear plan should be established to define goals and build requirements, understand customer needs, scope potential threats, and specify success criteria. The planning phase sets the foundation for the entire deployment process, helping ensure teams are aware of the scope, potential risks, and costs before moving into development.

Key Considerations

Operational Risk Assessment

Implementing a comprehensive, ongoing assessment of the risks and consequences of deploying software in an environment reduces risk. An operational risk assessment includes understanding system relationships and interdependencies, potential threats, safety, security, reliability, performance requirements, and risks associated with common defects. For example, if a service relies on another product's service account (SA), and the permissions of that SA change, parts of the system may fail in seemingly unexpected ways. It is essential to include the processes and technologies of a safe software deployment program in the overall written product threat model.

Conduct a Failure Anticipation Review

Sometimes referred to as a "pre-mortem," this review helps teams identify potential failures, drawing on insights from previous retrospectives (or post-mortems).

Platform Scale and Diversity

Deployment teams should extensively document the multiple potential roll-out scenarios associated with their products. Teams should plan for increasing platform and device diversity in cases where the software operates across multiple environments. Consider more than just device count when contemplating device diversity. Diversity can include different types of operating systems, hardware brands, firmware versions, firmware settings, bandwidth speeds and reliability, environmental settings, and geography. Teams should verify sufficient testing coverage before increasing their internal deployment rate.

Online Service Diversity

Online service diversity may include data center regions and network configurations and include failover and backup solutions as necessary.

Planning Deployment Cadence

Whether the organization sets a monthly "Patch Tuesday" or delivers multiple configuration updates per day, the team should formalize this plan and communicate it clearly to internal stakeholders and external customers.

Monitoring and Reporting Strategy

Manufacturers should identify the signals that provide the clearest view of system health and report identified issues to feed back into the continuous improvement process.

Staffing

Complex deployments require a team capable of monitoring both deployments and the refinement of the safe software deployment process. A shortage of either operations staff or software/process developers increases the risk of unexpected incidents.

Fault Tolerance

Systems can be designed to be resilient and/or fail safely even when presented with bad or malicious code or configuration changes. Each organization should consider ways to build resilience into the planning process.

Deprecation and End of Life

Software manufacturers should plan for feature deprecation and product end of life. Early warning will reduce the impact to customers and reduce the likelihood of them facing adverse effects.

Patching

Software manufacturers should plan for patching the products and services they develop. They should make the process of applying patches as seamless as possible. Security patches may require subsequent updates to fix identified vulnerabilities that could be exploited by malicious actors. Malicious actors may reverse engineer patches to identify vulnerabilities that may not yet be publicly disclosed. By leveraging information from newly released patch advisories, malicious actors can exploit customer systems that lack the most recent security patches. (see the N-1 Releases section below).

Development and Testing

This phase involves coding and continuous testing. Testing at this phase typically includes unit, integration, and automated (including static and dynamic) tests to catch issues early. Code should be tested in an environment that closely mirrors typical customer environments to help ensure accuracy and reliability. Organizations should consider devoting resources to actively trying to cause the deployment process to fail under controlled circumstances to preempt a failure in the field.

Internal Rollout (Dogfood)

When appropriate, based on the type of software and internal enterprise needs, internal teams should be the first to use new software versions. This “dogfooding” phase allows organizations to catch issues before the software reaches external users. By testing the product in real-world scenarios, internal users provide valuable feedback that helps improve the product’s stability and performance. The number of devices and amount of time needed to obtain sufficient coverage will vary; organizations should establish standards and adjust them with input from previous release cycles. Additionally, organizations should establish a culture of encouraging staff to report potential problems, even when the problems seem negligible.

Deployment and Canary Testing

The deployment to customers should be completed in a controlled way. Deployment options can include canary deployments (small-scale deployments to a limited number of customers or systems), allowing teams to monitor performance and resolve issues before a wider rollout. For software-as-a-service (SaaS) products, this may involve deploying the update to a small portion of servers or directing limited traffic at updated components. For on-premises products, the update may first be released to only a subset of customers. Organizations may wish to consider variations, such as “blue/green” deployment models or splitting customers into “Stable” and “Early Access” groups, allowing them to match access to new features with their risk tolerance.

Controlled Rollout

After verifying successful canary deployments, the deployment team can release the new version to more users. As confidence in the new version grows, the deployment can gradually expand to more devices or systems. This controlled rollout prevents sudden, widespread failures. Organizations will want to consider the appropriate velocity of a deployment for urgent security fixes compared to more routine content deployments. Organizations will need to factor in both their risk tolerance with expectations and the risk appetites of customers. During a rollout, teams may need an automatic breaker or the equivalent of an emergency stop button to halt the rollout, especially during an incident.

Feedback Into Planning

Continuous feedback is critical throughout the entire process, but particularly after release. Insights from customers, development and quality teams, system logs, examples of unexpected or abnormal system behavior, and performance metrics should feed directly back into the planning phase of the next development cycle. This feedback will enable continuous improvement and a faster response to issues.

