

SHIFTING SECURITY LEFT

The Real World Guide to DevSecOps

Contents

PART 1: Shift Left Methodology

INTRODUCTION

Buzzwords: A Danger in the Wrong Hands

The History of Shifting Left

The Container—Where It All Comes Together

- Continuity
- The Real Venue for DevSecOps

Why Shift Left

- Avoiding an Explosive Developer-Security Relationship

PART 2: How To Shift Left

1. Automate All Things

- Anchore's Automation Acid Test
- Workflow Integration

2. Best In Class Tools are Key

- Avoid the Ecosystem
- Container Scanning as a Tool Not a Function

3. Integrate Everything

- Shifting Legal Left

4. Shift Left Culture

- Creating the Venue

5. Challenging Process

- Using Anchore In the CI/CD Pipeline

6. Policy: The Vehicle for Taking Security Knowledge & Learning Left

- Policy as Code

7. Don't Stop Shifting Left When You Hit Production

- Admission Control

CONCLUSION

Introduction

Teams and organizations that embrace shifting more of their testing activities left within the software delivery cycle can outpace competitors that do not.

Shifting left is a genuinely transformative technique, which, when applied correctly, can lead to massive productivity gains that extend beyond the development team. It is a significant force multiplier, allowing organizations to be more productive. Its adherents find they are able to create higher quality, more secure products, while simultaneously speeding up their release cycles, innovating and responding faster to market demand. Equally important: shifting left can lead to a happier workforce with developer, security and operational teams able to collaborate without compromising their workflow.

The practice of shifting security left or DevSecOps can be especially powerful in the security arena. Up to now, the enterprise security team has often been viewed as its own entity – a progress prevention department, sitting off to the side and occasionally interceding at the last minute of a project, blocking a software release and causing chaos.

DevSecOps has started to challenge these myths. Supported by tools such as **Anchore** Enterprise, the security team can begin to collaborate with their colleagues in new and exciting ways. The DevSecOps movement is as important as the DevOps movement that preceded it, and it holds just as much potential.

Buzzwords: A Danger in the Wrong Hands

At the start of this type of paper, it is worth staring the elephant in the room straight in the eye and addressing it firmly. Both “shift left” and “DevSecOps” have already filtered into the hindbrain of marketing teams, and are being used in broad terms to describe everything from tools to techniques. They have started to become misused terms, and their real meaning is becoming diluted. In some narrow cases, shift left has been badly interpreted to mean “developers can do everything.”

It is easy to coin new terms like DevOps or shift left and misuse them. Often, this is simply a misunderstanding of what the terms mean. Under pressure from the board to implement the latest buzzword, an overzealous manager skim-reads whichever book is currently in vogue – suddenly developers find themselves in charge of hugely complex infrastructure because they are now DevOps and therefore domain experts.

This is the worst possible approach to shifting left: shifting responsibility and implied expertise to the left, without any support.

Shift Left Testing:

“By linking... functions at lower levels of management, you can expand your testing program while reducing manpower and equipment needs—sometimes by as much as an order of magnitude.”

Larry Smith, Dr. Dobbs’
Journal


September 1, 2001

When it comes to owning security, more than **25%** of developers feel solely responsible for security. Gitlab Survey, 2020

This type of approach has led to some disastrous attempts at DevOps transformation. Domain experts have been seen as surplus to requirements leaving developers to manage complex infrastructure, which they do not have the skills to operate. Not only does this end in spectacular failure, but it also has a real human cost of leaving burned out developers and talented operational teams out of work.

70% of DevOps engineers state that workload is coming at risk of burnout. [Logz.io Survey, 2017](#)

Shifting left is not about shifting responsibility and blame onto a single pair of overburdened shoulders. Instead, it is a mechanism to shorten feedback loops in the development process, and to make collaboration both pervasive and invisible: cooperation achieved by instinct rather than by ceremony and process.

 This paper explores how you can build an effective and collaborative shift left strategy while demonstrating the massive advantages gained by doing so. Shifting left successfully can make better use of existing team members and their real skills, allowing them to focus on their domain expertise with far fewer interruptions. This means increased productivity through more effective collaboration and less time wasted.

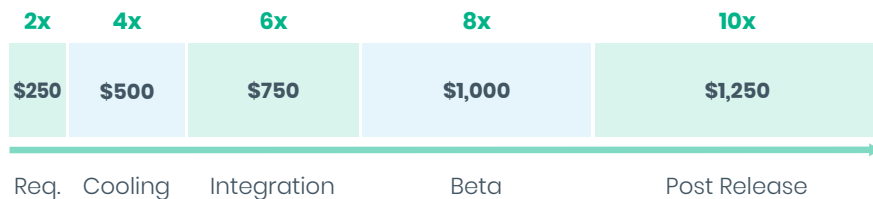
The History of Shifting Left

The term shift left was initially coined by Larry Smith back in 2001. Larry originally termed it “shift left testing” in an [article](#) published in Dr. Dobb’s Journal. Smith outlined a technique of shortening the software testing feedback loop so that developers would receive feedback early and often.

