Getting started with DevSecOps

The open source guide to DevOps security

ABOUT OPENSOURCE.COM

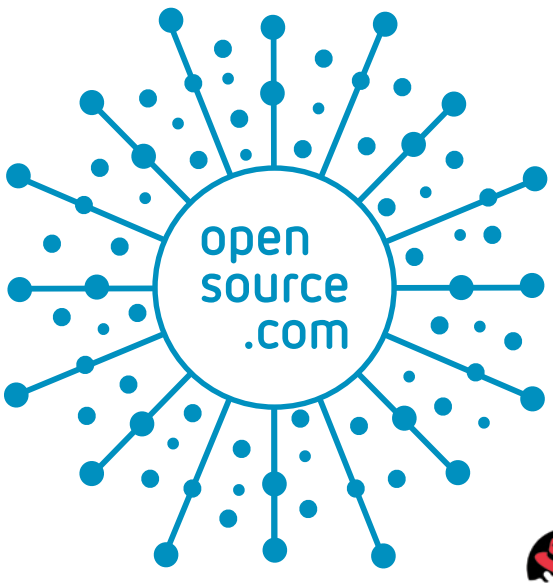
What is Opensource.com?

OPENSOURCE.COM publishes stories about creating, adopting, and sharing open source solutions. Visit Opensource.com to learn more about how the open source way is improving technologies, education, business, government, health, law, entertainment, humanitarian efforts, and more.

Submit a story idea: https://opensource.com/story

Email us: open@opensource.com

Chat with us in Freenode IRC: #opensource.com



SUPPORTED BY RED HAT

INTRODUCTION

CHAPTERS

GET INVOLVED | ADDITIONAL RESOURCES

What is **DevSecOps?**

BY BRETT HUNOLDT AND AARON RINEHART

The journey to DevSecOps begins with empowerment, enablement, and education. Here's how to get started.

DEVSECOPS as a practice or an art form is an evolution on the concept of DevOps. To better understand DevSecOps, you should first have an understanding of what DevOps means.

DevOps was born from merging the practices of development and operations, removing the silos, aligning the focus, and improving efficiency and performance of both the teams and the product. A new synergy was formed, with DevOps focused on building products and services that are easy to maintain and that automate typical operations functions.

Security is a common silo in many organizations. Security's core focus is protecting the organization, and sometimes this means creating barriers or policies that slow down the execution of new services or products to ensure that everything is well understood and done safely and that nothing introduces unnecessary risk to the organization.

Because of the distinct nature of the security silo and the friction it can introduce, development and operations sometimes bypass or work around security to meet their objectives. At some firms, the silo creates an expectation that security is entirely the responsibility of the security team and it is up to them to figure out what security defects or issues may be introduced as a result of a product.

DevSecOps looks at merging the security discipline within DevOps. By enhancing or building security into the developer and/or operational role, or including a security role within the product engineering team, security naturally finds itself in the product by design.

This allows companies to release new products and updates more quickly and with full confidence that security is embedded into the product.

> "DevSecOps enables organizations to deliver inherently secure software at DevOps speed."
>
> -Stefan Streichsbier

Where does rugged software fit into DevSecOps?

Building rugged software is more an aspect of the DevOps culture than a distinct practice, and it complements and enhances a DevSecOps practice. Think of a rugged product as something that has been battle-hardened through experimentation or experience.

It's important to note that rugged software is not necessarily 100% secure (although it may have been at some point in time). However, it has been designed to handle most of what is thrown at it.

The key tenets of a rugged software practice are fostering competition, experimentation, controlled failure, and cooperation.

How do you get started in DevSecOps?

Gettings started with DevSecOps involves shifting security requirements and execution to the earliest possible stage in the development process. It ultimately creates a shift in culture where security becomes everyone's responsibility, not only the security team's.

You may have heard teams talking about a "shift left." If you flatten the development pipeline into a horizontal line to include the key stages of the product evolution—from initiation to design, building, testing, and finally to operating—the goal of a security is to be involved as early as possible. This allows the risks to be better evaluated, socialized, and mitigated by design. The "shift-left" mentality is about moving this engagement far left in this pipeline.

