GitGuardian

DevSecOps.

Protecting the Modern Software Factory

Over the last 30 years, DevOps supplanted Agile, which itself had come to revolutionize Waterfall development. Loosely coupled microservices are now considered state-of-the-art to implement service-oriented architectures. Development timeframes have been compressed, deployments are done on a weekly or daily basis, and the cloud now supports a highly dynamic supply of computing capacity, infrastructure, storage, and network.

The DevOps philosophy has often been summarized by the slogan *"move fast and break things"*, which means that because it's so easy to deploy source code to production, you should be using this leverage to innovate faster, and fearlessly.

But there is a catch. DevOps organizations still need to satisfy security and compliance criteria, because cybersecurity's fundamental mission remains the same: make sure things work as they should, and *only* as they should. The high flexibility and openness of modern software supply chains force us to rethink them.

That's the core value proposition of DevSecOps: **imagine new security solutions to better protect the modern software factory.**

However, the road is not without challenges. A partnership is needed between development, security, and operational teams to make security a frictionless process. Supply chains and pipelines are becoming the preferred targets for attackers, and have to be protected in a holistic manner.

The application security shared responsibility model is a stepping stone on this road. It enables a platform to integrate automated security solutions, start small, build up capabilities, install feedback loops, and strive for continuous improvement.

01

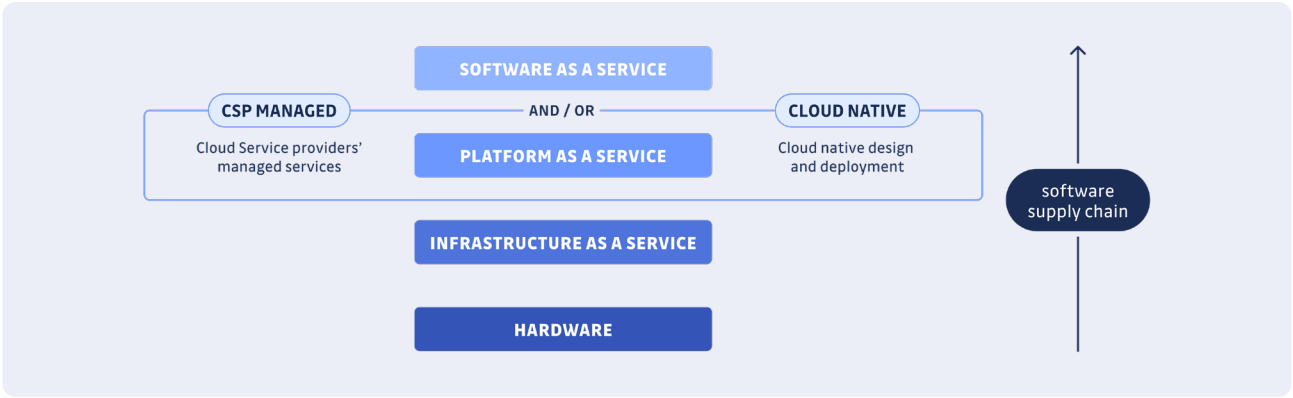
Security a next step for DevOps

DevOps is all about shipping fast. The goal is to make software experimentation efficient – and simpler. But security remains a bottleneck. For example, control checks might happen at the end of delivery lifecycles. Or after a release. This creates additional effort for development teams, which in turn, causes software delays – and frustration.

DevSecOps promise consists literally in *inserting* security principles, practices, and tools into the DevOps activity stream, reducing risk without compromising deliverability.

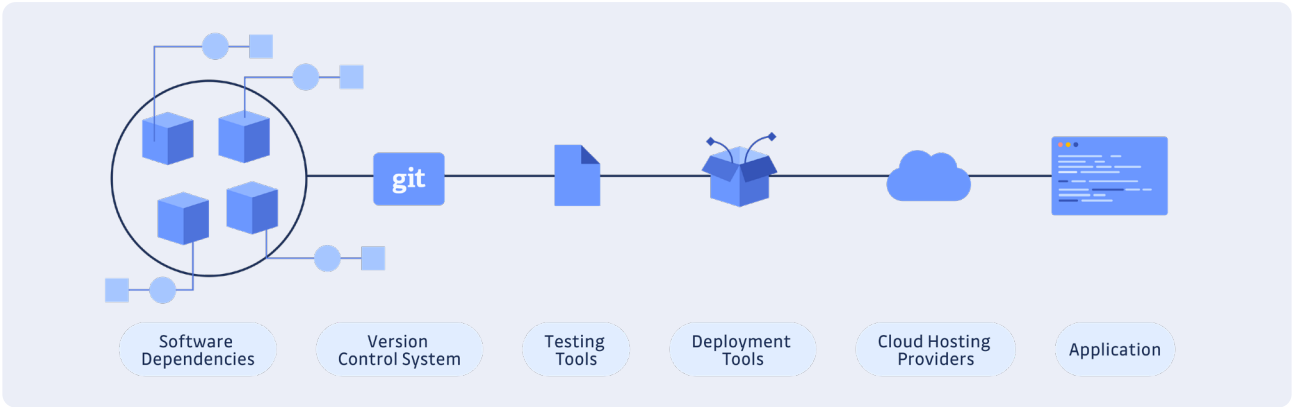
Understanding the modern software factory weak spots

Modern DevOps teams understand the importance of the **software supply chain** in their work. A supply chain is a broad term encompassing a lot of different realities (open or closed source tools, dependencies, platforms...) to describe the way modern software is built. Formally, the software supply chain is a logistical pathway that covers the entirety of all the hardware, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), tools, and practices that are brought together to deliver software capabilities. **It is a layered structure on top of which companies can build their software factories**.



