Accelerate and secure your SDLC with DevSecOps

sumo logic

A phase-by-phase guide

WRITTEN BY:

Chas Clawson
Field CTO, Security
Sumo Logic

Colin Fallwell
Field CTO, Observability
Sumo Logic

DevSecOps is a term that's grown in usage and popularity.*

Driving the adoption of DevSecOps is the need for teams to maintain innovation at speed and establish a competitive differentiation that maintains and accelerates market share.
In the past security was often pushed to the end of the software development life cycle (SDLC). As hacks have become more prevalent, more costly, and more difficult to detect, security has become of top-line importance to teams.

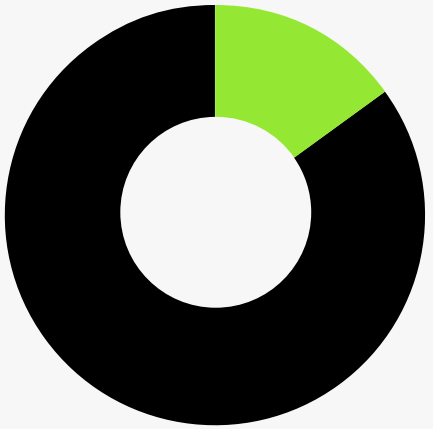
In short, when your team has to go from requirements to production in a day — and do it safely and securely — DevSecOps is the path that gets you there.

Shared dashboards extend visibility across teams.

Elite teams embracing DevSecOps don't conduct security code reviews as an afterthought.

They have a security-first, security-always approach and integrate and prioritize security and observability at every step of the SDLC. They embrace DevSecOps practices everywhere from developer laptops to pushing code into production — and everything in between.

In this guide, we'll take a look at tactical steps you can take at each phase of the SDLC to help transform your team into an elite DevSecOps performer.



15%

of dev teams can be considered "elite"

Elite team performance

DEPLOYMENT FREQUENCY	On demand
LEAD TIME FOR CHANGES	Less than one day
TIME TO RESTORE SERVICE	Less than hour
CHANGE FAILURE RATE	0-15%

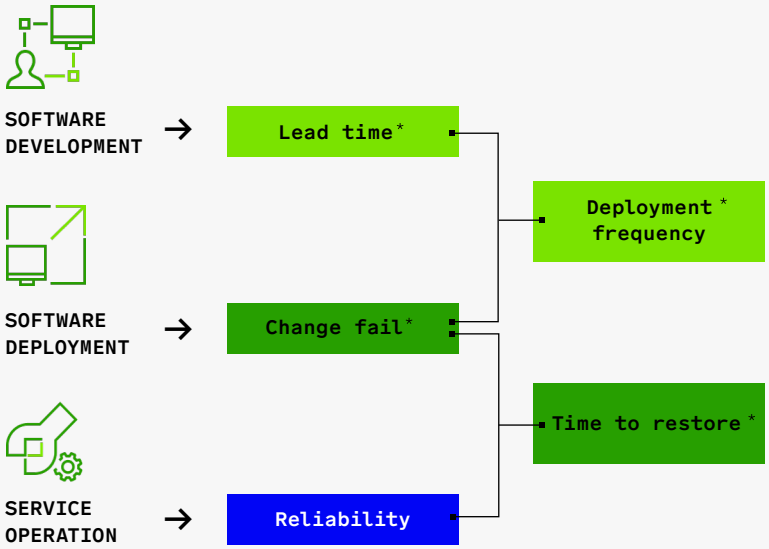
Elite teams, as defined by the DORA (DevOps Research and Assessment) metrics, deploy software multiple times per day, have a lead time for change and a restore of service in under an hour, and keep their change failure rate under 15%.

Source: Accelerate State of DevOps 2021

DevSecOps at every phase of the SDLC

DevSecOps can be viewed not only from a maturity perspective but from a discrete capabilities perspective.

That is, your team can add individual tools and practices one at a time. Further, teams adopting DevSecOps may need to think beyond the typical metrics used by development, security and operations teams. GSA, the Government Services Administration shares a range of high-value and supporting metrics and capabilities to consider as teams collaborate across disciplines, including deployment frequency and MTTR as well as frequency of vulnerability patching and privilege auditing.