This journey begins with three key elements:

- empowerment
- enablement
- education

Empowerment, in my view, is about releasing control and allowing teams to make independent decisions without fear of failure or repercussion (within reason). The only caveat in this process is that information is critical to making informed decisions (more on that below).

To achieve empowerment, business and executive support (which can be created through internal sales, presentations, and establishing metrics to show the return on this investment) is critical to break down the historic barriers and siloed teams. Integrating security into the development and operations teams and increasing both communication and transparency can help you begin the journey to DevSecOps.

This integration and mobilization allows teams to focus on a single outcome: Building a product for which they share responsibility and collaborate on development and security in a reliable way. This will take you most of the way towards empowerment. It places the shared responsibility for the product with the teams building it and ensures that any part of the product can be taken apart and maintain its security.

Enablement involves placing the right tools and resources in the hands of the teams. It's about creating a culture of knowledge-sharing through forums, wikis, and informal gatherings.

Creating a culture that focuses on automation and the concept that repetitive tasks should be coded will likely reduce operational overhead and strengthen security. This scenario is about more than providing knowledge; it is about making this knowledge highly accessible through multiple channels and mediums (which are enabled through tools) so that it can be consumed and shared in whatever way teams or individuals prefer. One medium might work best when team members are coding and another when they are on the road. Make the tools accessible and simple and let the team play with them.

Finally, and perhaps most importantly, DevSecOps is about training and awareness building.

Different DevSecOp teams will have different preferences, so allow them to be independent whenever possible. This is a delicate balancing exercise because you do want economies of scale and the ability to share among products. Collaboration and involvement in the selection and renewal of these tools will help lower the barriers of adoption.

Finally, and perhaps most importantly, DevSecOps is about training and awareness building. Meetups, social gatherings, or formal presentations within the organization are great ways for peers to teach and share their learnings. Sometimes these highlight shared challenges, concerns, or risks others may not have considered. Sharing and teaching are also effective ways to learn and to mentor teams.

In my experience, each organization's culture is unique, so you can't take a "one-size-fits-all" approach. Reach out to your teams and find out what tools they want to use. Test different forums and gatherings and see what works best for your culture. Seek feedback and ask the teams what is working, what they like, and why. Adapt and learn, be positive, and never stop trying, and you'll almost always succeed.

Authors

Brett Hunoldt – Technologist, Security and Privacy Advocate, Parent, Gamer & Explorer.

Aaron Rinehart – DevSecOps, Security+Chaos Engineering=ChaoSlingr, Entrepreneur, RuggedSoftware, Innovation Catalyst @UnitedHealthGrp.

Talking to normal people about security

BY MIKE BURSELL

Normal people generally just want things to work.

MOST PEOPLE[1] don't realise quite how much fun security is, or exactly how sexy security expertise makes you to other people.[2] *We* know that it's engrossing, engaging, and cool, *they* don't. For this reason, when security people go to the other people (let's just call them "normal people" for the purposes of this article), and tell them that they're doing something wrong, and that they can't launch their product, or deploy their application, or that they must stop taking sales orders immediately and probably for the next couple of days until this is fixed, then those normal people don't always react with the levels of gratefulness that we feel is appropriate.

Sometimes, in fact, they will exhibit negative responses—even quite *personal* negative responses—to these suggestions.

The problem is this: security folks know how things *should* be, and that's secure. They've taken the training, they've attended the sessions, they've read the articles, they've skimmed the heavy books,[3] and all of these sources are quite clear: everything *must* be secure. And secure generally means "closed"—particularly if the security folks weren't sufficiently involved in the design, implementation, and operations processes. Normal people, on the other hand, generally just want things to work. There's a fundamental disjoint between those two points of view that we're not going to get fixed until security is the very top requirement for any project from its inception to its ending.[4]

Now, normal people aren't stupid.[5] They know that things can't always work perfectly; but they would like them to work as well as they can. This is the gap[7] that we need to cross. I've talked about managed degradation as a concept [1] before, and this is part of the story. One of the things that we security people should be ready to do is explain that there are risks to be mitigated.