Layered nature of software supply chains

It is extremely rare to see companies producing software that is 100% in-house. Most of the modern software firms are using hundreds, if not thousands, of building blocks, from open-source libraries and tools, deployment systems, cloud infrastructure, and SaaS services. Each of these is in turn the final product of its own supply chain, on which you don't have control and very poor visibility.



Supply chains composed to build a final artifact

CASE OF SUPPLY CHAIN ATTACK

SolarWinds

The SolarWinds attack has become the poster child for supply chain attacks because it is one of the worst-case scenarios: the compromised system (SolarWinds Orion) was used by tens of thousands of downstream systems and run with privileged access to the networks.
Even more serious, users included high-profile clients such as the US military, state departments, and some of the biggest IT corps. All these organizations fell victim to the Sunburst attack which used the supply chain software to bypass state-of-the-art security protections and remained undetected for a significant amount of time. Read about other recent attacks involving supply chains here.

CI/CD PIPELINES

CI/CD pipelines are the backbone of DevOps. They are linking a developer's work to its deployment to production. They are being implemented in software factories, making the most of cloud capabilities, to streamline four building blocks:

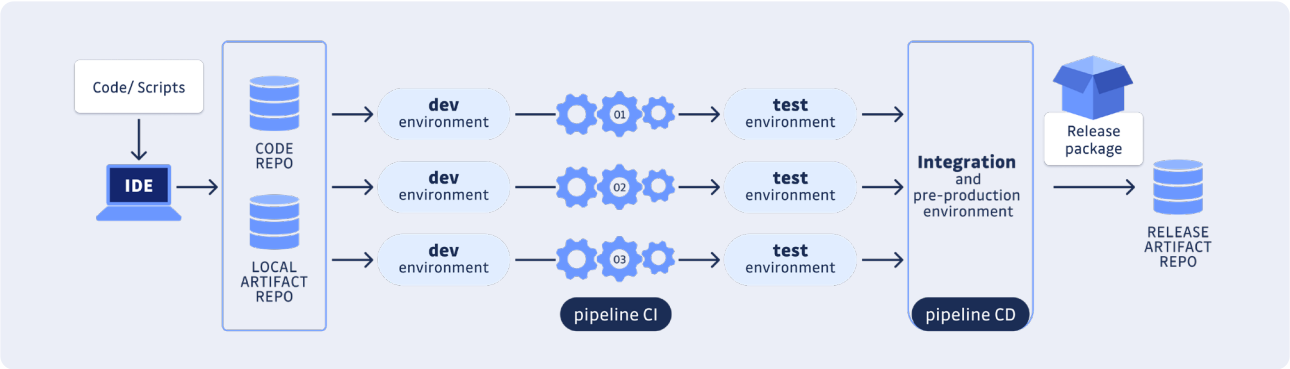
CONTINUOUS INTEGRATION
source code in shared repositories

CONTINUOUS TESTING
automated tests after every commit

CONTINUOUS MONITORING
gather metrics about environments and applications

CONTINUOUS DELIVERY
one-click deploy into production environnment

These four pillars implemented together allow teams to reach the ultimate goal of DevOps: **Continuous Improvement**.

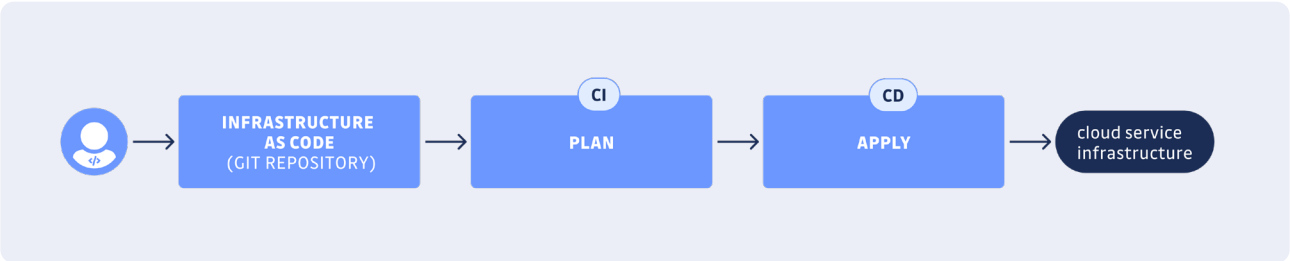


A typical software factory

Infrastructure as code

Not only products and services, **but the infrastructure itself** can be the result of a CI/CD pipeline: **Infrastructure as Code** or IaC is the process of provisioning and managing infrastructure defined through code, instead of doing so with manual processes. With IaC, users don't need to configure an environment every time they want to develop, test, or deploy software. All infrastructure parameters are declared in files called manifests, which are managed through version control. Manifests make building, testing, staging, and deploying infrastructure quicker and consistent.

Infrastructure as Code has become the de-facto solution to meet the demands of Dev-Ops modern rapid software development cycles.