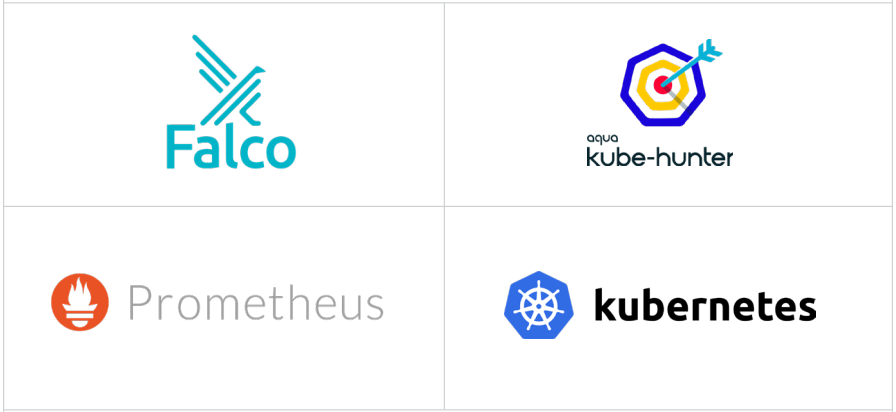


Performance metrics

* Key metrics
Source: “Is it ODD to shift left? Becoming elite DevSecOps performers” Colin Falwell, 2022

DevSecOps has changed considerably with the advent of highly-capable open-source projects.

Tools like Falco, kube-hunter, Prometheus, et al have become mainstays in practice as enablers of innovation speed. Today, enterprise security teams are adopting more open-source security solutions as they integrate with and compliment the growth in environments running open-source, such as Kubernetes.



Observability is foundational to enabling DevSecOps.

Building an observable system that provides telemetry to everyone involved — SecOps, DevOps, IT, finance, business intelligence (BI) and data science teams, and more — is critical.

Observability in DevSecOps is not just metrics and traces and logs for dev teams. It's embracing observability-driven development. This means dev teams are highly capable and mature in observing the internal processes that ultimately enable them to test code directly in production. They are measuring not just deployed services, but also CI/CD pipelines, telemetry pipelines, control planes for automation, processes that govern software delivery, the standards these processes employ and more. Having security processes as part of the DevOps pipeline reduces friction and increases adoption best practices.

Now let's step through the five phases of the SDLC and look at specific and valuable actions you can take to embrace DevSecOps.

1 The exploration phase

Two distinct DevSecOps aspects you can bring to the planning phase are creating a standardized specification and defining security inputs and outputs.

Creating a spec during the planning phase ensures that the needs of development, security and operations are all being considered. The spec should not only establish functional “what should the software do” requirements, but it should also define operational, performance and security requirements. These specifications should be defined for each supported language, and all pipelines for that language type should be standardized to the spec.

Operational and performance impacts:

Defining the upper and lower bounds for critical metrics

Golden Signals or any other metrics used to observe a system should have upper and lower specification limits that define a performance corridor that conveys normal operation. The metrics defined should answer common operational questions like “How does our environment behave during peak utilization?” “Can we identify the point when we need to scale?” and “Do I have the telemetry to detect anomalous behavior or malicious use of services?” Answering these questions in the spec makes it clear when you’re no longer meeting the agreement between all stakeholders and when security threats might unfold.

This is the basis for reliability management during the later phases of the SDLC. Your metrics should become the ‘canaries in the coal-mine’, signaling with confidence when action is warranted. It is also critically important to accomplish this with the least number of metrics possible.

Logging format and verbosity

A standardized logging format across the enterprise reduces churn. Define the metadata to include in the log message. Invest time gathering input from stakeholders. Logged attributes directly impact the quality of dimensional analysis in the automated machine learning (AutoML) that identifies root cause when canaries fall off their perches.

Open-source tooling decisions

Open-source software has become a foundation of observability. Tools like Prometheus, Grafana, and Jaeger create a common platform to share information across multiple teams. Commercial solutions are exclusive and proprietary, limiting what development teams can do and ensuring vendor lock-in. In a culture of continuous improvement owning the code and libraries to build observability is a necessity.