This was contrasted with the standard practice of throwing releases “over the fence” to the QA team while they waited for the testing to be performed. Instead, Smith encouraged developers and QA engineers to begin collaborating as close to initial development as possible. QA should become part of development, not the release process. This approach was somewhat revolutionary at the time and led to a renaissance in both testing practices and the tooling that supports it.

Modern QA professionals now spend a large proportion of their time writing automated testing and working with developers to improve their unit tests. Testing has become a habitual process of collaboration.

The ROI of the “Shift-Left” Philosophy: Cost of defect discovery by stage



Data sourced from *The Economic Impact Of Inadequate Infrastructure for Software Testing*. Assumes cost per defect of \$125.

Source

We now take it for granted that the CI (continuous integration) pipeline performs considerable testing work and returns results to the developer quickly, as part of a single smooth process.

Six years later, a systems administrator by the name of Patrick Debois was surfeited with resolving infighting among developers rather than doing the system administration he was employed to do. He began to give serious thought to what better collaboration between developers and operational teams could look like.

In 2008, as Debois was still pondering this collaboration, software developer Andrew Shafer gave a session on agile infrastructure at the Toronto Agile Conference. It was not a packed session, with precisely one attendee – Patrick Debois.

Sensing there was a swell of support for closer cooperation between developers and system administrators, Debois launched an event to flesh out the concept. Looking for a catchy title for the event he landed on the first “DevOps Day” and the rest is history.

DevOps popularized the idea of cooperation between different software development functions and encouraged developers to create new tools to support this. As with the first shift left movement, DevOps succeeded by blending good working practice with support tools. It demonstrated that success in pushing left is not about technical capability or organizational structures, but a blend of both.

Finally, a left shifting history wouldn't be complete, without a reference to AWS (Amazon Web Services). It could be argued that without the launch of AWS Elastic Compute Cloud in 2006, DevOps would have fizzled out and been another also-ran technique.

AWS shrank the launch time of new servers from hours, or even days, to seconds. More importantly, it offered a well-crafted API. And for many developers, this was the first time they were able to bring their skills to fruition on tricky infrastructure jobs. It made infrastructure a commodity.

Larry Smith's 20 year old observations on QA apply just as much to today's common processes for security testing:

“[It] is too often seen as overhead and not part of the development process. I've worked in shops where the relationship between the two sides was actually adversarial, as if it was part of [the] job to keep from shipping the product. That is simply wrong.

In fact, QA is as much a part of the development process as writing the code in the first place.”

The days when a handful of physical racks played host to some VMs (virtual machines) came to a crashing end. This was replaced with elastic, ephemeral madness, and new tooling was required to tame the madness. DevOps had arrived and it wasn't going anywhere.

Tools such as Puppet, Ansible, Terraform and Chef were developed, allowing infrastructure to be expressed as code. Developers and operations teams could now collaborate to define infrastructure through pull requests and merges rather than process and paperwork.

The Container—Where It All Comes Together

With all of this history, the greatest tool to emerge from the DevOps movement is the container. Containers are the culmination of the DevOps movement, putting immutable infrastructure in the developers' hands. This allows them to develop incredibly sophisticated and complex applications, publishing them straight from the desktop and making them as simple to run as a single Docker command. These same containers can then be consumed by the operational teams, allowing them to run software reliably on every platform, from a simple Linux server through to a massively scalable Kubernetes cluster. The same container runs the same way, regardless of the size of the infrastructure.

According to a recent survey of more than 500 respondents, “security ranks as the number one most difficult challenge to overcome when deploying containers.” [Portworx Survey, 2019](#)

The platform agnosticism of containers has combined with incredibly rapid growth in adoption to create profound changes in the enterprise software ecosystem. We once saw fierce battles as large proprietary vendors zealously guarded the boundaries of their enterprise platforms against open source upstarts. But with the container as a default, platform-neutral distribution mechanism there are no boundaries to protect.

Microsoft SQL can now be run with a single command on a Linux desktop. A few years ago telling a developer they could do this would have been met with very raised eyebrows, now it's the new normal.

And Microsoft is far from alone. Most vendors of server software either offer containers or advice on how to run their software in containers. They are rapidly becoming the conventional way to ship software, and vendors who do not provide container capabilities will quickly find themselves losing market share to those who do.

Containers have completely re-shaped how we develop and deploy software. For the first time, we have a mechanism that can scale the same software from a developer's desktop through to a massive cluster.