For security people, those risks *should* be mitigated by "failing closed." It's easy to stop risk: you just stop system operation, and there's no risk it can be misused. But for many people, there are other risks: an example being that the organisation may in fact go completely out of business because *some* _____[8] security person turned the ordering system off. If they'd offered me the choice to balance the risk of stopping taking orders against the risk of losing some internal company data, would I have taken it? Well yes, I might have. But if I'm not offered the option, and the risk isn't explained, then I have no choice. These are the sorts of words that I'd like to hear if I'm running a business.

It's not just this type of risk, though. Coming to a project meeting two weeks before launch and announcing that the

project can't be deployed "because the calls against this API aren't being authenticated" is no good at all. To anybody. As a developer, though, I have a different vocabulary—and different concerns—to those of the business owner. How about instead of saying, "you need to use authentication on this API or you can't proceed," the security person asks, "what would happen if data that was provided on this API was incorrect, or provided by someone who wanted to disrupt system operation?" In my experience, most developers are interested—are invested—in the correct operation of the system they're running and the data it processes. Asking questions that show the possible *impact* of lack of security is much more likely to garner positive reactions than an initial "discussion" that basically amounts to a "no."

Don't get me wrong; there are times when we, as security people, need to be firm and stick to our guns.[9] But in the end, it's the owners—of systems, or organisations, or business units, or resources—who get to make the final decision. It's our job to talk to them in words they can understand and ensure that they are as well informed as we can possibly make them. Without just saying "no."

Footnotes

1. By which I mean "those poor unfortunate souls who don't read these posts, unlike you, dear and intelligent reader."
2. My wife, sadly, seems to fall into this category.
3. Which usually have a picture of a lock on the cover.
4. And good luck with that.
5. While we've all met our fair share of stupid normal people, I'm betting you've met your fair share of stupid security people, too, so it balances out.[6]
6. Probably more than balances out. Let's leave it there.
7. Chasm.
8. Insert your favourite adjectival expletive here.
9. Figuratively: I don't condone bringing any weapons, including firearms, to your place of work.

Links

[1] https://aliceevebob.com/2017/04/25/service-degradation-actually-a-good-thing/
[2] https://opensource.com/article/17/11/politics-linux-desktop
[3] https://aliceevebob.com/

Author

I've been in and around Open Source since around 1997, and have been running (GNU) Linux as my main desktop at home and work since then: not always easy... [2] I'm a security bod and architect, and am currently employed as Chief Security Architect for Red Hat. I have a blog – "Alice, Eve & Bob" [3] – where I write (sometimes rather parenthetically) about security. I live in the UK and like single malts.

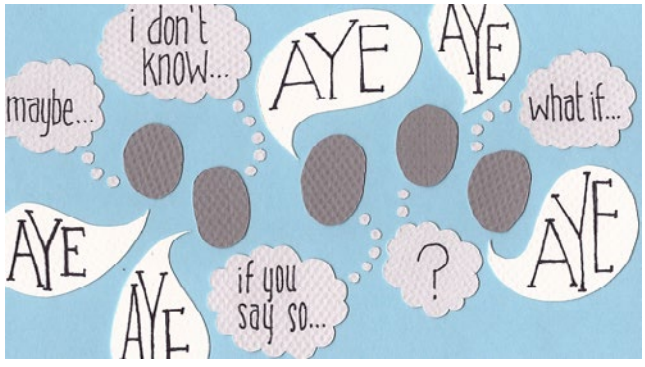
This article originally appeared on Alice, Eve, and Bob – a security blog and is republished with permission. [3]

Who will push back the most on a move to DevOps?

BY MIKE BURSELL

DevOps will definitely bring change to your organization, and not everyone likes change. Here's how to manage those who fight the inevitable.

YOU'RE MOVING to a DevOps [1] model for all or part of your organisation: well done! Somebody, somewhere has made the leap. Let's assume, for the sake of this article, that you have management buy-in: whatever hurdles needed to be jumped, whatever mountains needed to be climbed to get that momentous "Yes." You've got tooling agreed, you've worked out your processes, and now all you need to do is convince people to get involved. Should be easy, right? If only.