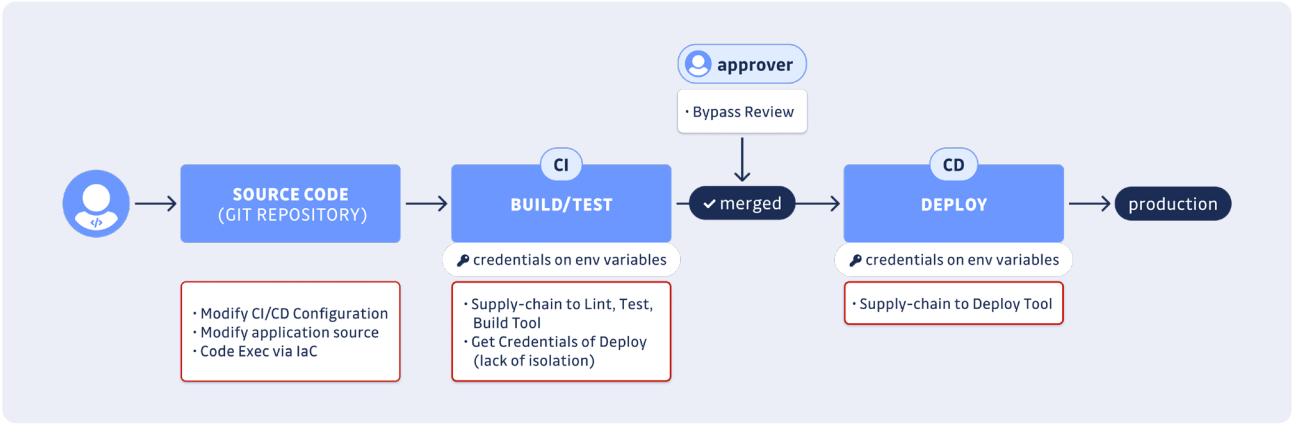
A pipeline-built cloud infrastructure

The big advantage of these pipelines is that they enable **incremental changes to be delivered** which are much easier to design, review, ship, monitor, and roll back if necessary.

Their downside, however, is that every step of a pipeline is one more attack surface. More worrying, these represent higher-valued targets for malicious actors: not only can they hold production-related **credentials**, but also because it's where an established attacker could silently modify the source code in what is known as **code tampering**.

Last year hundreds of Codecov users were affected by code tampering abusing their CI, learn more here.

As a result, supply chains are increasingly vulnerable because of their increased openness and complexity. They leave a lot of grey areas in traditional information security frameworks.



Some attack surfaces examples

Pipelines and their supply chains have outpaced traditional cybersecurity and threat modeling analysis frameworks. Various initiatives have been recently proposed to fill this gap (Google SLSA, MITRE ATT&CK, or NIST), yet taking action is urgent: the intersection of a high rate of adoption, poor manageability, and security oversight with high potential rewards have created a unique opportunity for malicious actors.

Organizations are acknowledging this new reality and looking to set up the right balance between productivity and resilience.

Security must preserve developers' productivity

It's no secret that security can be a cause of friction when developing software. When time-to-market is critical, anything impeding velocity can be perceived as a bottleneck.
The pain points include:

- Lack of security expertise, or visibility to deliver safe code across the lifecycle. Developers are encouraged to ship faster and remediate later.

- Security ownership: Who is responsible? The cloud provider? The open-source dependency maintainers? Organizations often make the flawed assumption that security questions have already been addressed by another link in the chain.

- Security professionals and software engineers can act in isolation from each other, so security is compromised by miscommunication.

- Both existing API-based online applications and evolving micro-services environments can become large-scale and highly distributed, increasing the scope and complexity of the attack surface.

These issues do not only create risks, they can also be the origin of a more risk-averse attitude.

As systems expand and get more complex, as people change, you have a natural trend towards more risks and downgraded observability due to an increase in unknowns. **Fear of missing security coverage can impact progress, disabling or slowing attempts at innovation**.

It is clear that a more proactive approach to risk management and security is vital to preserving the benefits of DevOps.

The core value proposition of DevSecOps

ACCELERATE DEPLOYMENT FREQUENCY

When security encloses the pipeline, speed will eventually improve. You have to be realistic and accept that the early stages of integration can be difficult. As teams learn to collaborate more smoothly, these problems will eventually go away and only the positive output will remain.

DECREASE TIME TO REMEDIATE CRITICAL VULNERABILITIES

DevSecOps will drastically improve mean-time-to-remediate thanks to automated scanning, better communication, and a shared responsibility model. When responsibility for security is shared across the pipeline, rather than siloed within one team, security issues are caught earlier. This in return is directly linked to cost efficiency, since it costs much more to fix a bug found during regular maintenance than to fix one identified during the design phase.

IMPROVE YOUR SECURITY POSTURE

Overall, the modern way to build software is complex. Including security as a feature from the start is a big win for the organization's global security posture. It saves a lot of time to security teams by eliminating benign issues or false positives thanks to automated processes gating at every step. It eliminates the need to retrofit security controls post-development. It will create a new culture where security best practices are shared and beneficial to all—from building, deploying, to securing production workloads.

To operate the shift towards DevSecOps, companies must be ready to tackle challenges.

02

Challenges to overcome

Rearchitecting the software factory to embed security is easier said than done. It requires finding a good balance between developers' and security needs while preserving agility and minimizing the risk of a security failure. But first and foremost, teams' objectives and priorities need to be realigned.

To do so — and without waiting for cybersecurity frameworks to come up with a DevOps-ready security model — you should examine what tools your organization needs and how to leverage them to build on your present capacity and deliver value.