Continuity

From a left shifting perspective, containers have delivered beautifully on the promise of DevOps. They enable systems administrators to define parent containers, creating a consistent infrastructure environment that spans from the start of planning and development right through to production.

Operations teams can bake in OS level best practice from the start of the development process. The same defined development container can then be deployed into test environments and, finally, into production — one container traveling through the software lifecycle.

Containers have brought their own scheduling, now mainly in the form of Kubernetes. This allows containers to be deployed into fully elastic, cloud-agnostic, self-healing infrastructure. Kubernetes itself is then managed using a well documented and powerful API, allowing developers and operators alike to develop custom software to manage it.

The Real Venue for DevSecOps

Containers have opened the door for another three letters to join the DevOps party. DevSecOps is possible when security is shifted left, increasing the collaboration among the Security, Developer and Operations teams. And the container is the venue where this new, bigger party is happening.

DevSecOps is not just about people in different IT functions moving their desks closer together, sharing joint daily standups around the same coffee machine, or even mixing up who's on whose side at company team building events. Some of these things may help, but it is questionable if they aid DevSecOps culture, or whether they result from it.

On a scale of one to five ranking *least stressed to most stressed*, more than a third (34%) of respondents ranked themselves at four to five. [Logzio Survey, 2017](#)

The real collaboration is happening in lines of code, pull requests and automation, all nestled at the heart of the container. Using containers as the collaboration point for development, security teams can evaluate both the code and the infrastructure that runs it entirely in its production state, right from the start of the development process.

By offering a standardized and simple way to audit artifacts, containers provide the ideal site for tools such as [Anchore](#) to perform detailed auditing of the contents of the container, exposing CVEs (common vulnerabilities and exposures) early in the development process. In this way, DevSecOps is just one more group coming to the container: the natural progression of shifting left and the DevOps movement.

Why Shift Left

It is worth looking in detail into what shifting security left in a DevSecOps workflow can offer your organization.

Security teams have often been seen as shadowy figures who occasionally descend into the developers' bullpens as harbingers of chaos and irritating delays. Typically, they come bearing doom, despair and problems: pointing out security issues that need to be rectified late in the development cycle.

There is even some truth in this stereotype, but often the security teams' more irritating behaviors are not the result of choice. In most cases, they have had to wait until the release is in pre-production to assess the security issues. With early-stage variations in software and infrastructure, pre-production typically offers the only environment close enough to represent what is being deployed as a whole.

This limitation on where and when the security team can begin any meaningful audit means that security issues can only be discovered late in the software cycle. They become an immovable, late-stage roadblock on what is otherwise a fast DevOps highway.

Twice As Expensive – Cost association when finding and fixing non-severe software defects “after” delivery vs finding and fixing defects “before” delivery.

100x More Expensive – Cost association when finding and fixing during the requirements and design phase.

Systems Engineering Research Center Survey, 2019

Security blockages can then necessitate another full development cycle and a significant sprint of work to rectify. This process is inherently inefficient and demotivates everyone, from developers to security teams, and especially product owners watching their deadlines slip.

In this regard, software development is unique within manufacturing. The creators are leaving entire processes until after the product has been designed, built and sent for testing. If an executive in the automotive industry found that their new flagship SUV had been delayed because someone had forgotten to check the door locks until the first unit had rolled off the production line, you can imagine that questions might be asked.

And yet, in software, we routinely wait until the end of the development process to locate security issues, and then send it right back onto the factory floor to be fixed. In some cases, multiple times.

Avoiding an Explosive Developer–Security Relationship

There are no absolutes in IT security. However, by adopting an iterative and collaborative approach—shifting left—SecOps teams are able to raise security levels and achieve more over time. This incremental approach is far preferable to any “Big Bang” security strategy which as its name suggests, tends to make a lot of noise and create havoc within a large blast radius.

Big Bang implementations of security are doomed to failure, putting too much onus on the SecOps teams to be the gatekeepers of unrealistic change, and creating a trail of bewildered developers who find their existing workflows are blown out of the water.

Shifting security left iteratively by using policies, allows more than just managed change; it also allows the SecOps teams to inculcate a security thought process within their colleagues.

Double the Confidence – When security is integrated into the SDLC teams have stronger assurance about security posture.

[Puppet, Circle CI & Splunk State of DevOps Report, 2019](#)

This change in thought process is often overlooked but is one of the most significant benefits that shifting left can bring. In this sense, shifting security left is akin to positive habit-forming, creating a corporate routine that instinctively bakes good practice into everything that is produced.

Pushing thought processes and habits left ensures a genuine team effort, rather than just a single team, or even one engineer thinking about security. It means that right from inception, software and operational engineers are focused on security hygiene asking themselves the right questions from the start of the process.

Shifting left is a true blend of practice and people skills, and how you approach it will depend on your company culture and your technical needs.