It turns out that not all people are as enlightened as you, the reader of this article. Not everybody likes change, and if there's one thing you can be sure of, it's that DevOps will bring change to your organisation—how you work, what you do, how you interact with other people within the team and beyond.

I'm going to describe five types of people or roles who may push against a move to DevOps, along with a few thoughts about possible tactics to help move them along. We should remember, however, that you may not be able to move everybody along, and that there may be good reasons why people don't want to change what they do, including the fact that what they do at the moment may work pretty well, both for them and for the organisation.

Not invented here: Fear of the unknown

"We've done it this way for the past [1/2/5/10/25] years, and it's worked till now." We've all heard this. It may be true, or it may not, but if your management has decided that a move to DevOps should be undertaken, even if the existing practices have been working, there's probably been a realisation that things could be more efficient, or faster, or more secure.

One of the defining points about this type of person or role is that it often exhibits as a team concern. Teams become used to a particular way of doing things and settle into roles and routines that work for them. What you're suggesting is upsetting that team and making people do different things. You should consider how to make the most of the team as it exists now, maybe even transitioning members of that team together or making a point of celebrating their successes, rather than suggesting change is needed because they have in some way failed.

My domain: Fear of loss of control

As a security person by background, this is one I'm very aware of at a personal level. People who have gained a high level of expertise in a particular area or domain often feel threatened when asked to change how they work or apply their knowledge. They will often feel they are being asked to give up control and "water down" their expertise in a new world where "everybody is the expert."

What's important to stress in this context is that, rather than diluting their expertise, this is an opportunity to apply it across a broader set of processes. Testing experts need to explain to developers and operations folks, for instance, how testing methodologies can be exposed in their realms. Typically, exposing experts to a wider audience will be seen by them as a positive and, although there will always be "ivory tower" type personalities who struggle to interact in a more team setting, using them in ways where they take on a "consulting" type role may offer positive opportunities.

Stuck in my way: Fear of the new

While very similar to "Not invented here," this is more of an individual than a group trait. Knowing what your tasks will be on a day-to-day basis may feel stultifying to some but can be very comforting to many, which is why they may not want to move to a world that seems much more "freeform" and unstructured. Not everybody can become the sort of generalist who thrives on understanding all the different parts of the DevOps cycle.

The good news is you will still need people who are ready to settle down on specific tasks and complete them in particular ways. In fact, though there may be initial concerns about moving to a different way of working, explaining that team members will have a fair amount of control over exactly how they perform particular tasks may be a positive message when trying to help this sort of individual. Hopefully, you will be including training—whether formal or informal—as part of your transformation to DevOps, and the chance to learn new skills (thereby increasing individuals' mobility and career prospects) may also act as an incentive.

People managers: Fear of losing power

In many organisations, particularly those with a strongly developed hierarchy, managers have a great deal of control over how their staff are deployed, what their tasks will be, and how their career progression is managed. All of these can be directly at odds with a more open DevOps approach. For managers who have built their own little empire, controlling their reports and subreports like pawns around a chess board, a move to DevOps *will* be challenging. For managers who are keen to grow members of their teams into more expert employees, who measure their success on how many other teams ask for their reports to be seconded to their teams, and who enjoy seeing new skills and career progressions taking place, DevOps should be an exciting opportunity.

Part of any fix to the problem of resistant people managers is likely to be for executive management to offer both a carrot and a stick. The carrot can include changing how people managers are rewarded into a mechanism that embraces these new behaviours, while the stick may involve removing team members from those who are obstructive or changing those managers' role definitions.

Unions: Fear of lack of certainty

In certain industries and geographies, there are strong unions. A core mission of unions is to protect workers from exploitation by management who may try to impose changes on workers that will not benefit them. Unions are by default (and understandably) suspicious of changes introduced by management, so any move to DevOps that has been "blessed" by management may raise concerns and resistance from unions and members of unions. In some cases, employees may have very carefully described job roles that make it difficult to introduce ways of working where they are expected to take a more generalist role and learn new skills—both characteristics of DevOps.