Figure 1 shows these phases over time. Each software and service provider will tune the slope of the curve to their risk tolerance and other business considerations. Providers should also factor in their customer risk tolerances.

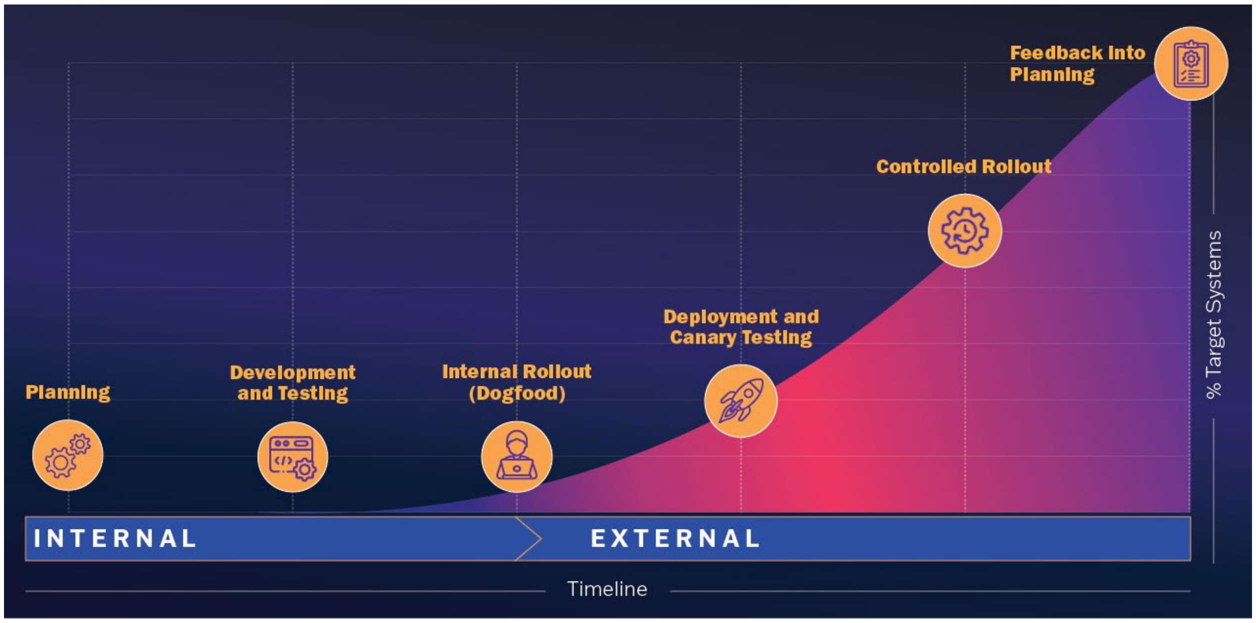

Figure 1: Deployment Timeline

Playbooks

Playbooks serve as essential tools for ensuring that safe software deployment processes are well-documented, repeatable, and resilient. They provide clear guidelines, best practices, and contingency plans to help teams navigate each phase of deployment. Playbooks also help software manufacturers reduce risks and respond effectively to any issues that arise, ultimately safeguarding both the software and the customers who rely on it. Automation, where possible, can help align the above factors to the deployment objectives.

Like all playbooks, deployment teams should have regular training and simulated incident testing. The time to learn how to react is not when the system is failing and potentially causing customer issues. It is essential to determine how people, processes, and technologies work together before an incident.

It is important to recognize that deploying software to customer systems involves business considerations, not just technical considerations. Senior business leaders should ensure teams curate the playbook over time, request changes, and formally approve playbooks at least annually. The joint guide, Shifting the Balance of Cybersecurity Risk emphasizes the principle of “leading from the top.” Having a business leader take ownership of the process will help ensure the organization allocates the necessary resources to achieve successful outcomes.

Like all other elements of an SDLC, playbooks require regular maintenance to monitor and improve their level of maturity and ability to achieve the organization’s mission.

Errors

At any phase of the deployment process, the product or support system may encounter errors. These errors may be the result of latent coding errors in the software or deployment system that escaped testing measures. They can include performance issues, compatibility issues, or even security issues.

It is important that the technology leadership team establish written playbooks to guide staff when they encounter these types of errors. These playbooks should include thresholds for acceptable error rates, staff response, and command chain escalation.

Organizations that deploy real-time monitoring systems and automated alerting and escalation systems can react more quickly to anomalies and errors than those that rely solely on human operators. Earlier detection of issues and automated remediation reduce the time between identification and resolution, leading to fewer impacted customers and minimized downtime.

Emergency Protocols

The playbook should include detailed steps for handling emergencies during or after software deployments, including detailed recovery tasks. At a minimum, the following topics should be covered.

Incident Detection and Reporting

Clearly define how issues are identified, whether through automated monitoring systems, internal testing and/or customer reports. Social media may also be a source of important information that may be missing from other, more formal channels.

