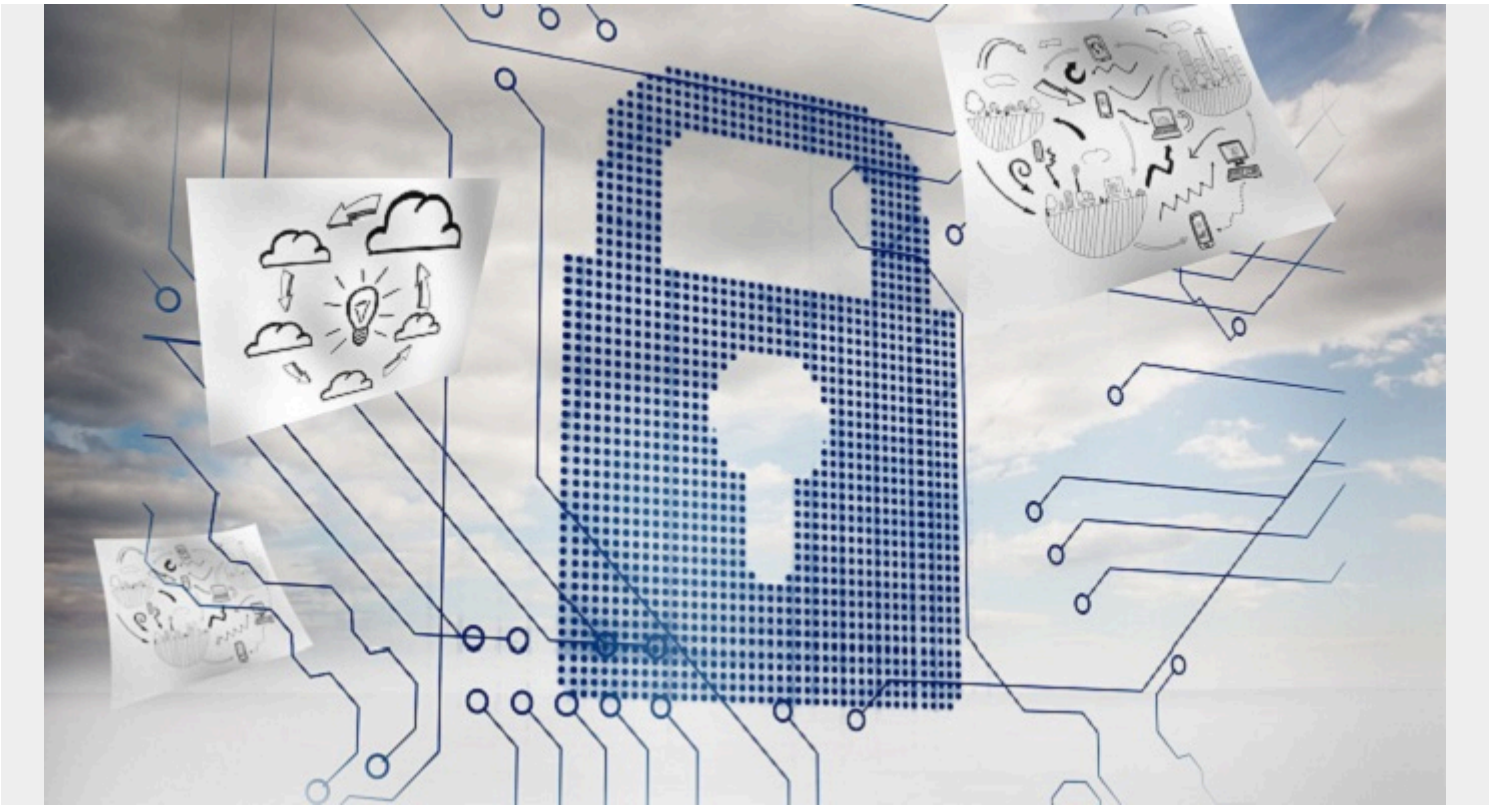
FIVE BEST PRACTICES FOR BUILDING SECURITY AS CODE INTO A CONTINUOUS DELIVERY PIPELINE



This is the fourth blog in our mini-series illustrating how BMC used agile development, cloud services, an Infrastructure as Code approach, and new deployment technology to deliver a cloud native product.

Be sure to also read: Getting Started with Cloud Native Applications, 9 Steps for Building Pipelines for Continuous Delivery and Deployment, and Infrastructure and How "Everything as Code" changes everything

Building security as code into a CI/CD pipeline means treating security policies, tests, and compliance checks as versioned, automated artifacts that run at every stage of delivery—catching vulnerabilities during development rather than at the point of release. The five best practices below show how engineering teams can embed Security-As-Code and Compliance-As-Code principles directly into a continuous delivery pipeline to maintain both security and speed.

Why does security as code matter in a CI/CD pipeline?

When security is not integrated from the start, it becomes a bottleneck at the worst possible moment. As a release approaches deployment readiness, teams are forced into a late-stage gauntlet of vulnerability scanning, penetration testing, and threat modeling—executed through manual tools, checklists, and review documents. When critical or high vulnerabilities surface at this stage, releases stall for weeks or months while issues are resolved.

The Security-As-Code principle eliminates this bottleneck by codifying security practices and automating their execution as part of a DevOps pipeline. Security and compliance are enforced continuously—at the use case, design, development, and testing stages of application delivery—rather than deferred to a pre-release review. Security policies, tests, and results are treated as first-class code artifacts: versioned in the same repository and progressed through the same CI/CD pipeline as application code.

What are the five best practices for applying security and compliance as code?