The good news is that DevOps can provide *more* control to members of the team, in many different ways, somewhat reducing the control exercised by the management function. Explaining this and ensuring that appropriate checks are put in place to safeguard jobs will be key tasks in convincing unions and their members that this is a good change for them. The other thing that should happen, of course, is that management should have included them early on in the process to make sure there has been buy-in from the beginning, rather than a decision "sprung" on them at the last moment.

Some final thoughts

As we progress to a bright new future, it is worth bearing in mind that a general good for all does not always translate into a positive change for every individual. It is hard to argue that the construction of sewerage systems is anything other than a general good, but it hits those whose only job has ever been collecting the waste from the streets. Hopefully you don't see your move to DevOps as the construction of a new set of sewers for your organisation, but be aware of those for whom change can be difficult and disruptive. There can be a human cost to even the most well-intentioned development.

For me, the most important point to remember is that when people get defensive—and occasionally aggressive—it is generally because they feel threatened, and in all these cases we've examined, change can be threatening. These people are your colleagues, they are people, too, and they should be treated with respect and consideration as people, not just as roles or obstacles to be overcome. In some cases, preserving the status quo in particular parts of your organisation may be the safest approach—for now, at least.

Links

[1] https://opensource.com/resources/devops
[2] https://opensource.com/article/17/11/politics-linux-desktop
[3] https://aliceevebob.com/

Author

I've been in and around Open Source since around 1997, and have been running (GNU) Linux as my main desktop at home and work since then: not always easy... [2] I'm a security bod and architect, and am currently employed as Chief Security Architect for Red Hat. I have a blog – "Alice, Eve & Bob" [3] – where I write (sometimes rather parenthetically) about security. I live in the UK and like single malts.

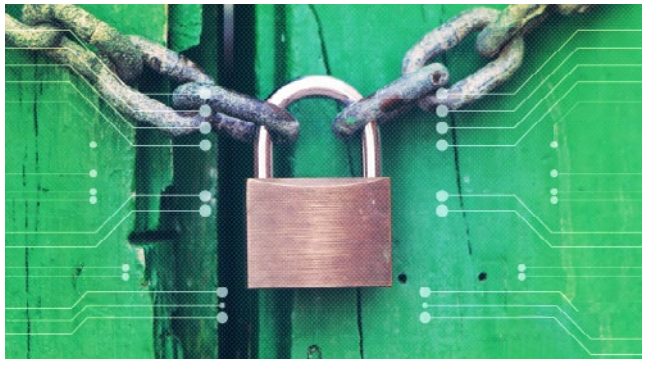
3 security tips for software developers

BY PETE SAVAGE

Don't make these common security mistakes that leave you vulnerable to attack.

EVERY DEVELOPER knows the importance of following best security practices. But too often we cut corners, maybe because we have to work hard until those security practices sink in. Unfortunately, that usually takes something like seeing a security malpractice that's so bad it gets marked in indelible ink in our brains.

I've seen a lot of instances of poor security practices during my career as a sysadmin, but the three I'm going to describe here are basic things that every software developer should avoid. It's important to note that I've seen every single one of these errors committed by large companies and experienced developers, so you can't chalk these mistakes up to novice junior engineers.



1. Don't encrypt passwords, hash them.

Earlier in my career, I worked for a company that used a management system that held some pretty important information. One day I was asked to perform a security review of the network and the software that stored our critical information. I spent a few minutes poking around before deciding to fire up Wireshark to see what traffic was running around the network.

I used my local workstation, logged into the information system, and noticed something weird. Even though this was before SSL was all the rage, I did not expect to see data in plain text containing bytes such as "username" and "password." Upon closer inspection, it appeared that the system was sending my username and a random string—that was not my password—across the wire. I couldn't let it rest. I tried logging in again, except this time I purposely entered my password wrong. I didn't change all of it, just a single character.

What I expected to see was a completely different random string representing the password. Instead, only the first two bytes changed. This was interesting. Even though I was relatively inexperienced, I knew that if the representation of my password were hashed, as it should have been, it would be *entirely* different, not just two characters different. Heck, even a GOOD encryption scheme would do that. This, however, was not doing that at all. I tried two more wrong passwords.