Reconciliation of the Dev and AppSec teams

A survey conducted by the Ponemon Institute in 2020 was a powerful revealer of the cultural divide existing between developers and application security professionals. The institute found out that the perception of this cultural divide was not only high cohorts but also more much pronounced on the application security side (75% vs 49% on the developers side). They were also much more concerned about increased risk for their organization (not surprising), and thinking that

> *development teams push code with known vulnerabilities, with many also complaining that developers accept flaws if they believe an app will be a big seller.*

The study is a clear indication of the lack of ownership on security: 67% AppSec think their teams are responsible, while only 39% of developers said the same for their team.

DEVELOPERS

- Speed is a priority
- Need to remain innovative while meeting more pressurized deadlines than ever before
- Reviewed on productivity
- Think AppSec team don't understand the pressure to deliver and innovate
- AppSec is harder than other areas of security
- The cultural divide has a serious impact on meeting deadlines
- Admit working together is difficult

APPSEC

- Security is a priority
- Need to make sure code vulnerabilities don't reach production
- Reviewed on the number of incidents
- Think developers do not have visibility into the overall application security
- Published code contains vulnerabilities
- Think the cultural divide is putting the security of applications at risk
- Admit working together is difficult

The end result is that **priorities, goals, and objectives are not aligned**.

A cultural change is required to bring these opposed mission objectives together. But culture by itself is not enough. You have to think about integrating new tools to help break the silo.

The need for tailored automated solutions

Bringing AppSec into DevOps means you need tools that can be completely integrated into the development workflow: IDEs, version control systems and build tools. Then it is a matter of smartly placing guardrails all along the path, allowing developers to move forward with relatively low friction and high visibility.

With automated reports set up, it creates a common interface to facilitate the exchange between application security and developer teams while enforcing processes. It is a huge opportunity to automate most of the low-value tasks and make sure engineers focus on their area of expertise.

Take pentesting for example. It is a highly valuable security practice where you can quickly gain a realistic vision of the number and magnitude of breaches across your systems by simulating a realistic attack. But pentesting requires days of expert work to uncover only one attacking vector out of the many that possibly exist. In other words, it cannot be scaled. Defenders know how the application they are protecting works under the hood. This is one weapon you have, that attackers don't.

By leveraging the deep knowledge of the system developers have, you can finally leverage your advantage position to find defects in a faster, more efficient way. It's only a matter of adopting the right tools for the job.

That's why only specially designed AppSec technologies can enable DevSecOps.

Traditional technologies required specialized AppSec human efforts to onboard, configure, and operate the tools, and didn't cover the entire software life cycle (SLC). Most checks ran just before application deployment and, occasionally, after deployment, which isn't a fit for DevSecOps.

We can see DevSecOps tooling as the set of capabilities that directly increases pipeline governance in order to reduce application and infrastructure risk. **DevSecOps tools have capabilities that automate best practices**, augment the pipeline, and support development activities, addressing security challenges across the software development to operations pipeline.

Broader security tooling, such as tools for threat modeling, application and infrastructure scanning, hardening, and event can be seen as important inputs to DevSecOps. **However, these are not DevSecOps tools, and vendors that offer scanning tools with an API that integrates with CI/CD are not necessarily DevSecOps vendors**.

Lack of industry standards is a slowing factor

The accelerated digitization of the world propelled by the global pandemic has cast a crude light on one reality: **cybersecurity frameworks are lagging behind the reality of most modern organizations**, even the ones that haven't gone 100% virtual. This has been happening for years, but as the gaps widen between the compliance-related security standards and the actual security challenges on the ground, DevSecOps adoption may be hampered.

Standards are important for multiple reasons. First, a framework is necessary to create a reference point for everyone. It encodes knowledge so others can practically assess their security posture. Second, they are used as the basis for compliance rules. This means that a static, outdated, framework is doomed to cost a lot of money to comply with little to no effect on actual security.

Promising steps

NIST is actually working towards a DevSecOps-specific framework, conscious that its current standards (like the well-known Secure Software Development Framework) are not adapted to the modern DevOps way of building software:

"There are many existing security guidance and practices publications from NIST and others, ***but they have not yet been put into the context of DevOps****. Updating affected NIST publications so they reflect DevOps principles would also help organizations to make better use of their recommendations."*

The institute has also been mandated *"with enhancing cybersecurity through a variety of initiatives related to the security and integrity of the software supply chain"* by the government's Executive Order 14028.

Fortunately, many of the industry's top players have been actively working on this issue and are now pushing new, open, propositions that could benefit the industry as a whole.

One of the most interesting has been Google's SLSA (Supply chain Levels for Software Artifacts), an end-to-end framework for ensuring the integrity of software artifacts throughout the software supply chain. This framework focuses on identifying and mitigating issues in CI/CD pipelines and defines 4 levels of adoptable guidelines. The idea is to settle on an industry-wide consensus over clearly defined guiding principles, which would benefit both producers and consumers: the former has a clear path forward to make their software more secure, while the latter can make better decisions based on the software package's security posture.

> *Achieving the highest level of SLSA for most projects may be difficult, but incremental improvements will already go a long way toward improving the security of the open-source ecosystem.*

Using a similar approach, security engineer Hiroki Suezawa proposed in 2021 an ATT&CK-like matrix on CI/CD Pipeline specific risk, several attack scenarios, and how to defend.

