Building your DevSecOps pipeline: 5 essential activities

No matter what you call it, SecDevOps, DevSecOps, or DevOpsSec, you have to build security into your continuous integration, continuous delivery, and [continuous deployment](https://www.synopsys.com/glossary/what-is-continuous-deployment.html) pipeline. This checklist will guide you through the DevSecOps journey—as we’ll call it within this checklist—to assure that you’re integrating security into your pipeline.

Here, we’re going to look at each of the five activities listed in the overview above. We’ll examine their purpose, benefits, key enablers, and use cases. Let’s dig in.

1. Pre-commit checks

Pre-commit checks, the first step in the DevSecOps pipeline, consist of steps to complete before the developer checks code into the source code repository.

**Purpose.** Pre-commit checks are used to find and fix common security issues before changes are committed into source code repositories.

**Benefits.** The use of pre-commit hooks is very powerful. They can help a team automate manual tasks and increase their productivity. Additionally, security checks using [static analysis tools](https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html) in the IDE can take place with a limited number of rules.

**Key enablers**. Pre-commit checks enable activities such as updating a threat model when controls or new assets are added to the application. They also enable manual code review when a large change in the code base is detected. These checks can additionally trigger risk analysis when identifying security vulnerabilities.

**Use case.** These checks enable development teams to run scans in their IDE using [Code Sight](https://www.synopsys.com/software-integrity/code-sight.html). This tool automatically provides ‘just in time’ security guidance as the code is written. Rather than scanning for bugs after the code is written and committed to your source code repositories, Code Sight acts as a desktop security expert. It provides guidance automatically when developers create code where risk may be introduced.

Next, create hooks to trigger activities such as [threat modeling](https://www.synopsys.com/software-integrity/software-security-services/software-architecture-design/threat-modeling.html), [architecture risk analysis](https://www.synopsys.com/software-integrity/software-security-services/software-architecture-design/risk-analysis.html), and manual code review. Create additional hooks to review your configuration files for hard-coded credentials.

Finally, use these hooks to distribute email notifications to your application security team or software security group (SSG). Notify them about critical code changes that developers have checked into source code repositories.

2. Commit-time checks

The next step in the DevSecOps pipeline is commit-time checks. This activity is automatically triggered by a check-in to a source code repository.

**Purpose.** To build and perform basic automated testing of the application. These tests return fast results to developers who committed the change to the source code repository.

**Benefits.** Commit-time checks ensure that code is compilable and buildable at all times. They also bring attention to critical and high security issues.

**Key enablers.** These checks identify well-defined processes for various [software security activities](https://www.synopsys.com/blogs/software-security/bsimm-software-security-activities.html). They also empower development teams to remediate critical and high risk issues. Additionally, they empower QA security testing.

**Use case.** First, compile and build the code. Next, configure and run static analysis with limited rule sets. One recommendation is to run your firm’s top 3 vulnerabilities (identified annually). For instance, vulnerabilities such as [SQL injection](https://www.synopsys.com/glossary/what-is-sql-injection.html) and/or [reflected and stored cross-site scripting](https://www.synopsys.com/glossary/what-is-cross-site-scripting.html) (XSS). Use static application security testing (SAST) tools like Coverity to identify security issues. This is a fast, incremental scan that provides feedback to developers in minutes.

Next, automate security testing and gather metrics. Break the build and alert relevant teams on critical and high security issues.

3. Build-time checks

Build-time checks, the third activity in the DevSecOps pipeline, are automatically triggered by successful commit-time checks.

**Purpose.** To perform advanced automated testing of the application. This includes a deeper level of SAST, open source management, security testing, risk-based security tests, signing binary releases with PGP signatures, and storing artifacts in repositories.

**Benefits.** Build-time checks break the build in any failure, including:

* When code doesn’t compile
* In the event that unit tests fail
* SAST failures
* A high number of findings
* When vulnerabilities are found (e.g., SQL injection or XSS)

These checks also identify dependencies and checks if there are any known, publicly disclosed vulnerabilities using tools (e.g., SCA).

**Key enablers.** These checks empower QA security testing with a well-defined process for various software security activities. They also empower development teams to remediate critical and high risk issues as they’re introduced.

**Use case.**Build-time checks allow users to configure more comprehensive SAST rule sets, such as the OWASP Top 10 when dealing with web applications. They also configure jobs to identify risks in third-party components, using tools such as [Black Duck](https://www.synopsys.com/software-integrity/security-testing/software-composition-analysis.html). These checks automate risk-based security testing. Risk-based security testing runs specific security tests based on the risk profile of the system.

Each test is intended to probe a specific risk that has been previously identified through risk analysis. They alert development teams of critical and high risk issues. They even digitally sign artifacts and store them in your artifact repositories. Last, but not least, build-time checks gather useful metrics.

4. Test-time checks

Moving down the DevSecOps pipeline, test-time checks are automatically triggered by successful build-time checks.

**Purpose.** Pick the latest ‘good’ build from the artifact repository and deploy it to a staging or test environment. All tests, including functional, integration, performance, advanced SAST, and DAST are executed on this build.