Engineering and Management Escalation Processes

Outline a structured escalation path that indicates when and how incidents are escalated to senior engineers, management, or even external partners.

Recovery and Rollback Procedures

Document a detailed recovery plan that can be executed quickly if a deployment causes significant failures. This should include the steps needed to revert systems to a previous stable state, validation checks to ensure the rollback is effective, and safeguards to prevent data loss. Understanding how to bring customers back to the last known “good” state will require careful planning, especially if the target systems are highly diverse. One common strategy is to enhance systems to degrade gracefully by, for example, disabling certain configurations, rather than stopping a deployment entirely.

Note: Recovery planning scenarios where automated rollback cannot be implemented require additional care.

Root Cause Analysis and Reporting

After resolving the immediate issue, conduct a thorough root cause analysis (RCA) as part of a blameless retrospective to identify what went wrong and why. RCA should be followed by documentation of the findings and corrective actions that can prevent similar incidents from occurring in future deployments. “Near misses” can provide significant value in this step.

Customer and Partner Notification Plan

Ensure there is a plan in place for notifying customers and partners in the event of a critical issue. This includes determining the appropriate communication channels (email, social media, in-app notifications) and message timing and providing clear information on the issue, its impact, and expected resolution time. Revalidating customer and partner contact information can prevent costly delays in communication. See below for more detail.

Note: The safe software deployment process should be referenced in the larger incident response plan (IRP). Integrating the deployment process into the incident response plan allows for a seamless transition from deployment management to incident handling. It also helps ensure that all relevant stakeholders—developers, operations teams, security personnel, external communications, and customer support—are aligned and equipped to minimize downtime, address security vulnerabilities, and maintain business continuity.

Customer Notification Plan

Even with a safe software deployment process, incidents may still occur. To maintain transparency and trust, software manufacturers should have structured plans for notifying customers. The following elements should be considered in the notification plan:

1. **Pre-deployment notifications**: Before any major update, notify customers about upcoming deployments, including the expected timeline, potential impact, and any planned downtime. **Note:** This element needs to be calibrated to avoid alert fatigue.
2. **Support customer controlled deployment**: Allow customers control over the deployment schedule and how they receive updates.
3. **Rollout status update**: During the deployment process, provide real-time or frequent updates on the rollout status via an appropriate channel, such as a well-known and continuously updated public web portal.
4. **Incident and outage reporting**: If an issue arises during deployment, quickly inform customers about the nature of the incident, its impact on their systems, and the steps being taken to resolve it.
5. **Post-deployment notification**: Once the deployment is complete, send a follow-up communication to confirm the successful rollout. Include details of any new features, bug fixes, or changes and provide a channel for customers to report any post-deployment issues they may encounter.

Additional Consideration: N-1 Releases

Some customers prefer to stay on older versions of software or configurations—often referred to as N-1 (the previous release) or N-2 (two releases back)—to avoid potential risks associated with new updates. These risks may include bugs, compatibility issues, or disruptions that could affect their systems or operations.

However, while staying on older versions may seem safer, delaying updates can introduce unmanaged risks, particularly when updates include critical security enhancements or vulnerability patches. Software manufacturers should focus on improving their deployment practices and demonstrating their reliability to customers. Rather than slowing down deployments, software manufacturing leaders should prioritize enhancing deployment processes to ensure both security and stability.

Conclusion

Safety and security incidents often result from multiple contributing factors, including people, processes, and technology elements working together in a system that may become misaligned (sometimes in unexpected and unlikely ways). A safe software deployment process should be integrated with the organization's SDLC, quality program, risk tolerance, and understanding of the customer's environment and operations. By adopting a systems-thinking approach, teams may reduce the likelihood of their deployment process operating outside the safety boundary.

Two approaches can help teams maintain this safety boundary. First, foster a blameless retrospective (also called "postmortem") culture, where teams analyze both positive and negative outcomes by focusing on the processes that contributed to the result, rather than assigning blame to any individual. Individual actions should not lead to an incident if the environment and processes are resilient.

Second, treat "near misses" as real incidents. "Near misses" provide significant information to improve processes. They offer an opportunity to evolve programs without suffering the consequences of an actual incident. Analyzing "near misses" allows teams to uncover weak points in the system and address them proactively, reducing the likelihood of future failures. By studying these events, organizations can build a culture of continuous improvement that strengthens security and operational resilience.

Organizations should formally evaluate their software deployment processes based on the points outlined herein and develop a plan to address them through a continuous improvement program. A well-designed software deployment process can ensure that customers receive new features, security, and reliability in a timely manner, while minimizing unplanned outages.

Disclaimer

Acknowledgements

This information includes input from the following Joint Cyber Defense Collaborative (JCDC) participants: Microsoft, CrowdStrike, and Google contributed to this guidance.

Additional Resources

- National Institute of Standards and Technology (NIST) Special Publication (SP) 800-218: Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities.
- Australian Signals Directorate – Australian Cyber Security Centre's Secure-by-Design Foundations