The following seven steps offer a good place to start. These steps combine actionable elements with hints and tips and can form the basis for your own approach to shifting left. We have found that these steps are present in any successful implementation of a shift left methodology and, although no guarantee, they will help any shift left strategy to succeed.

1. Automate All Things

Automation is at the core of a shift left strategy.

Without automation, shifting left simply becomes a weaponized form of passing the buck from person to person.

Automation opens many new doors, often leading in unexpected directions. However, it is vital for automation to extend and enhance existing workflows, not add new barriers.

Anchore's Automation Acid Test

Consider the following when looking at each element to automate:

1. **Is this automation an integration with existing tooling, or a new tool that practitioners need to learn and use out-of-band with existing processes?**
2. **Is the automation shortening the feedback loop between engineers?**
3. **Are engineers able to continue work while waiting for feedback, or does it block their workflow until it has completed its actions?**
4. **Does the new automation offer actionable intelligence for the recipient engineer?**

If the answer to any of the above is no, it's likely that the automation being considered may not add much to the shift left strategy, and may, in some cases, actively work against it.

Generally, an excellent place to start is within the unit testing of code; this is where an enormous amount of feedback happens for software engineers, and is an excellent place to insert additional checks that practitioners such as platform or security engineers would typically perform themselves. Often, these checks are in the form of linting or scanning to check, for example, that code has a README.md, or that a Dockerfile is appropriately formatted.

Linting is typically fast, cheap and easy to integrate into unit tests. This should be done in the developers' IDE (integrated developer environment) using plugins or, at the very least, at check-in time using the CI/CD pipelines.

Scanning, on the other hand, is often more heavyweight, and should ideally take place in the CI/CD pipeline. This will provide fast and actionable feedback to the developer, and minimize frustrations from scans interrupting their core workflow.

An effective DevSecOps toolchain transforms elements such as security scanning and license audits from manual tasks to fully automated ones.

Workflow Integration

Anchor is a powerful keystone in a container scanning security landscape. However, it cannot sit in splendid isolation. **Anchor** shines at its brightest when its full API and CLI capabilities are utilized to integrate into existing workflows. This integration often starts with the build pipelines but spreads from there.

Anchor is ideal for placing inside the CI/CD pipeline to offer developers fast feedback on potential container security issues. It allows developers to receive actionable reports on security issues right alongside any other automated testing that might already be present. In addition, it enables security engineers to seamlessly integrate security and licensing policies into the workflow without requiring software or platform engineers to be aware of either the policies or any changes to them. This approach is in contrast to "fire and forget" container scanning that can be run in an ad hoc fashion on the desktop, or as a broad, hard to customize and overly noisy CI/CD tool.

2. Best In Class Tools are Key

Automation is at its best when it is assembled out of best in class parts and introduced iteratively.

Avoid the Ecosystem

Within IT security, there is an enormous temptation to buy into comprehensive suites or ecosystems. With a security solution, one purchase order or credit card payment and you solve a whole host of problems – or so the story goes.

Many vendors now offer a complete solution, covering every possible angle of security. And increasingly, these security ecosystems have terms like shift left and DevSecOps front and center in their marketing. But despite their claims, there is no one suite or ecosystem that you can drop into a team that magically implements your shift left strategy.

To achieve such a broad umbrella of functionality, security ecosystems have to bring strong opinions to the table. Views that extend beyond the tools in the suite to dictate workflow, process, policy and practice.

DevSecOps is about promoting a new more collaborative culture between Developers, Operations and Security. Security teams walking in and dictating tools, workflow, process, policy and practice to the other two groups is unlikely to provide their best starting point.

Adopting a highly prescriptive security suite can create months of painful resistance to change as every team tries to salvage precious elements of their former workflow and treasured working practices from being subjugated by the new ecosystem.

No suite of tools is ever going to shift your team to the left by forcing them to follow its process over their own. Every engineer, whether they are in Operations, Security or Development, is attached to their existing tooling and workflows. In many cases, they will have spent years honing both.

In this case, resistance to change is more than just an emotional reaction. It is a well-reasoned aversion to arbitrarily changing out tools for no benefit other than fitting in with a particular vendor's narrow vision of what DevSecOps should look like.

Selecting the right tools is vital to support any successful shift to the left, especially where your organization currently has little or no automation in place. The preferred alternative to security ecosystems or suites is for organizations to build a custom security toolbox or toolchain, combining best in class tools to fit their unique needs.

In contrast to the suite approach, most specialist tools are built to integrate. For effective DevSecOps, the goal should be to draw insight from the best available security tools and shift this insight left, back into the existing developer toolchain.

Although it hasn't always been the case, there is now a plethora of powerful specialist tooling available in the security arena. There are also a growing number of open source options that can simplify integration and which allow security teams to freely evaluate and compare tools on merit.