Armed with some sheets of paper and a pencil, I spent the next two hours figuring out the decryption scheme. At the end of those two hours, I had a Python script that could take any of those "encrypted" passwords and decrypt it to reveal the original password, something that no one should *ever* be able to do. I'm sure the person who dreamed up this encryption scheme never thought that someone with a couple of hours on their hands would ever sit down and work it out, but I did.

Why? Because I could.

If you have to store passwords for comparison, never encrypt them, as there is always the possibility that someone can find a decryption algorithm or key. A hash has no direct reverse, meaning no one can reverse it unless they already have a table with the mapping from plain text to hash (or they simply guess it). Knowing the hash mechanism doesn't betray the integrity of the data, whereas knowing the encryption scheme and keys will.

2. Don't put secret backdoors in software.

As part of a third-party software rollout, I was supporting some users who told me that their logins didn't work. This was a paid-for service provided by a vendor, but before troubling with what is usually one of the most annoying support calls ("My login doesn't work"), I thought I would try it myself. It was true, the logins didn't work.

The system was a web-based learning management platform, of which we had paid for a small portion of its greater capabilities. As I poked around on the login page a little more, something caught my eye. One character in one of the words looked different. Perhaps it was a different font, a slightly different shaped "o." Me being me, I viewed the page in source view, and noticed that there was a link associated with this particular letter. The link was purposefully hidden. The mouse cursor didn't change on hovering over it.

I gingerly loaded that mystery link into a new browser window. All of a sudden, I was met with a screen detailing an entire suite of computers, giving me full control over what they could do and the ability to shut them down, reboot them, take screenshots, you name it. I telephoned the software vendor and asked to speak to the IT guy. After jumping through a few hoops, I finally got to someone who knew what I was talking about.

"Oh yeah!" he said. "We put that there for easy access, and no one ever found it until you. We'll remove it right away." Before we ended the call, he asked me one final question: "Why did you start digging around in the HTML?"

My answer was simple: "Because I could."

It's just not worth the risk of putting some fancy backdoor access into any system, because you can bet your bottom dollar someone will find it. No matter how obscure, code analysis—and just general prodding and poking—often yields the most surprising and interesting results.

3. Authenticate users on *every* page—not only on the login page.

At one point in my career, I was involved with a software development project that was being implemented by a seasoned developer. Feeling a little out of my league with this particular application, I told my manager that we would need an in-depth security review of the code. I was asked to look anyway to see what I could find. I started playing with the app, logged in, and viewed some of the data. Then I found something really interesting.

If I bookmarked one of the URLs that I hit further into the system, I could just copy and paste it into another browser, and boom! I'd be there, without having to log in. I asked the developer, "Why don't you check the login on every page? If I just enter the URL of a page further into the system, I can get there without logging in." He asked, "Why would you do that?"

"Because I can," I replied.

Don't leave anything up to chance

Even seasoned developers can make these mistakes. They think that someone won't ever try to delve deeper into a system that they have no real access to. The problem is people will prod, they will poke. The overriding advice I, someone who only dabbles in security, want to impart here is: Don't leave anything up to chance. There are people out there like me, who like to dig into things and see why and how they work. But there are also a great many people who will dig to exploit your flaws and vulnerabilities.

Why? Because they can!

Author

Peter is a passionate open source enthusiast who has been promoting and using open source products for the last 10 years. He has volunteered in many different areas, starting in the Ubuntu community, before moving off into the realms of audio production and later into writing. Career wise he spent much of his early years managing and building datacenters as a sysadmin, before ending up working for Red Hat as a Principal Quailty Engineer for the CloudForms product. He occasionally pops out a book, loves photography, occasionally cooks, and lives in the UK with his wife and two children.

5 ways DevSecOps changes security

BY GORDON HAFF

Security must evolve to keep up with the way today's apps are written and deployed.

THERE'S BEEN AN ongoing kerfuffle over whether we need to expand DevOps [1] to explicitly bring in security. After all, the thinking goes, DevOps [2] has always been something of a shorthand for a broad set of new practices, using new tools (often open source) and built on more collaborative cultures. Why not *DevBizOps* [3] for better aligning with business needs? Or *DevChatOps* to emphasize better and faster communications?