While there is a lot of literature on secure development, from secure coding to securing the global lifecycle, CI/CD pipeline or supply chain security is mostly a blind spot for most existing frameworks. As they become more relevant for public breaches, guidelines like SLSA will undoubtedly have a positive impact on increasing software resiliency. Until we see more general adoption, companies are left to build their own, DevOps-adapted, security model.

In order to do so, they should be inspired by the nascent cloud security model that has been elaborated to rationally and efficiently allocate security between stakeholders.

03

Shared Security Model

Most cloud providers have a shared responsibility model. The provider is only responsible for security ‘of’ the cloud, while customers are responsible for security ‘in’ the cloud.

They provide the physical and architectural security, along with tools to properly secure the services they offer, but it is up to the user to configure those settings properly.

To make the parallel, DevSecOps should be implemented in such a way that AppSec can provide Devs and Ops with the platform and the tools they need to secure their workloads, all along the pipeline, to finally break the security silo.

A new generation of DevOps-native security tools

In its broadest sense, security needs to be thought along these 5 axes:

- Management and **governance** for guiding **prioritization** and resourcing.
- Application **hardening**, **instrumenting** code, **remediating** vulnerabilities and highlighting and **fixing** poor practices.
- Security **testing** of code and external services, including vulnerability scanning, penetration, and regression testing.
- Mitigation of **infrastructure risk** and verifying and applying security mechanisms to **platforms** in use.
- **Operational security**, incident and event management, identification, diagnosis, and resolution of vulnerabilities.

As we have said, many of these areas are already evolving apace.
While automation is key, **it is *not* the only principle**.
To be successful, tooling needs to handle complexity across development targets, work with minimal interruption to the innovation process, and support collaboration.
This not only applies to security, engineering, and operations professionals, but also to the management teams.

A true DevSecOps solution can be a stand-alone tool/dashboard or working with existing frameworks, but it needs to expand capabilities by providing a process-level view and satisfying criteria:

DESIGN STAGE

Supports application-specific policies and their collaborative development, potentially stored as code.

DEVELOPMENT STAGE

Offers guardrails and potentially automated remediation, integrating with developers' environment (CLI or IDE plugin).

TESTING STAGE

Presents a clear view of outstanding risk based on multiple scanning and testing sources.

DEPLOYMENT STAGE

Offers visibility to all the stakeholders so they can be sure the artifacts align with their security policy.

Fortunately, we are seeing more and more "DevOps-native" solutions emerge in the security field. By DevOps-native, we unusually mean tools that are language/framework/architecture agnostic, purposely built to be deployed in a cloud environment, and accessible remotely. It should automatically test any application architecture, whether it's a single-page application or multi-page application, a microservice, or an API.

AppSec in particular, should "speak the language" of developers, and be accessible:

- Easy onboarding and configuration.
- Doesn't require customized login/credential handlers or any other heavy human involvement.
- Covers the entire software lifecycle and be able to test any business increment, not just a particular endpoint.

This doesn't mean that developers shouldn't learn best security practices, on the contrary, any DevSecOps platform should be able to softly propagate best practices. For that, developers should not be required to learn how to operate AppSec technologies. This is why trying to retrofit ancient AppSec technologies into a DevOps-oriented factory is not a good idea. Tools are meant to empower people, not the contrary.

Empowering developers beyond DevOps

Most of the time, if developers are not using the tooling at their disposal, **it isn't because they don't want security, but rather because they are too complicated**.

A mindset change is necessary. Selecting the right tools to maintain security and deliver value should be one of the top priorities of a DevSecOps roadmap.

The idea is simple: **make the safe path also the shortest**. Developers will end up taking that route naturally without even having to make it mandatory or needing to understand the details. For instance, Single Sign-On is a great example of an operational side of security that everybody uses without thinking about it. Users don't have to type in passwords, administration is easier, and password-based attacks are neutralized. If it actually works, everybody wins. If we stretch the reasoning, why not consider getting rid of patching servers with serverless if that can cover most use-cases?

Product owners and managers have a role to play in this scenario. Good decisions regarding product security are far more difficult than other technical aspects, yet they should be taken. Baking security and risk-based approaches into development are what will empower developers to build and deploy secure, high-quality software.