In building a custom security toolchain, security teams are able to adopt a more targeted approach focused on quick wins, rather than becoming bogged down in implementing a single DevSecOps solution. Teams should adopt a solution-focused approach, viewing DevSecOps as an ongoing methodology for addressing security challenges, rather than one overwhelming solution.

It is best to identify a problem, determine the best means to solve that problem and then shift left by integrating that tool into the existing workflow.

When considering tooling, use the following checklist to help guide your decisions:

- **Is the tool focused on a single element (container scanning, artifact storage), or does it offer a broad set of added extras?**
- **Does the tool have a proven record of integrating with the favored tools/systems your engineers use?**
- **Have you approached practitioners to check what experience they have with the tool?**
- **Does the tool have wide mindshare outside of your organization?**

If the answer to any of the questions above is no, it should be a warning flag that this tool or ecosystem may not be a good fit, and may actively hinder your left shifting ambitions.

Container Scanning as a Tool Not a Function

Many vendors who offer container repositories now offer container scanning at minimal, or even zero, additional cost. These tools do an OK job of discovering CVE's, albeit, generally from a small list of security feeds. However, by the time the information surfaces, it's usually a long way from where the developer first checked in their code. Information this late-in-the-day destroys the concept of fast feedback loops and generally hinders pushing container security left.

The alternative is to use **Anchor**; integrating it into the CI tools that the developers are actively using. The CI pipeline is where developers expect to receive feedback about the build, and information that comes from other mechanisms tends to either be ignored or observed without merit. Developers will not deliberately ignore the information, but if it is not in their usual feedback channels, it is less visible and less likely to be acted upon.

Anchor can integrate with virtually any CI tool, either using its API or CLI. Either of these options offers 100% functionality, allowing any aspect of **Anchor** to be driven by the CI tool. In turn, the information it produces can also be integrated into the CI pipeline, giving developers that all-important single place to view not only functional issues such as failing unit tests, but also non-functional elements such as security. Without changing their workflow, the security team can bring a wealth of security-related issues to the developers' attention. The security team has shifted a massive amount of actionable information left.

3. Integrate Everything

Any form of shifting left should be viewed as integrating specialist knowledge and insight left into the existing development pipeline, rather than shifting additional tasks to the left onto the developer. And the responsibility is on the team shifting their priorities left to do this as unobtrusively as possible. If developers are required to add new steps to their existing workflow and to learn and use new DevSecOps tools, something has gone wrong.

Shifting left must be about incorporating additional considerations into an existing process and not adding in a new step. The aim is to seamlessly integrate the output of security tools into the developers' existing toolchain.

For instance, security teams may find it advantageous to add license scanning integration for their colleagues in the legal team to divest themselves of a tedious manual job sending over a report. The operations

team may wish to integrate the admission controller into their infrastructure to add more surety around running containers. And finally, the developers may want to extend their development tools to integrate **Anchore** to get even faster feedback than the CI pipeline can give.

Integration is often seen as an expensive luxury, only performed when it's imperative. For the most part, this is based on somewhat out-of-date views of the current state of tooling. In the past, many tools used their own data format and lacked a clear, usable API. This is no longer the case.

Modern tooling is designed more often than not with integration in mind, either via a command line suitable for scripting, or an API, often using either SOAP, REST or GraphQL. In tandem, documentation tends to be good, with enough to make integration relatively straightforward. If you are in an organization that is considering a shift left strategy, you probably already have all the skills required to integrate tooling.

One common objection to integrating tooling is time. Integration of internal tooling is often not considered to be a core competency, and developers are instructed to focus on delivering features for the company's own product. This can be a very false economy. Integrating tools is often straightforward, with some simple glue code required to link the two together. The integration, once delivered, can save significantly more time than it takes to develop, and can often reveal deficiencies that the non-integrated procedures missed.

By integrating the best in class tools, you are creating a platform for your use with your assumptions and practices built in. A well-crafted internal platform can massively simplify a shift left strategy, removing friction from collaboration, and not forcing highly-skilled contributors to contort and compromise processes to fit with a single vendor's narrow vision.

Shifting Legal Left

Software licensing is a complex area with 107 approved open source licenses currently in use. The sheer number and complexity of licenses makes it unreasonable for the majority of software developers to understand the nuance between GPLv1, GPLv2, LGPL and others. However, these licenses have a profound legal effect on the delivery of software, and incorporating the wrong license in a product can have disastrous consequences if it is accidentally included and released into production.

Using **Anchore** it is relatively trivial to address this challenge. **Anchore** scans the complete container image and examines the licenses in use within the container bill of materials (BOM). With full automation coverage from both a CLI and API, **Anchore** is designed to be able to integrate with virtually any tool. This means

that it can be used in conjunction with a pre-release pipeline to update a spreadsheet automatically, feed an existing auditing system or just produce an email to a suitable stakeholder.

