circleci

Vulnerability Management and DevSecOps with CI/CD

Many of the world's highest performing software development teams have adopted DevOps practices. And while DevOps has simplified software builds, testing, and deployment, it fails to address a vital issue: security and vulnerability management.

Vulnerabilities can occur at many points in the DevOps pipeline. Developers might accidentally write insecure code. An application may use a library containing a security bug. Even containerized applications can contain vulnerabilities in the operating system running inside the container.

With companies large and small continuing to invest heavily in DevOps and containerization, vulnerability management requires new approaches. Developers can't just rely on operations teams to patch servers to the latest updates. Instead, development teams must include security in their workflow.

In this ebook we will discuss how modern developers and DevOps practitioners can use CI/CD to adopt a DevSecOps approach to vulnerability management.

More power, more responsibility

Developers have many useful automation tools at their disposal, but these tools can add potential security vulnerabilities when used carelessly. Our code, packages, pipelines, and container images are all sources of value and also all sources of vulnerabilities.

Hackers and security researchers continually discover vulnerabilities in software, libraries, operating systems, and infrastructure. Vulnerability management is the ongoing process of scanning, classifying, prioritizing, and patching software vulnerabilities. A DevSecOps approach can help by automating all of these areas.

Next to vulnerabilities in software and hardware, companies may have unknown vulnerabilities in their internal processes, for example, when hiring new people and managing access to resources. Security policies should also address and minimize these vulnerabilities.

DevSecOps can help address all these issues by having clear, documented, and — most importantly — automated processes in place.

Code vulnerabilities

Developers often rely on other people's code. Most software projects start out with importing packages using a package manager. This code can be open source or proprietary, and it often relies on additional packages. Especially when writing JavaScript and using npm, it's packages all the way down.

These external dependencies are not without risk. Packages may contain incorrect or malicious code. Companies often release new versions of their software and packages that include new security updates, but those updated packages may introduce new vulnerabilities (or bugs).

That said, your software may contain its own vulnerabilities as well. It might be because you made a quick fix — we all do that from time to time — or because you overlooked something. It may simply be because your older code no longer follows today's best security practices.

CI/CD pipelines

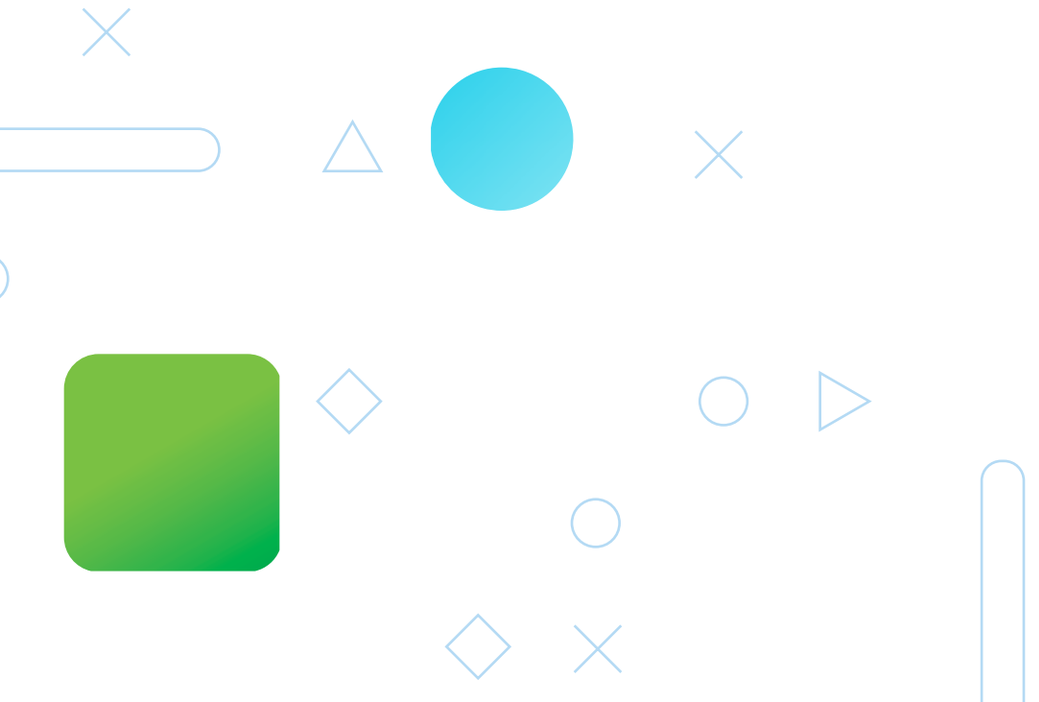
DevOps practices aim to automate as much work as possible to save time and minimize human error. Central to this automation are CI/CD pipelines. Pipelines automatically trigger builds, run tests, and deploy software, either in the cloud or on your private server.