From a pure security perspective, a spec means defining security inputs and outputs.

For example, to support supply chain security policies, the development team may need to create a plan around developing a software bill of materials (SBOM) generation practice or need to expose the right metadata to support a downstream security information and event management (SIEM) software tool. Recommendations from security tools should be actionable and incorporated into the specification.

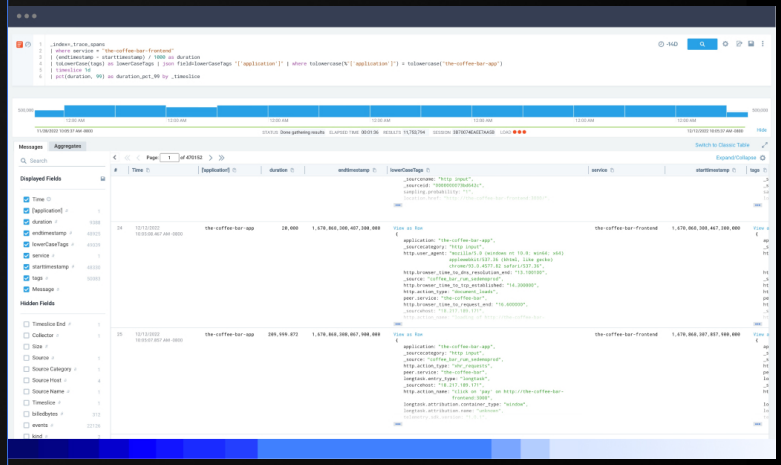
Infrastructure security is also imperative.

Making these decisions in a repeatable, measurable way during the exploration and planning phase allows you to integrate additional tooling like kube-hunter for vulnerability discovery or kube-bench for compliance. Making these decisions in this exploration phase allows you to integrate them into the overall deployment strategy and increase the speed at which you find security problems.

As part of creating your spec, you may also consider starting your threat model. Threat models are planning documents that consider and plan for various likely attack scenarios, the data that could be exposed by the attacks, and possible solutions. The threat model will evolve throughout your SDLC, but now is a great time to start.

Log data

Log data is critical for monitoring, troubleshooting and investigating reliability and security issues to get down to the root cause of “why” that issue occurred. Log data is often the most detailed information available about a company’s systems, so it makes sense to put that data to work, pulling log files across the organization into a single analytics platform for end-to-end visibility and faster troubleshooting.

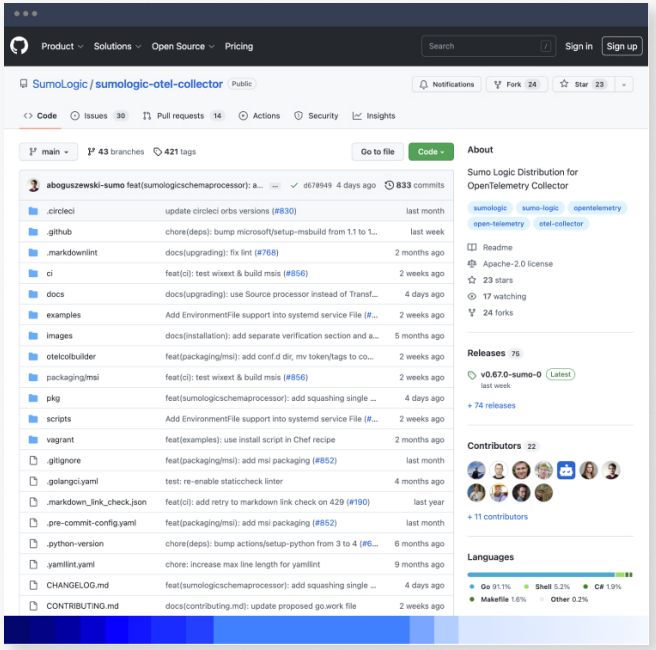


Log data is often the most detailed information about a company’s systems.

2 The coding phase

DevSecOps in the coding phase means building security into the code itself. In the exploration phase, you defined the development spec. In the coding phase, you'll implement that spec in your code. Two ways you might do this are observability and security policies and vulnerability prevention.