However, as John Willis wrote earlier this year [4] on his coming around to the DevSecOps [5] terminology, "Hopefully, someday we will have a world where we no longer have to use the word *DevSecOps* and security will be an inherent part of all service delivery discussions. Until that day, and at this point, my general conclusion is that it's just three new characters. More importantly, the name really differentiates the problem statement in a world where we as an industry are not doing a great job on information security."

So why aren't we doing a great job on information security, [6] and what does it mean to do a great job in a DevSecOps context?

We've arguably never done a great job of information security in spite of (or maybe because of) the vast industry of complex point products addressing narrow problems. But we also arguably did a good enough job during the era when defending against threats focused on securing the perimeter, network connections were limited, and most users were employees using company-provided devices.

Those circumstances haven't accurately described most organizations' reality for a number of years now. But the current era, which brings in not only DevSecOps but new application architectural patterns, development practices, and an increasing number of threats, defines a stark new normal that requires a faster pace of change. It's not so much that DevSecOps in isolation changes security, but that infosec circa 2018 requires new approaches.

Consider these five areas.

Automation

Lots of automation is a hallmark of DevOps generally. It's partly about speed. If you're going to move fast (and not break things), you need to have repeatable processes that execute without a lot of human intervention. Indeed, automation is one of the best entry points for DevOps, even in organizations that are still mostly working on monolithic legacy apps. Automating routine processes associated with configurations or testing with easy-to-use tools such as Ansible [7] is a common quick hit for starting down the path to DevOps.

DevSecOps is no different. Security today is a continuous process rather than a discrete checkpoint in the application lifecycle, or even a weekly or monthly check. When vulnerabilities are found and fixes issued by a vendor, it's important they be applied quickly given that exploits taking advantage of those vulnerabilities will be out soon.

"Shift left"

Traditional security is often viewed as a gatekeeper at the end of the development process. Check all the boxes and your app goes into production. Otherwise, try again. Security teams have a reputation for saying no a lot.

Therefore, the thinking goes, why not move security earlier (left in a typical left-to-right drawing of a development pipeline)? Security may still say no, but the consequences of rework in early-stage development are a lot less than they are when the app is complete and ready to ship.

I don't like the "shift left" term, though. It implies that security is still a one-time event that's just been moved earlier. Security needs to be a largely automated process everywhere in the application lifecycle, from the supply chain to the development and test process all the way through deployment.

Manage dependencies

One of the big changes we see with modern app development is that you often don't write most of the code. Using open source libraries and frameworks is one obvious case in point. But you may also just use external services from public cloud providers or other sources. In many cases, this external code and services will dwarf what you write yourself.

As a result, DevSecOps needs to include a serious focus on your software supply chain [8]. Are you getting your software from trusted sources? Is it up to date? Is it integrated into the security processes that you use for your own code? What policies do you have in place for which code and APIs you can use? Is commercial support available for the components that you are using for your own production code?

No set of answers are going to be appropriate in all cases. They may be different for a proof-of-concept versus an at-scale production workload. But, as has been the case in manufacturing for a long time (and DevSecOps has many analogs in how manufacturing has evolved), the integrity of the supply chain is critical.

Visibility

I've talked a lot about the need for automation throughout all the stages of the application lifecycle. That makes the assumption that we can see what's going on in each of those stages.

Effective DevSecOps requires effective instrumentation so that automation knows what to do. This instrumentation falls into a number of categories. There are long-term and high-level metrics that help tell us if the overall DevSecOps process is working well. There are critical alerts that require immediate human intervention (the security scanning system is down!). There are alerts, such as for a failed scan, that require remediation. And there are logs of the many parameters we capture for later analysis (what's changing over time? What caused that failure?).

Services vs. monoliths

While DevSecOps practices can be applied across many types of application architectures, they're most effective with small and loosely coupled components that can be updated and reused without potentially forcing changes elsewhere in the app. In their purest form, these components can be microservices [9] or functions, but the general principles apply wherever you have loosely coupled services communicating over a network.