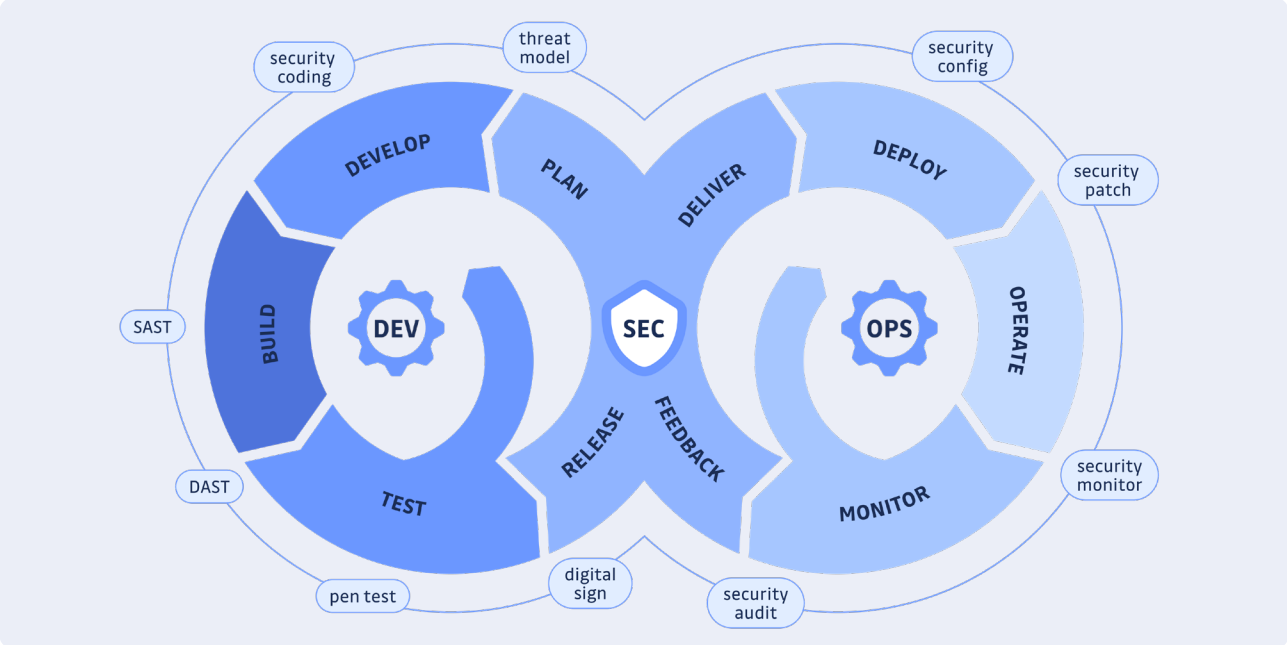
When policies and workflows are well established within microservices and guardrails set up, developers are also empowered to develop faster with more ownership. Operators can be sure that their teams are consistent and comply with established business requirements.

EXAMPLE

False positives

If we want developers to use scanners in their pipelines, it is mandatory to select a tool able to maintain a quality signal noise: because of alert fatigue, false positives will make or break the success of any security process. According to a 2020 report from the Neustar International Security Council (NISC), over one-quarter of security alerts fielded within organizations are false positives. Despite all, achieving zero false positives is not a panacea either, read why.

DevSecOps to improve visibility, control, and compliance



DevSecOps distinct lifecycle and operations

The road to a successful DevSecOps program is highly dependent on increased visibility. Metrics and visibility help drive change. Vulnerabilities ownership is enabled by sharing the reports. Of course, the reports must be accurate and digestible in the first place. That's one reason why automation is key to building a reliable security program.

Some success criteria include (but are not limited to):

- Minimal friction between developers and security.
- All vulnerabilities are tracked in the same system as normal bugs.
- All vulnerabilities are processed through the same workflow (bug bounty, pentesting, tools, internal tests).
- Relevant metadata is tracked and enriched automatically and/or by hand.

To achieve these results, there should be at least 4 stepping stones for continuous security monitoring:

01 CONTINUOUS SECURITY MONITORING OF APPLICATION/API INVENTORY

The first step in any application security program is to automate the discovery, profiling of exactly what code you're running across your environments. You can't secure what you don't know. Your attack surface spans production code in a variety of data centers, virtual environments, private or public clouds, containers... All of which may be running different versions or branches of the code.

The DevSecOps approach to this problem is to use a combination of automated discovery and self-inventory tools. Discovery tools help you identify what applications and APIs you have. Self-reporting tools enable your applications to inventory themselves and report their metadata to a central database. An up-to-date inventory is always the starting point.

02 CONTINUOUS SECURITY MONITORING FOR CUSTOM CODE VULNERABILITIES

Next comes the continuous monitoring of all your software for vulnerabilities throughout development, test, and operations. Most vulnerability discovery and remediation should "shift left" to very early in the software development process. Of course, some applications aren't in active development yet need vulnerability discovery too.

Static and dynamic scanners are your weapon of choice here. For them to work in a modern CI/CD pipeline, they should be able to scan any significant increment in a timely manner. Remediation enabling tools are the primary way to empower developers to fix vulnerabilities themselves and check in clean code.

Because they also require significant triage effort for every scan, accuracy is a prominent factor.

A low-hanging fruit

One special note about a typical, potentially devastating, vulnerability that can be monitored at this stage: **unmanaged secrets and poor privileged access controls**. Secrets may include privileged account credentials, SSH Keys, APIs tokens... and may be used by humans or non-humans (e.g., applications, containers, microservices, and cloud instances).

Most facets of DevOps are highly interconnected and utilize secrets: a typical DevOps environment may leverage several dozen tools that all require secrets management. Additionally, to help expedite workflows, DevOps teams may allow almost unrestricted access to privileged accounts (root, admin...), by multiple individuals, who may share credentials — a practice that virtually eliminates the possibility of a clean audit trail. Various orchestration, configuration other DevOps tools may also be granted vast privileges.

Inadequate secrets management is a common shortcoming of DevOps environments, providing an avenue for attackers to tamper with controls, disrupt operations, steal information, and basically own an organization's IT infrastructure. With privileged access rights in hand, a hacker or piece of malware can gain full control of the systems and data, so it's essential for organizations to rein in excessive privilege rights and access.

03 CONTINUOUS SECURITY MONITORING FOR OPEN SOURCE VULNERABILITIES AND LICENSE VIOLATIONS

The use of open-source software (OSS) is exploding, and DevOps companies are highly leveraging these powerful libraries. **Every time you add a library, you take on the risk that there is a security vulnerability (or multiple vulnerabilities) lurking in that code**.