We live in a complex, multi-architecture, multi-microservice world that requires full observability across stacks and environments. In the context of DevSecOps, observability refers to making all phases of the SDLC visible to prevent security issues before they make it to production.



Sumo Logic Distribution for OpenTelemetry Collector

An easy-to-implement observability solution that considers logs, metrics, and traces is key. This is especially critical in a microservice architecture and across all of your application footprints. Practically, this means leveraging a tool like Sumo Logic Distribution for OpenTelemetry Collector.

In the coding lifecycle, you should also implement software supply chain security policies and vulnerability prevention (among other practices).

There are many approaches and tools to making code secure — some language-specific, some industry-specific and you'll need to research to find the ones appropriate for your project.

Common tool examples:

Static application security testing (SAST)

Tooling in this category often scans your code, helping you to catch common issues like SQL injection, buffer overflows, and other well-known vulnerability categories. While not perfect, the outputs from SAST tooling provide data points that inform the overall health of the code base.

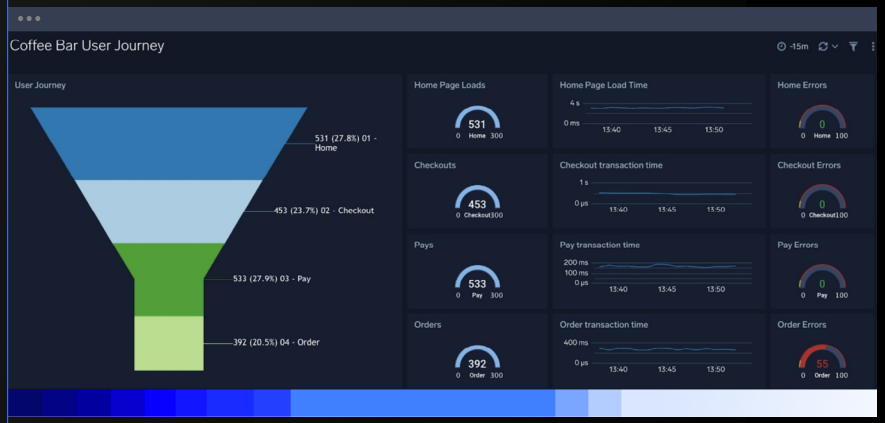
Author verification

Validating that commits in your code come from a verifiable source is another recommended security practice. In some cases, creating commits that are GNU Privacy Guard (GPG)-signed ensures that the author is verified. This also creates a bar for merging code into the mainline: no code merges unless its author is verified through a GPG signature. (GPG isn't the only option but it has some notable benefits over other options, such as key revocation.)

Reliability management

The coding phase is the root where you'll implement capabilities that support your reliability management framework.

For example, you'll need to consider how you'll measure your service level indicators (SLIs) against your service level objectives (SLOs) and how your error budget informs how you prioritize work. Exceeding your error budget will trigger more work to bring it back in compliance with your service level agreements (SLAs).



Solve customer-impacting issues faster with visibility to all your application data including logs, metrics and traces across the entire development lifecycle.

3 The building phase

The building phase should focus on reproducible and verifiable builds. Code that is committed or checked into the repository should create a deployable artifact that is certified reliable and secure.

Many tools and practices can help with the above goal, as CI/CD is a practice unto itself.

Real-world examples of CI/CD practices you can implement:

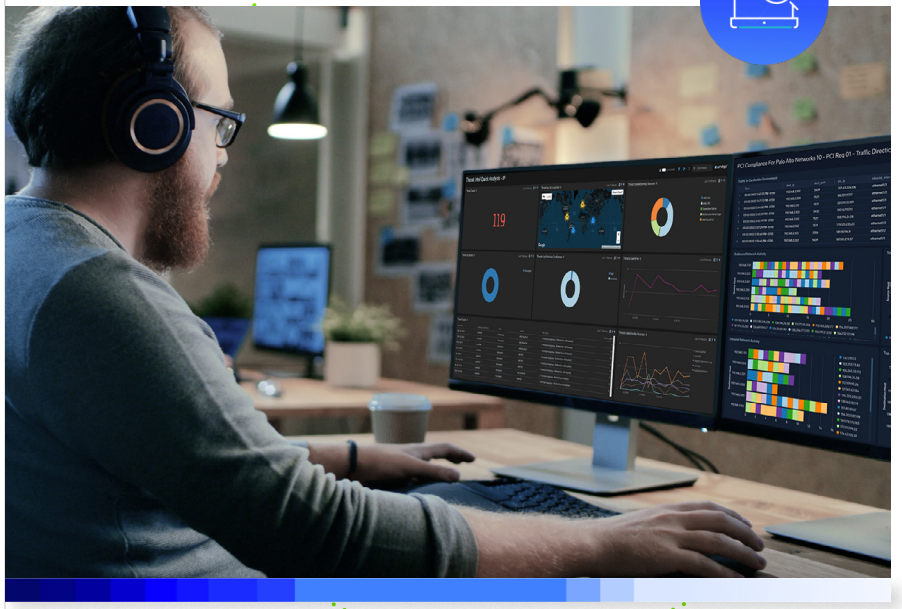
Ensure that your process for creating container images is reliably reproducible and verifiable.

Pulling forward the concept from the coding phase, only allow GPG-signed commits, thereby performing builds for commits from verified authors.

Automate your pipeline to collect metrics and artifacts, keeping the feedback loop short and consistent. For example, capturing and measuring build time will inform the overall time it takes to release a fix to production.

Generate and archive an SBOM for every build, providing insight into the supply chain. Use this alongside software composition analysis (SCA) tools like OWASP Dependency-Check to find vulnerable software in your dependencies.

4 The testing phase



You've integrated DevSecOps practices and tools earlier into your SDLC. You've secured your code, tightened up your build process, and maxed observability. The next step is to test.

One of the keys to DevSecOps in testing is automation. Running automated tests whenever possible creates a predictable and consistent testing practice. Run automated tests throughout the testing (and other) phases of the SDLC.

Testing tools

In addition to functional Dev/QA and non-functional performance testing, the following tests are important during this phase:

Unit Tests
Verify that critical code paths have coverage and are hardened against vulnerabilities discovered through SAST tooling and code linters.

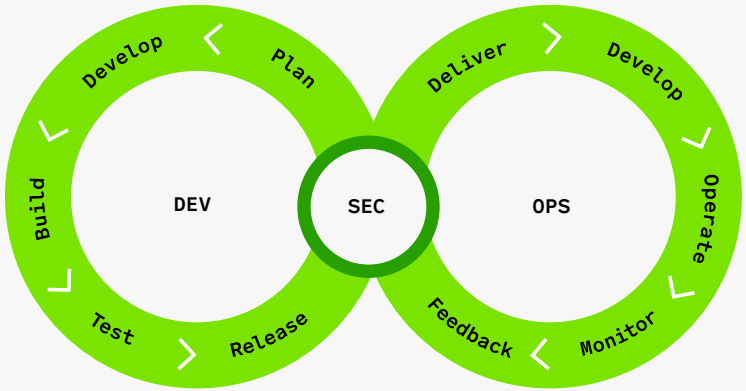
End-to-end Testing
Work in automated penetration testing that can then trigger additional code iterations to fix vulnerabilities. Validating deeper operational aspects like log emission and metric exposure can also happen at this phase.

Dynamic application security testing (DAST)
“Black box” tests that have no knowledge of the inner workings of the application and approach the application as a hacker would, simulating real-world attacks.

Acceptance Testing
Validate that the correct observability and security signals have been captured. This phase will usually take place in a staging or pre-production environment and is the time for activities that are difficult to do manually, such as custom penetration testing and operational or performance testing trigger additional code iterations to fix vulnerabilities. Validating deeper operational aspects like log emission and metric exposure can also happen at this phase.

5 The deploy and runtime phase

Once your application is live, you can't just forget about it. The operational/runtime phase is where you rely on instrumentation to identify any issues you missed. Humans can't write perfect code, and you'll never be aware of every possible problem. It's imperative to have tools that continuously scan and monitor for possible exploits or performance problems.