This allows the technical teams to integrate with the legal team and provide timely information on potential license issues without requiring developers to create a report manually.

It would even be possible for teams such as legal or compliance both to view licenses in use, and to collaborate with the security team to block licenses that may not fit with the business. By codifying these license policies they can then be passed transparently back to developers. From this point on, developers would receive warning within their build pipeline if they include a library that does not meet license standards.

4. Shift Left Culture

Even the most effective automation will not deliver a successful shift left strategy on its own. Developing a shift left culture is both an essential part and, importantly, a desired outcome of any change.

Unity is a crucial part of any successful shift left strategy. It is important that everyone involved listens, acknowledges each others' concerns and collaborates, both technically and culturally. If only one group in an organization shifts left and the others don't, you are not technically shifting left; you're doing the splits. And much like doing the splits in person, it can put a strain on unusual places in your organization.

An essential part is in recognizing that shifting left is a continuous process—a culture rather than an end goal. It is not something that can be delivered by new IT solutions or outside consultancies on an ambitious and arbitrary timeline.

All participants should feel confident that it's OK to evaluate and change. If teams are concerned that stopping to consider issues may have an impact on their timeline, this instantly dissuades people from trying to fix processes, and encourages adoption of an unhealthy *make do and mend* approach.

Those leading the process must foster a mindset of shifting left and encourage everyone no matter which specialism they work in to both think of how they can collaborate better with others, and to be open to suggestions from others. Once the mantra of innovate, listen and accept is in place, then you can start to pick specific areas to focus on.

When gathering feedback on your shift left strategy, the following list of actions can be useful to keep in mind:

- **Set out a shift left vision, but not a deadline. Shifting left is a practice, not a destination.**
- **Ensure you make time to shift left. Shifting left involves a lot of conversation and understanding, it's not something that can be ad-hoc.**
- **Make space for feedback without allowing it to be personal. Ensure people are free to give honest feedback but make sure it's constructive.**
- **Identify quick wins. Run workshops to quickly identify mirrored concerns and automate to fix it.**
- **Feedback fast, feedback often. Make shifting left iterative and not a flash in the pan. Trial a change, gain feedback and refine.**

Finally, don't be afraid to fail. Shifting left can mean replacing and automating things that may have been present for years, if not decades. Sometimes this fails, either because it's more complicated than first thought, or because it was not understood at the right level. Rather than keep a broken new process, don't be afraid to throw it away and try a new approach. Learn, adapt, test, feedback and learn again.

Creating the Venue

It is essential to provide a concrete venue for practitioners to collaborate on how best to shift left.

Existing rituals such as a retro will provide the best setting for practitioners to challenge processes. This is a space that is already set aside to critique and improve how work is approached and is a good venue for allowing people to question the value of different processes. However, it is crucial that this is constructive critique and not a session to vent.

What is more tricky is where little or no existing relationships exist. This is often the case between security and software engineering. In many companies, these are two departments that rarely meet, and when they do, it's usually to mitigate disasters such as data breaches.

As part of the shift left strategy, you need to find where teams are dependent on each other but not communicating. Typically, the lack of communication will be time-based; the usual "I'd love to meet my colleagues, but I've got a million things to do" response.

The first step is for leadership to ensure that time is blocked out for the departments to meet each other, rather than relying on staff to make the time. This can be a simple brown bag lunch or another type of session where both teams can show what makes them unique.

Sessions should be blame-free where both sides can put down what they like about the other team and what they struggle with. This can be as simple as having a whiteboard and post-it notes, and a column of like/not like. In the end, you will generally find that there is a lot of crossover between them, with issues that can be solved using automation.

5. Challenging Process

Once you have gathered the data, you can evaluate as a team whether it is worth automating the process or, at least, making it more lightweight. You can also ensure that everyone on the team is OK with the process being changed. Once you agree, you can develop some suggestions for implementation and offer it to the other stakeholders involved.

Those suggesting change must have empathy for the other people involved in the process. It is vital to offer up suggestions as positive change and to not put people on the defensive. The conversation should ideally be approached from the position of “we think this could help us all work together...” rather than “this process is causing us real pain.”