The following practices are drawn from BMC's experience building a cloud native application on Amazon Web Services (AWS). Each addresses a distinct layer of the Security-As-Code approach—from initial policy definition through production remediation.

1. Define and codify security policies

Security policies should be defined at the very start of a project and stored in a source code repository alongside application code. For a cloud native application on AWS, this includes requirements such as:

- Security groups and firewalls secured
- Virtual Private Cloud (VPC), Elastic Load Balancing, and related services enforced and secured
- All EC2 and other resources must reside within the VPC and be individually firewalled
- All AWS resources and logs encrypted using Key Management Services

Security policies must also be automated—a single action should be able to evaluate them for any application at any stage or environment. Everything about security is codified, versioned, and automated; security artifacts are checked into a repository just like application code.

Enterprises should build standardized security patterns to enable reuse across organizations and applications. A 3-tier application and a dev/test cloud environment, for example, each require their own standard security template. With AWS, the five NIST AWS CloudFormation templates can harden networks, VPCs, and identity and access management roles—forming a "hardened cloud" foundation. For private cloud or on-premises infrastructure, BMC BladeLogic Server Automation handles server hardening.

2. Define security user stories and assess application architecture risk

Security-related user stories must be defined in the agile process just like any other feature story. Examples include:

- "Input validation for cross-site scripting and SQL injection"
- "SSL/TLS enabled for all communication"
- "Automated application security testing"

This practice ensures security is never treated as optional or deferred. Security controls are identified based on application security risk and threat models for both the application and infrastructure architecture, then implemented in application code or through policies.

3. Test and remediate security at application code check-in

Security testing should be automatically triggered as soon as code is checked in—for both application and infrastructure changes. Many security vulnerability scanning tools are available from BMC, third-party vendors, and open-source projects to automate detection and remediation. When critical or high vulnerabilities are found, automation tools can generate defects or tickets assigned to the component owner for resolution.

For cloud native applications, testing at code check-in involves evaluating infrastructure-as-code artifacts—such as AWS CloudFormation templates—against defined security policies.

4. Test security policies to ensure they don't violate requirements

Security policies themselves must be tested to ensure they don't permit known-bad configurations. Examples of policy violation checks include:

- "No S3 buckets should be publicly readable or writable"
- "Do not use open security groups with 0.0.0.0/all traffic allowed"

This testing should be automated as part of the application delivery pipeline. In BMC's pipeline for its cloud native application, security scans are orchestrated from Spinnaker to validate every push to production. For regulatory compliance checks—such as Center for Internet Security (CIS) and Defense Information Systems Agency (DISA) requirements across servers, databases, and networks—the BMC BladeLogic product suite can detect and remediate compliance violations.

5. Test and remediate security and compliance in production

Automated security and compliance testing continues as an application moves through development, QA, and test environments. The risk of finding critical issues at this stage is lower because most security automation runs much earlier in the pipeline. For production environments, BMC BladeLogic Server Automation and BMC Threat Director support security patching after scanning production servers for mutable infrastructure.

This step remains critical: more than 80% of attacks target known vulnerabilities, and 79% of vulnerabilities have patches available on the day of disclosure. For applications deployed in public clouds such as AWS, security and compliance evaluations can be automated through Spinnaker.

In modern cloud-native stacks, components are never patched in place—they are replaced entirely. Even so, new containers or servers must pass security and compliance testing before being deployed to production.

Shift left: start security at the beginning

The shift-left principle means moving security from the end of the release process to the beginning—and keeping it there throughout the DevOps pipeline. Rather than treating security as a final gate or running periodic scans of production environments, teams embed Security-As-Code practices at every stage of delivery, from policy definition at project kickoff through automated compliance checks in production. By codifying policies, automating tests, and integrating compliance into the CI/CD pipeline from day one, product teams can maintain both security and the speed of innovation.

Frequently asked questions about security as code in CI/CD

What is the difference between security as code and DevSecOps?

Security as code is a specific practice within DevSecOps that involves treating security policies, tests, and compliance checks as versioned, automated code artifacts stored in a source repository. DevSecOps is the broader cultural and organizational practice of integrating security responsibilities across the entire software development lifecycle—security as code is the technical implementation that makes DevSecOps actionable.

What does "shift left" mean in the context of security testing?

Shifting left means moving security testing and validation earlier in the development process—ideally to the point of code check-in—rather than waiting until staging or release. In a Security-As-Code pipeline, automated security checks run at every commit, catching vulnerabilities at the development stage where they are cheapest and fastest to fix.

What tools support automated security testing in a CI/CD pipeline?

Common tools include vulnerability scanners, infrastructure-as-code policy validators, and compliance automation platforms. BMC BladeLogic Server Automation, BMC Threat Director, and pipeline orchestrators such as Spinnaker are examples of tools that support automated security and compliance testing across development, QA, and production environments.

Why are standardized security patterns important for cloud applications?

Standardized security patterns allow teams to reuse proven security configurations—such as hardened VPC templates or encrypted storage policies—across multiple applications and environments. This eliminates the need for each team to define security from scratch and ensures consistent, out-of-the-box security coverage across the organization. NIST AWS CloudFormation templates are one example of a reusable hardened cloud pattern.

How does Security-As-Code reduce release delays caused by security reviews?

By automating security and compliance checks at code check-in and throughout the CI/CD pipeline, Security-As-Code catches vulnerabilities during development rather than at the point of release. This eliminates the last-minute manual security assessments—penetration testing, vulnerability scanning, threat modeling—that traditionally halt deployments for weeks or months when critical issues are found close to production.

The views and opinions expressed in this post are those of the author and do not necessarily reflect the official position of BMC.