While CI/CD pipelines are key to successfully implementing DevOps, they also come with potential security threats. Unauthorized access to the pipeline can put your company at risk. As we'll see, CI/CD pipelines can play an important role in vulnerability management — but we must first ensure they are secure.

CI/CD pipelines may contain sensitive information, like usernames and passwords, for things such as an automated database release or an application that needs to connect to a database. Security management must be implemented correctly, or your developers will have direct access to production databases that contain sensitive data.

When a developer leaves the company or team, not only would you have to disable access to the pipeline, but you'd have to refresh all your passwords as well in case they wrote them down or remember them.

Secret management can be partly automated, such as by generating secure passwords for automatically rolled out services. Make sure you always use a secret manager like AWS Parameter Store, Azure Key Vault, or HashiCorp's Vault to manage your secrets and secret access.



Containerization

Containerization with tools like Docker and Kubernetes can be a useful tool for developers. It provides more control over their application's runtime environment. Containers ensure that all the dependencies and secrets the application needs are available. Testing becomes more straightforward because the testing environment is identical to production.

But containerization also introduces a new challenge: development teams must now worry about vulnerabilities in their container images, not just in the libraries they use. Vulnerabilities may exist because images or the software they run are outdated. Or perhaps your images themselves are safe, but the configuration isn't. We need a new vulnerability management approach that keeps up with modern DevOps practices.

Also, working with containers does not mean you do not have to worry about servers. Containers still run on hardware and may run other software alongside containers. You could leave both private and cloud-hosted servers unprotected by using poor passwords or unintentionally leaving ports open.

Up and running with DevSecOps

Automated builds, tests, and deployments with CI/CD pipelines are familiar to teams implementing DevOps. But how can we incorporate vulnerability management seamlessly into the DevOps processes?

The answer is DevSecOps.

DevSecOps focuses on making security an integral part of an organization's DevOps processes. According to the **2019 State of DevOps Reports** by Puppet, CircleCI, and Splunk, teams implementing DevOps are already more secure on average than teams without these practices. However, with DevSecOps, security becomes an explicit and integral part of your DevOps workflow. The goal is to make everyone responsible for security.

DevOps has a focus on bringing teams together to encourage collaboration and shared responsibility. Before DevOps, multiple teams were often responsible for deploying software, with separate scopes of responsibility. Developers built the software, testers tested the software, and system administrators would deploy it on a server.

In some cases, the software would be tested for security vulnerabilities by yet another team, often in the final release stages. Sometimes, these teams would never even meet or talk to

each other. They were often located in other buildings or even in different countries — security as a black-box exercise.

This wasn't much of an issue when software release cycles lasted months or years, but teams practicing DevOps sometimes release software multiple times a day. Outdated security practices can undo many great DevOps practices and greatly delay software releases.

DevOps encourages the development, testing, and operation teams to work together, often in multi-disciplinary teams. This approach ensures that the team that writes an application also knows how to build, test, deploy, and monitor it.

If you're interested in learning more about bringing DevOps to your team, read our other ebooks Leading Your Team to DevOps Maturity and Software Testing for DevOps-Driven Teams

By involving the security team in DevOps (thus forming DevSecOps), security concerns are addressed at every stage across the software life cycle by the people most familiar with their application and its needs. Think of your security team as an enablement team which sets up the rest of the organization to build, test, and deploy faster and without worry. In this arrangement, each development team handles package management, secret management, and container image vulnerability management for their own application.

How can you implement DevSecOps in your organization across your development teams?

You can't have DevSecOps without the DevOps, so that's a prerequisite. Teams should be working together, and software builds and deployments must be automated. If you're not practicing DevOps yet, the road to implementing DevSecOps starts with **adopting DevOps practices**.