When gathering questions around the process, the following list can be helpful to focus thinking:

- **Is what we are considering actually a regular, repeated process: does its frequency warrant attention?**
- **How disruptive is the current process? Does it affect an entire team for five minutes per day or a single developer for 10 minutes per week?**
- **Are existing tools in place that would allow for this process to be automated, or would new systems need to be created?**
- **Is there a hidden requirement for this process via certification or regulation? If possible, have a stakeholder present who might know its inception.**
- **Is everyone involved in the process open to change? If not, has the person proposing the change been able to find out why the others are opposed? It's worth asking in a non-judgmental and open way why that is.**

Often, these areas will immediately suggest themselves; for instance, platform and security engineers already tend to have a working relationship. Where these working relationships are already in place, you can encourage further growth, potentially by running workshops exploring how existing collaboration works, and suggesting new tools and practices that can be integrated. These workshops can be surprisingly practical at discovering how to improve collaboration and can take no more than an hour to start.

At the end, you will find that you have a list of potential tool integrations and approaches that can be placed into a workstream. It's important that these sessions are regular; perhaps once a month depending on the size of your team. This allows the teams to continue their focused shift left discussion and to change direction if something isn't working. Fast feedback without simply abandoning the effort.

Using Anchore In the CI/CD Pipeline

Looking in more detail at our example, you will often find the following issues raised by software engineers about their colleagues in security:

“They don’t tell us about issues until it’s too late!”

“We have to do a ton of manual spreadsheets for them.”

“The policy documents for what we can and can’t use are super confusing and not updated.”

And on the flip side, you’ll often find issues like:

“Developers aren’t paying attention to what dependencies they use.”

“We’re not told about major changes to the software until it’s almost delivered.”

“We’re not involved early enough in the process.”

When you look at the list above, you can see they are mirroring each other. Sadly, this is all too common. But on the brighter side, it’s relatively easy to solve using automation. In the above example, judicious use of **Anchore**, as well as other automated security scanning tools, would do a considerable amount to alleviate the issue.

In the case above, we could start by placing **Anchore** into the CI/CD pipeline in a permissive mode that reports on issues without blocking further development. Software and security

engineers are both made immediately aware of what is required of them and of any potential upstream issues. Again, once you have implemented this step you need to step back and let it run for a trial period and then get feedback from both parties. Once they are happy, you can add further integrations, for example Github security for further scanning or **Anchore** admission controller, to loop in the ops team and more.

6. Policy: The Vehicle for Taking Security Knowledge & Learning Left

The best policies are applied without anyone even being aware that they are following policy.

Good policy is the bedrock of a swath of technology functions across Security, Operations and Development. However, even the most well-written policy is usually complex and it rarely makes compelling reading.

Simply relying on practitioners to memorize the various policies required and to continually refresh their knowledge is not an effective solution. Policy becomes a box-ticking exercise inflicted on bewildered new starters and then left abandoned by the wayside. Even with the use of mandatory and recurring web-based training, there is a tendency for this to devolve into a game of chance, with examinees safe in the knowledge that if the test is failed it can be retaken.

The solution lies in expressing policy as code and integrating this into the development pipeline as automation.

Policy as Code

Shifting security left has the capability to make developers allies of security, giving them the information they need as early as possible within the development feedback loop, and allowing them to identify issues early and rectify them before the code leaves development.

Anchore allows this collaboration to be expressed in the form of security policies as code. Policies enable security teams to stop

chasing individual CVE's and instead set policies for what level of CVE is allowable in code. This policy, in turn, is pushed to the **Anchore** Engine running inside the CI toolchain, giving feedback to the developers right where they expect it—inside their existing tooling. No new processes, no new places to log on and see security issues. Policies allow for frictionless collaboration between developers and security teams without disrupting existing workflows.

By coding and automating policy it becomes an invisible, ever-present guard rail for practitioners. They can be made aware of the policy in general terms, and then allow the domain experts to quietly update and push new versions of the policy as code into the automation. By taking this approach, policy becomes less fiction and more fact, and adherence is not only mandatory, but a given. It becomes almost impossible not to adhere to the policy because the policy is taking care of itself.

Automation and coded policies massively increase productivity in addition to compliance. Developers no longer have to take up headspace and memory on which procedure to apply. Instead, they can concentrate on the job in hand, safe in the knowledge that if they do something that is not allowed by policy they will have automated feedback that is fast, actionable and relevant. This is in marked contrast to a manual process, where issues can make it to the end of the process before a missed procedure is spotted, which in some cases, can push a piece of work right back to the starting line.

For security teams, policy as code allows them to iterate over time rather than having to introduce disruptive changes into development workflows. Instead, they can first adopt a policy of scanning and reporting, allowing them to judge how much remediation work is required. Next, they can feed this data to the developers, straight into their existing workflow, albeit still in a warning posture. Over time, the capabilities and breadth of the policy can change to increase security, always in an iterative fashion, and always with an eye to the impact it has, not only on the security stance but also on the disruption it is causing for developers and production deadlines.

7. Don't Stop Shifting Left When You Hit Production

Shift left strategies often focus on improving developer workflow, but stop when it comes to production.