This pattern does introduce some new security challenges. The interactions between components can be complex and the total attack surface can be larger because there are now more entry points to the application across the network.

On the other hand, this type of architecture also means that automated security and monitoring also has more granular visibility into the application components because they're no longer buried deep within a monolithic application.

Don't get too wrapped up in the DevSecOps term, but take it as a reminder that security is evolving because the way that we write and deploy applications is evolving.

Links

[1] https://opensource.com/resources/devops
[2] https://opensource.com/tags/devops
[3] https://opensource.com/article/18/5/steps-apply-devops-culture-beyond-it
[4] https://www.devsecopsdays.com/articles/its-just-a-name
[5] https://opensource.com/article/18/4/devsecops
[6] https://opensource.com/article/18/6/where-cycle-security-devops
[7] https://opensource.com/tags/ansible
[8] https://opensource.com/article/17/1/be-open-source-supply-chain
[9] https://opensource.com/tags/microservices

Author

Gordon Haff is Red Hat technology evangelist, is a frequent and highly acclaimed speaker at customer and industry events, and helps develop strategy across Red Hat's full portfolio of cloud solutions. He is the co-author of Pots and Vats to Computers and Apps: How Software Learned to Package Itself in addition to numerous other publications. Prior to Red Hat, Gordon wrote hundreds of research notes, was frequently quoted in publications like The New York Times on a wide range of IT topics, and advised clients on product and marketing strategies. Earlier in his career, he was responsible for bringing a wide range of computer systems, from minicomputers to large UNIX servers, to market while at Data General. Gordon has engineering degrees from MIT and Dartmouth and an MBA from Cornell's Johnson School. Follow me at *@ghaff*

GET INVOLVED

If you find these articles useful, get involved! Your feedback helps improve the status quo for all things DevOps.

Contribute to the Opensource.com DevOps resource collection, and join the team of DevOps practitioners and enthusiasts who want to share the open source stories happening in the world of IT.

The Open Source DevOps team is looking for writers, curators, and others who can help us explore the intersection of open source and DevOps. We're especially interested in stories on the following topics:

- DevOps practical how to's
- DevOps and open source
- DevOps and talent
- DevOps and culture
- DevSecOps/rugged software

Learn more about the Opensource.com DevOps team: https://opensource.com/devops-team

ADDITIONAL RESOURCES

The open source guide to DevOps monitoring tools

This free download for sysadmin observability tools includes analysis of open source monitoring, log aggregation, alerting/visualizations, and distributed tracing tools.

Download it now: The open source guide to DevOps monitoring tools

The ultimate DevOps hiring guide

This free download provides advice, tactics, and information about the state of DevOps hiring for both job seekers and hiring managers.

Download it now: The ultimate DevOps hiring guide

The Open Organization Guide to IT Culture Change

In The Open Organization Guide to IT Culture Change, more than 25 contributors from open communities, companies, and projects offer hard-won lessons and practical advice on how to create an open IT department that can deliver better, faster results and unparalleled business value.

Download it now: The Open Organization Guide to IT Culture Change

WRITE FOR US

Would you like to write for Opensource.com? Our editorial calendar includes upcoming themes, community columns, and topic suggestions: https://opensource.com/calendar

Learn more about writing for Opensource.com at: https://opensource.com/writers

We're always looking for open source-related articles on the following topics:

Big data: Open source big data tools, stories, communities, and news.

Command-line tips: Tricks and tips for the Linux command-line.

Containers and Kubernetes: Getting started with containers, best practices, security, news, projects, and case studies.

Education: Open source projects, tools, solutions, and resources for educators, students, and the classroom.

Geek culture: Open source-related geek culture stories.

Hardware: Open source hardware projects, maker culture, new products, howtos, and tutorials.

Machine learning and AI: Open source tools, programs, projects and howtos for machine learning and artificial intelligence.

Programming: Share your favorite scripts, tips for getting started, tricks for developers, tutorials, and tell us about your favorite programming languages and communities.

Security: Tips and tricks for securing your systems, best practices, checklists, tutorials and tools, case studies, and security-related project updates.