If you are practicing DevOps, you probably already have CI/CD pipelines in place, which is a great start. The next steps to DevSecOps may involve creating an inventory of your software assets (you can't secure what you don't know about), making sure you know who is responsible for those assets, as well as making some changes to your CI/CD pipelines to make sure they're scanning for issues.

Preparation for DevSecOps

First, determine the software, pipelines, and container images your organization is running. This includes the software your users are using as well as packages your company maintains, tools your developers use, and software that is currently in development.

Then, determine which teams are responsible for each. This undertaking can be arduous, and sometimes even contentious, but know that there are many other teams who will also want and benefit from this scoping for their own clarity of ownership. Track this in whatever way works best for your company. A spreadsheet, kanban board, and tools like Airtable can all work equally well. The result is a complete list of software, pipelines, and images you'll need to check for vulnerabilities, and what teams will be responsible for fixing them.

Get everyone on board

Before opening issues or bug reports, talk to all development teams. Ensure they understand the following:

- Why you are making these changes to adopt DevSecOps practices.
- What will be required of them.
- How it will enable them to do their jobs better.
- Nobody is being singled out. All development teams are now responsible for patching and updating their source, pipelines, and container images.

One thing to watch out for is ensuring that teams get the resources they need to make these changes. Writing software is often a struggle between wanting to write stable and maintainable code and wanting to ship new features as soon as possible.

Simply dropping vulnerability management on your developers' plate isn't going to work. Ensure your teams get the time and resources necessary to make those changes now and keep maintaining them in the future.

Adding security as part of your DevOps process is like developing any new feature: it has to be planned, developed, and tested and takes time. It's not some switch you can turn on or off. Like DevOps, DevSecOps must be a company-wide effort in order to succeed.

Use your CI/CD pipeline

If you're already working with CI/CD pipelines as part of your DevOps process, make sure these pipelines are secure.

You should look for passwords stored as plain text as part of some script or setting. This happens more often than you'd think.

Pipelines often allow for parameters to be secrets instead. You can set secrets, but secret values will not show in either the designer view or logs afterward.

If possible, use a secret manager instead. They allow you to store secrets in a single, safe place while still sharing secrets with various applications. Authorized people, like managers, administrators, and lead developers, can access the secret manager. However, others on the team cannot access the secret manager or its secrets.

Add scanning to your pipeline

Once your CI/CD pipeline is secure, consider adding scanners for various types of vulnerabilities. By adding scanners to your existing CI/CD pipeline, developers get valuable vulnerability feedback from the tools they're already using. This is a far more effective approach than emails, meetings, and reports, as those provide distractions, while the pipelines are already part of the daily job. By using scanners, you can automate repetitive and error-prone work for the security team.

You can add scanners like:

- Alcide, which scans Kubernetes clusters for vulnerabilities.
- Snyk, which enables developers to continuously find and fix vulnerabilities in open source libraries and containers.
- Stackhawk to scan code for bugs.

With these types of scanners, developers will be notified when their code contains security vulnerabilities. When a vulnerability is severe, a build fails, and the vulnerability will never make it to production.

Note that you can find CircleCI orbs — reusable packages of CircleCI configuration — for all of these scanners, enabling immediate integration into your CI/CD pipeline. We'll explain more about orbs later.

When working with containers, it's wise to add a scanner for image vulnerabilities as well. Since Docker containers are self-contained and specify their dependencies, it's easy to add container vulnerability scanning to your CI/CD pipeline. There are both paid tools and open source tools you can use. One great open-source option is Clair.

The initial rollout

Consider an introductory period where vulnerability reports are advisory, but not build-breaking. Your team can use this time to fix any security issues and get used to the new security practices. Again, make sure teams have the time to fix security issues. It will take a few story points out of your next couple of sprints.

Development teams might be unhappy about being given responsibility for vulnerability management. When you first enable scanning, use it to build a comprehensive report of all your organization's container vulnerabilities. With this report,

create a plan with each affected team to fix the vulnerabilities in a reasonable time frame while still finishing other work they need to complete.

Once you've fixed the initial vulnerabilities, make the scanner a build-breaker. If container changes introduce a vulnerability, developers must fix it before an automated CI/CD build and deploy can happen.

The automated vulnerability testing you've added to your pipeline makes it easy to test and validate vulnerability patches. This way, security management becomes part of the regular development cycle instead of something you do periodically, like once a week or a month.