DevSecOps delivers better quality and more secure software

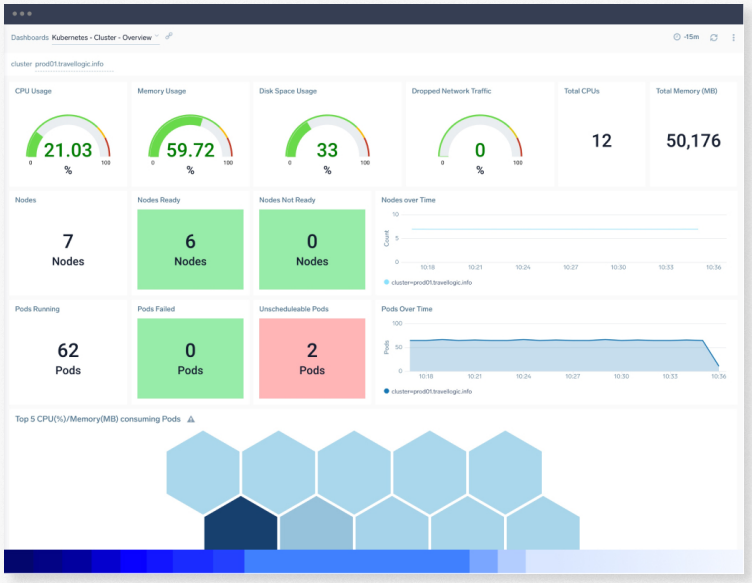
By this point, your DevSecOps planning, implementation, and testing metrics and artifacts will coalesce into a single pane of glass accessible to the development, issues and operations teams. This solution should provide ubiquitous data collection, easy analysis, alerting, and visualization. This culmination of your DevSecOps practices allows your security and operations teams to be not just reactive but proactive.

For example, you might use the Kubernetes integration with Sumo Logic, which can detect anomalous container activity, such as a misbehaving container reading a sensitive file on the filesystem.

This activity triggers a security alert rule that notifies your security team, who might trace the deployment back to a git commit with a configuration not captured by the test suite or QA team. The security team can then notify the development team to get a fix and gauge whether a deployment rollback is necessary. The team can decide whether to wait for the fix to deploy, basing their judgment on the captured build time and deployment time metrics.

This is one of many potential scenarios where development, security, and operations can have shared insight throughout the SDLC and work together to improve their security posture and operational efficiency using a single pane of glass.

Pulling forward the idea of reliability management, this phase will inform what the next cycle of the SDLC looks like. Burning through your error budget here will indicate that the next development cycle requires a focus on bug fixing and stability whereas a surplus of error budget could mean there's more room for innovation. Ultimately, the goal of the reliability management framework is to balance the speed of innovation with serving your customers.



The Kubernetes integration with Sumo Logic

What's next?

We've looked at the phases, tooling, and processes that embody the spirit of DevSecOps and how they can enable an SDLC that results in quick and secure innovation. DevSecOps comes down to integrating security and observability culture, processes and capabilities throughout your SDLC. By shifting security and observability concerns to your entire team — and at every step—your application will be more robust, cost-effective and secure.

Sumo Logic is an all-in-one observability and security platform that coalesces metrics, logging, and tracing into a single pane of glass.

It's designed to work with all of the tools you already use and provides additional utility such as the Sumo Logic OpenTelemetry Distribution to collect your data in a non-proprietary manner and deliver real-time insights.

Visit Sumo Logic to learn more about how our SaaS analytics platform helps global industry leaders deliver reliable and secure digital experiences.

About Sumo Logic

Sumo Logic, Inc. (NASDAQ: SUMO) empowers the people who power modern, digital business. Through its SaaS analytics platform, Sumo Logic enables customers to deliver reliable and secure cloud-native applications. The Sumo Logic Continuous Intelligence Platform™ helps practitioners and developers ensure application reliability, secure and protect against modern security threats, and gain insights into their cloud infrastructures. Customers around the world rely on Sumo Logic to get powerful real-time analytics and insights across observability and security solutions for their cloud-native applications. For more information, visit: SUMOLOGIC.COM

sumo logic