It may seem strange to advocate shifting left even when you hit production – which is by definition the far right of the software development cycle. However, extending your push left strategy right through to production can bring enormous benefits. For developers, it means easier and safer deployments; for operators, greater visibility of what is deployed; and for security engineers, the surety that what is deployed has been assessed against the current security stance.

At its most basic, pushing left into production means allowing developers and platform administrators to collaborate. This is now easier than ever before with tools such as Terraform, Ansible or Pulumi allowing platform engineers to create, collaborate and improve using the shared lingua franca of code. Teams can use a source code management tool such as Git to issue pull requests if they see improvements, or simply to allow interested parties to see how the infrastructure is created. Developers can amend infrastructure to accommodate new features, and collaborate early and often with the domain expertise in the platform engineering team.

It also gives the platform engineering team insight into what is coming down the pipeline, and allows them to start planning for any new capacity or monitoring that might be needed. This approach is now broadly accepted within the remit of DevOps, and many organizations have been using this form of collaboration for some time.

Containers have allowed this approach to evolve, making it vastly simpler for the security engineers to be involved in a shift left into production strategy. They have enabled new levels of automation to be applied to both securing the platform, and controlling the artifacts that are being deployed onto it.

Security engineers are now able to shift left to the platform engineers. Automation has removed the need for platform engineers to manually audit containers running on systems such as Kubernetes and report this information to security. Moreover, security engineers can automatically apply policy to ensure that containers that do not meet certain criteria are not even allowed to run on certain platforms.

Admission Control

Anchore has developed an ingress controller for Kubernetes to help create more effective shift left workflows. Using the **Anchore** admission controller together with **Anchore's** support for policy as code, allows security engineers to produce secure Kubernetes policy centrally. This is then applied anywhere in the organization where the ingress controller is present.

These policies can be tuned for the environment at hand. New code implemented in development environments can be audited but left permissive to allow for draft development iterations without blocking progress. Whereas production environments can be made more restrictive with non-permissible code blocked for the optimum security stance.

This is entirely automated from the point of view of the platform engineers, allowing them to focus on the areas where they add value without getting bogged down with manual procedures.

When considering shifting left into production, you should keep the following things in mind:

- **Always keep the full software lifecycle in mind. Automate all the way to production, not just inside the development workflow.**
- **Consider shifting left from SecOps into PlatformOps as part of shift left; platform engineers are as relevant as developers for certain workflows.**
- **Integrate the development and production workflows policy as code to retain a consistent workflow.**

Next, the **Anchore** admissions controller allows the security team to shift left into the operations team. By using the admissions controller in increasing levels of enforcement as containers journey through the development lifecycle, the security team can give the platform operators a new and valuable insight into what is running on their platforms. In turn, these insights can be integrated into existing operational tooling, allowing operators to be warned about security issues or unexpected container loads that have attempted to launch on a cluster they manage. Again, this integration brings a fundamental shift in security visibility without requiring the platform operators to learn and use a new tool.



Conclusion

Shifting left is becoming the next DevOps – a working practice inspired by and promulgated by developers. Increasingly, it will be seen as the de facto approach to software development, mirroring DevOps before it. Security is an ideal early candidate for shift left methodology. There is a confluence of tooling, appetite and mindshare that is pushing security into the fray and the term DevSecOps is going to be prominent, not only as a buzzword but also as a rapidly growing practice over the next 12 months.

To shift left, organizations must look first to culture rather than technology, with tools offering support to aid the cultural shift. There is no magic bullet when it comes to tools and technology. However, one-size-fits-all-solutions will quickly show their limitations, constraining developers to work within the workflow of the tool rather than allowing developers to enrich existing workflows. In this regard, it is crucial that any tooling selected to help shift left enables developers to retain their preferred workflows, but adds security information inside a tight development feedback loop.

A large part of the shift left strategy is to build up strong security habits within all teams, not just the SecOps personnel. To do this, tooling can be combined, integrating best in class security tools such as **Anchore** into the existing developer toolchain. This methodology helps keep developers abreast of emerging threats in near real-time, warning them within the development cycle as security issues arise. Equally, using policy-based security allows the SecOps team to push security iteratively, avoiding disruption, and hardening the organization's security stance in a transparent and frictionless manner.

By shifting left with a set of well-integrated, best in class tools, organizations can transform often conflicting interests to simultaneously harden their security stance and increase developer productivity.

About Anchore

Based out of Santa Barbara, California and Northern Virginia, **Anchore** provides a set of tools that provide visibility, transparency, and control of your container environment. **Anchore** aims to secure container workloads at scale without impacting deployment velocity. Our **Anchore** Professional Services team helps users leverage **Anchore** to analyze, inspect, scan, and apply custom policies to container images within custom CI/CD pipelines.

anchore

✉ info@anchore.com

🌐 anchore.com