**Benefits.** This is the last testing phase before a product is released into production. The staging environment is the most representative of the production environment.

**Key enablers.** Test-time checks require well-defined processes for various software security activities. They empower development teams to remediate critical and high risk issues as soon as they’re introduced. They additionally empower QA security team testing methods. In addition, they trigger manual code review using SAST tools and out-of-band penetration testing.

**Use case.** Configuring a broader set of rules for SAST, in this case, might include using the tool’s full security rule sets. Since you already ran SAST in the earlier checks, ensure that you run tests that haven’t yet been covered. Configure to run DAST tools. The rule sets should test for common critical and high severity issues such as those outlined in the OWASP Top 10.

Include [fuzz testing tools such as Defensics](https://www.synopsys.com/software-integrity/security-testing/fuzz-testing.html). Fuzz testing provides random data to the program’s input parameters in the hopes of causing an error state. Failing to handle malformed input properly can lead to security issues.

Configure and automate the deployment of the latest ‘good’ build to the staging environment. Then, alert the development teams of the critical and high risk issues. And finally, gather metrics from these activities.

5. Deploy-time checks

If all of the previous steps have been completed successfully, and the application is ready for deployment, deploy-time checks involving additional pre- and post-deployment security checks finish out the DevSecOps pipeline.

**Purpose.** Testing post-deployment provides an ongoing level of assurance that changes to the production environment haven’t introduced security issues. A good strategy is to implement a process that periodically triggers security testing.

**Benefits.** Deploy-time checks can help find bugs that may have slipped through pre-production testing activities. Continuous monitoring allows an organization to gain insight into the types of traffic a given application is receiving. Additionally, collecting application-level security metrics helps identify patterns of malicious users.

A threat intelligence program can also help teams stay ahead of the curve by proactively responding to newly discovered security issues that affect applications and platforms.

**Key enablers.** Defects identified through this activity can be fed back to development teams and used to change developer behavior.

**Use case.**

Pre-deployment

* Automate configuration management
* Automate provisioning the runtime environment

Post-deployment

* Automate collecting application-level security metrics during continuous monitoring
* Schedule security scanning
* Perform vulnerability scanning
* Assist in bug bounty scanning
* Create an incident response plan
* Provide insight to the DevSecOps team to drive a threat intelligence program

Pre-commit phase

As a developer checks in code, the pre-commit hooks review changes to the code and configuration before committing it to the source code repository (e.g., SVN or Bitbucket). Client-side and server-side hooks are both relevant here.

Commit-time phase

Next, once the code is committed to the source code repository, run commit-time checks. This includes incremental SAST with predefined rule sets that provide quick feedback for developers (within seconds of code check-in). If you find critical or high risk issues (e.g., SQL injection or XSS), you must break the build and notify the developer immediately.

SAST should be incremental. In other words, run SAST only on the set of files that change. Additionally, be sure to gather metrics into a centralized dashboard. Automate defect tracking and create a defect. After all, security issues should be treated in the same fashion as quality issues.

Build-time phase

Once the commit-time checks are successful, the next phase in the DevSecOps pipeline are the build-time checks. Here, you’ll want to run even more comprehensive rule sets using SAST. Perhaps you’ll also want to run the OWASP Top 10 for your application. At this point, you should also [run a software composition analysis (SCA) tool](https://www.synopsys.com/software-integrity/security-testing/software-composition-analysis.html) to identify vulnerabilities within your [free and open source software (FOSS)](https://www.synopsys.com/glossary/what-is-open-source-software.html) or any associated licensing issues. As we saw with the commit-time checks, here you’ll want to break the build, automate bug tracking, and gather metrics.

Test-time phase

At this point, it’s time to move on to test-time checks. Before running security activities, ensure that you use the latest successful build artifact. If you aren’t confident of your SAST rules, run a full rule set of SAST analysis. Once your SAST tools give you a green signal, run DAST or IAST tools that you have configured. If you also own a fuzz testing tool, this is the right time to run it as well.

All vulnerabilities identified during your SAST, DAST, IAST, and fuzz testing activities should break the build, gather metrics, and immediately create a defect in your bug tracking system.

By now, you’re probably getting an idea of how your changes are progressing through the DevSecOps pipeline. You’re probably seeing what activities completed successfully, which vulnerabilities were found all in one dashboard, and whether to continue the DevSecOps pipeline or pause/stop the change from going into production.

Deploy-time phase

Now we move into the final phase: pre- and post-deployment checks. This is the time to automate configuration management. Provision the runtime environment (e.g., check access control and group permissions) if your application is using SSL to connect to the database.

Once the application goes live, schedule security scanning to identify bugs that may have slipped through pre-production testing. Implement a bug bounty program to triage and investigate issues reported by users. Enable continuous monitoring to gain insight into the types of traffic a given app receives.

Additionally, collecting application-level security metrics helps to identify patterns of malicious users. Last, but certainly not least, a threat intelligence program can help teams stay ahead of the curve. It can help teams proactively respond to newly discovered security issues affecting applications and platforms.