Look for time-savers

Once you've fixed critical vulnerabilities, look for ways to reduce security maintenance overhead.

One time-saver may be to move common code to shared libraries. Developers often copy and paste code and multiple teams will copy and paste the same code into different parts of the codebase. There are scanners available that detect (near) duplicate code, both for pipelines and code editors.

Consider moving any code highlighted by duplication checks to a shared library that can be used by all your teams. That way, when your scanners report a security issue on your shared code, you will only have to patch it once. This can be especially helpful for code accessing a database, doing calls over the internet, code that deals with authentication and authorization, and code that directly uses underlying OS features.

Another common practice in larger organizations is to move to shared base images. Separate teams often choose their container base images in isolation, without checking what other teams are using. Teams should use shared base images as much as possible.

When you update a shared base image in response to a discovered vulnerability, all teams using that image will benefit immediately. If each team were using its own image, each one would have to be updated separately. With shared images, any build/test/deploy cycle automatically uses the updated version of any base image it uses.

DevSecOps in production

Once all the security practices are in place and are part of your default development process, it's vitally important to keep monitoring. Code that is perfectly safe today may contain known security vulnerabilities tomorrow. Monitor the software that's already running, as well as code that's actively being developed.

You can do this with tools like **Splunk** or **Prisma Cloud**. Generate your reports automatically and in a format that all interested parties understand. Monitoring and logging tools like **Honeybadger**, **Honeycomb**, or **LogDNA** can help significantly — and there are CircleCI orbs that let you quickly integrate them with your pipeline.

When you're hosting in a cloud environment, make sure to check the monitoring tools of that environment. Azure has Application Insights, and AWS has CloudWatch Application Insights. Put them to good use. They can track malicious login attempts, unauthorized access, and errors coming from your application.

Third-party tools often add value by making it easier to get started, making monitoring accessible for other teams, generating reports, and monitoring additional metrics.

Patching software

It’s important to patch your software as soon as possible when your tools report a vulnerability. Unfortunately, updates may break software, and that includes updates from open source projects and third-party vendors.

To limit any risks from patching, be sure to follow sound development practices in your patching process, including DevOps principles such as **automated unit testing and integration testing**. Integration tests, especially, allow you to patch software with confidence that the fix is not causing additional problems. Automating integration tests will also significantly reduce human efforts on releasing a patch. If you're sharing standardized assets between teams, you'll be sure that all teams get the update.

The next level: CircleCI orbs

If you're using CircleCI for your CI/CD pipelines, you should consider using **CircleCI orbs**. Orbs are reusable, shareable, open source packages of CircleCI configuration that enable the immediate integration of many third-party services, including valuable security tools such as scanner services. CircleCI offers many vulnerability scanning orbs that make it easy to integrate vulnerability scanning into your pipeline with minimal time spent on setup. With orbs, you get an out-of-the-box solution for securing your pipeline.

You'll find scanners for the tools we already mentioned like Alcide, Snyk, and Stackhawk, and there are more scanners like:

- **Anchore** (for images)
- **AWS Parameter Store** (for managing and loading environment secrets)
- **Checkmarx** (for static and interactive application security testing)
- **Probely** (for scanning your web application for vulnerabilities)
- **Secret Hub** (to provision passwords and keys to applications)
- **SonarCloud** (for continuous code quality scans)

Other orbs for development on web applications like Docker, Kubernetes, Heroku, AWS, Azure, and Google Cloud **are also available**.

If you'd like to use a security scanner that doesn't yet have an orb, you can **create one** and push it to the open source CircleCI Orb Registry to contribute to the community.

Wrapping up

The DevSecOps approach to incorporating security awareness into DevOps practices offers a relatively painless way to leverage CI/CD to add vulnerability scanning and management to your existing deployment pipelines.

You can build this up over time by first introducing basic scanning to get development teams used to DevSecOps, then increasing the number and type of vulnerabilities you scan for over time.

Vulnerability management is just one area where CI/CD acts as a force multiplier for development teams. Building resilient systems allows teams to ship high-quality code in less time with lower risk. By putting your CI pipeline to work for you, you've got access to a key differentiator and leverage point for your company.

If you're ready to give it a try, you can create a complete DevSecOps pipeline on CircleCI.

Get started for free by creating an account at **https://circleci.com**.