Since the typical modern application has hundreds of these components and dozens of new vulnerabilities are disclosed every week, the risk and work both add up quickly. Fortunately, there are many solutions available for continuously monitoring your applications for library problems. The best will not only track your libraries, report vulnerabilities, and report license violations but will also tell you exactly why each library is included and whether it is actually invoked at runtime. Without this knowledge, you will waste a lot of time updating libraries that can't be exploited.

04 CONTINUOUS ATTACK MONITORING AND RUNTIME EXPLOIT PREVENTION

Finally, don't forget about continuously monitoring and protecting your applications in production. Even the best software projects and security teams miss vulnerabilities. And even if they were perfect, new library vulnerabilities are frequently discovered.

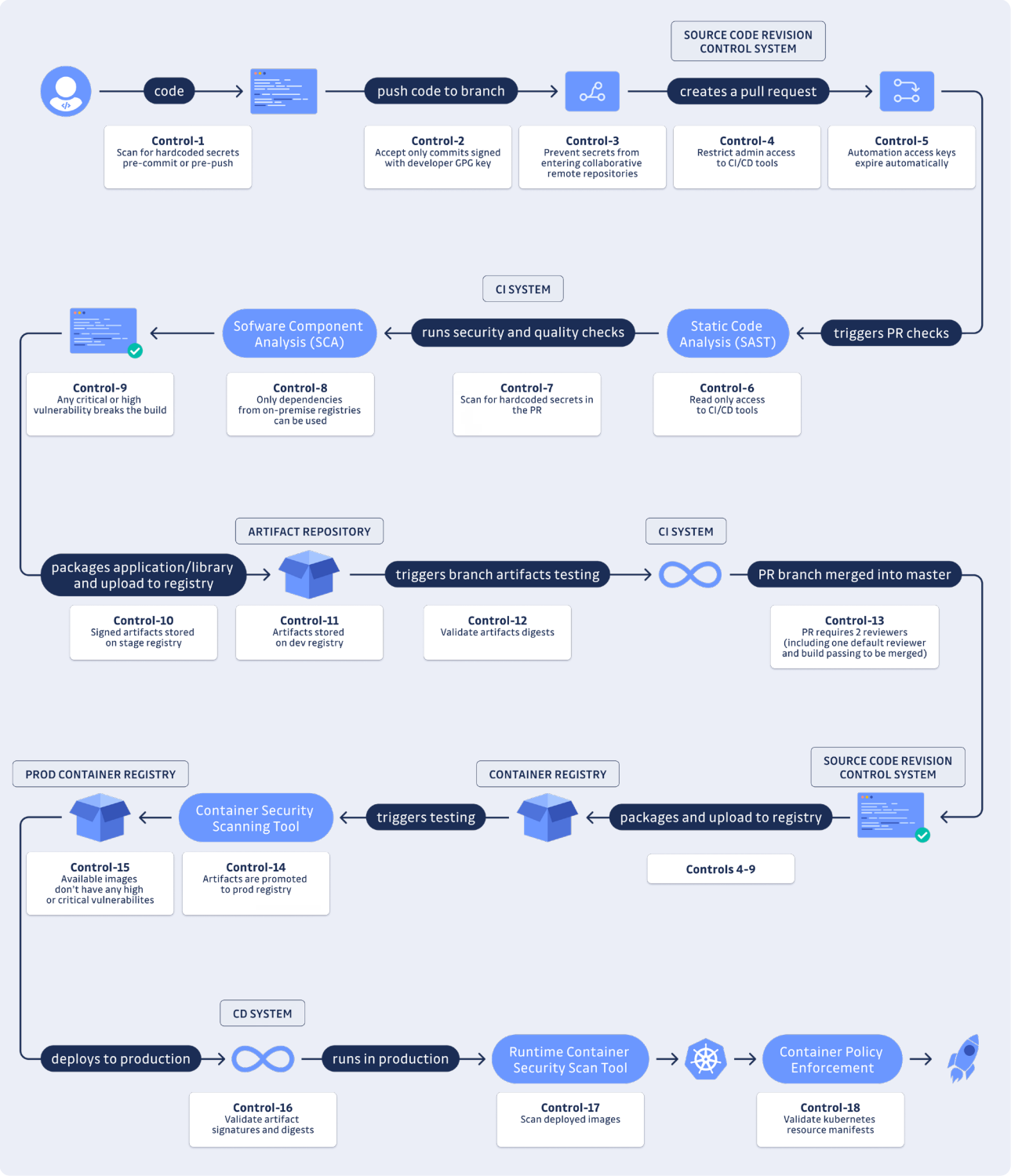
CASE OF A MAJOR OPEN-SOURCE COMPONENT VULNERABILITY

Log4Shell

Log4Shell is a software vulnerability in Apache Log4j 2, a popular open-source Java library for logging error messages in applications. It was publicly disclosed on 9 December 2021 with a CVSS severity rating of 10, the highest available score. What makes Log4Shell so dangerous is how ubiquitous the Log4j 2 library is: it's present in major platforms from Amazon Web Services to VMware, and services large and small. The ease of exploiting the vulnerability compounds its impact. Attackers can remotely take over any internet-connected service that uses certain versions of the Log4j library anywhere in the software stack.

Simply knowing who is attacking you, what types of attack vectors they are using, and which systems they are targeting is itself valuable. This threat intelligence will inform your threat modeling and security architecture process and help you make smart decisions.

Example: The legacy web app firewall (WAF) approach can only use HTTP information to differentiate attacks from legitimate transactions. With modern application protocols and data formats, this approach is extremely noisy and time-consuming. The modern approach is to use Runtime Application Self-Protection (RASP), which instruments applications, directly measures attacks from the prevents exploits from within.



Supply chain controls

Here is an example of what DevSecOps controls can look like when implemented all along the pipeline:

Automation is not only a matter of saving time, it is also essential in maintaining consistency between the distinct steps. Without it, security would not be reusable, scalable, and we would not be able to gather metrics on a centralized platform.

Conclusion

Security can't scale if it's siloed, and slowing down the development process is no longer an option in a world led by DevOps innovation. The design and implementation of security controls are bound to evolve. DevSecOps promises to ensure cloud software factories are well-protected against the emerging threat of supply chain attacks, but also that developers will be empowered to build and run safer software.

A lot can be improved on these fronts, and for that application security, development and operational teams need to take over a new generation of DevOps-native security tools. They are the gateway to creating an automated, tailored detection and remediation system. This security guardrail is integrated and unified to protect every step of the software development cycle, which in the DevOps era tends to be assimilated to CI/CD pipelines.

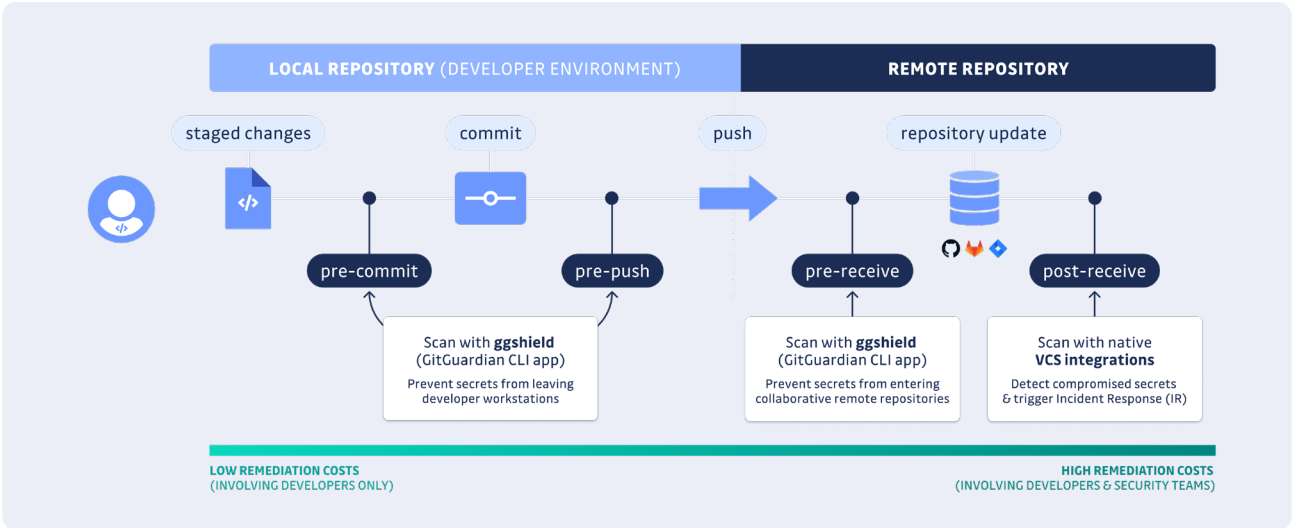
When security is rethought as a partnership between stakeholders, who are the different teams collaborating to produce software, a flywheel effect can take place: reduced friction leads to better communication and visibility, automating of more best practices, easing the work of each other while improving security, fewer defects, and so less friction and better productivity for all. This is how security will finally benefit from feedback loops and strive for continuous improvement.

About GitGuardian

The new ways of building software create the necessity to support new vulnerabilities and new remediation workflows. These needs have emerged so abruptly that they have given rise to a young and highly fragmented DevSecOps tooling market. Solutions are specialized based on the type of vulnerabilities being addressed: SAST, DAST, IAST, RASP, SCA, Secrets Detection, Container Security, and Infrastructure as Code Security. However, the market is fragmented and tools are not well-integrated into the developers' workflow.

GitGuardian, founded in 2017 by Jérémy Thomas and Eric Fourrier, has emerged as the leader in secrets detection and is now focused on providing a holistic code security platform while enabling the Shared Responsibility Model of AppSec. The company has raised a $56M total investment to date.

With more than 150K installs, GitGuardian is the n°1 security application on the GitHub Marketplace. Its enterprise-grade features truly enable AppSec and Development teams in a collaborative manner to deliver a secret-free code. Its detection engine is based on 350 detectors able to catch secrets in both public and private repositories and containers at every step of the CI/CD pipeline.



GitGuardian monitors every step of the pipeline

> GitGuardian's mission is ambitious but is built on a very simple philosophy at its core. **Developing** and **launching secure applications** must be a **shared responsibility** between **Dev, Sec**, and **Cloud Ops**. Developers, in particular, want a wingman at every step of the SDLC to help them write more secure code without limiting their productivity. And as defining threat signatures and keeping pace with the thousands of technologies that developers use will always be a never-ending battle, we have already laid the foundation of a powerful and flexible code security framework that can be extended rapidly to encode a wide variety of vulnerabilities.
>
> Jérémy Thomas, GitGuardian CEO

GitGuardian is a global cybersecurity startup focusing on code security solutions for the DevOps generation. A leader in the market of secrets detection and remediation, its solutions are already used by hundreds of thousands of developers in all industries.

www.gitguardian.com