

anchore

THE FUNDAMENTALS OF CONTAINER SECURITY

Transform a growing IT security liability into a powerful new asset.



According to recent research from the CNCF, the use of containers has increased significantly during 2019, with 84% of survey respondents now using containers in production – an annual jump of more than 15%.

The humble software container has created a tornado in the tech industry. It has taken all aspects of the software lifecycle, spun them around, and blended them together in new and previously unimaginable ways.

Security is no exception. Containerization has changed the required model for software security just as radically as it has changed software design, development, distribution, testing and implementation.

Without a solid, focused strategy for container security, the modern enterprise is opening up a significant new attack surface and leaving it largely unchecked.

As a result of this, the software container scanning marketplace is becoming increasingly crowded. New entrants are appearing regularly, eager to capitalize on the obvious need of development teams to secure this relatively new technology.

Container scanning is, in many ways, now a commodity, with many vendors offering it alongside 'traditional' security scanning services. The nature of containers makes it relatively simple to scan them on a superficial level. Creating additional services around containers makes sense for existing security vendors, allowing them to offer a tempting 'all-in-one' security suite.



However, container scanning is only the start point for effective container security. Most container security solutions miss the real point of containers, treating them much like virtual machines (VMs) or servers. By and large, existing vendors offer late-stage scanning solutions, waiting until a container image is fully developed and ready for deployment and, only then, blocking its release.

Waiting until the very end of the development lifecycle to intervene, destroys the velocity advantage that containers provide. Fixing security issues becomes, at best, slow and disruptive, forcing a complete, additional development and testing cycle to fix security issues. At worst, angry developers will actively seek to circumvent such a system or push for it to warn only. At this point, container security is in name only.

Container security is not purely a technical fix or tool. To deliver results without hindering the original efficiency benefits of containerization, container security must be designed, from the ground-up, to fit with a container-focused software delivery model. Security insights must be shifted left, and embedded back into the CI/CD (continuous integration and continuous delivery) workflow.

In this paper, we start by exploring some of the general background around containerization, its benefits and how many of these benefits can be extended to security. We then go on to look at some of the unique new security challenges presented by full-speed container-based development.

The paper explores the options for container security and looks in more depth at the role of tooling and the accompanying need for cultural change. Finally, we take a practical look at what is really needed for organizations to move towards frictionless 'security at full speed', examining what is needed to move DevSecOps from theory into practice.

Short-lived and Immutable

Before containers, software environments were bespoke and fractured affairs. It has always been possible to start servers and virtual machines from a single 'golden image'. But, over the course of their active life, most of them tended to suffer from configuration drift: collecting modifications, hotfixes, new agents, malware and more.

Virtual machines are mutable by design. And any alterations persist throughout the long lifecycle of the VM. Reboot it, move it, do anything short of destroying it; and these changes remain safely stored in that VM. The separate mutability of each and every VM running in production creates divergence. And this lack of any consistent state makes it incredibly challenging to create representative environments for debugging purposes.

By contrast, container images are immutable. The containers created from them are constantly started and stopped and may only exist for seconds. You can introduce changes to a container during its lifetime, however, these do not survive the frequent restarts. And because the images are both immutable and containers short-lived, this makes it relatively simple to avoid configuration drift.

The only place that changes can persist for any length of time is within allocated storage, either in the form of a volume mount or network file store. These stores should never be used for the application code or configuration itself, and are easily identifiable.

The container's immutability makes it far simpler for developers to stand up a debugging environment, even on a laptop. As long as you have the same version of the container image as the environment being debugged, along with any configuration items applied to it, you have pretty much the same software environment. In addition, you can easily stand up as many environments as you need, allowing developers and operations teams to collaborate and troubleshoot with the same point of reference.

Maintaining a development environment:

'It worked on my laptop!'

The software development industry has been on a long and storied quest to eliminate the phrase 'it worked on my laptop' from common usage. Sadly, for many teams, this is still a work in progress and maintaining a development environment can be hard.

Let us take Python as an example:

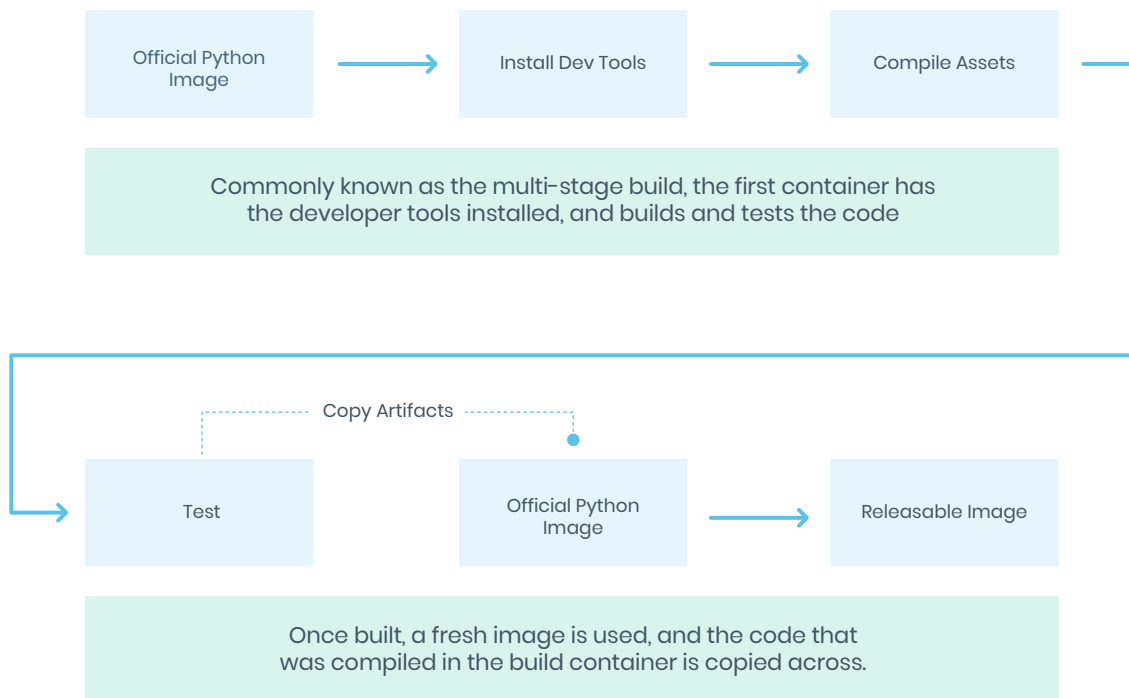
Python is enjoying massive growth in popularity, with StackOverflow trends showing a sustained upward curve of usage. Often, it is used in the context of Microservices, allowing developers to write and deploy discrete and focused code. However, this is where problems of maintaining a development environment become apparent.

While offering a powerful approach to software architecture, microservices suffer from version and configuration sprawl. Once you are into the realm of tens, or even hundreds of services, maintaining a single language runtime becomes challenging. Some new services may be at the latest version, whereas services written a few months ago may be using old versions of the runtime or libraries.

Microservices often force software developers to maintain multiple versions of their chosen language. Tools such as Pyenv help maintain different versions but can be brittle, complex, and they are rarely integrated into the final production deployment.

Containers offer a more workable solution to using pyenv or other tools to maintain versions.

A container offers an immutable version of an environment, allowing the same version of the runtime used in production to be used to develop software. In some ways, this is nothing new. Tools such as Vagrant offer the same ability using a virtual machine. Although powerful, Vagrant-style workflows lack features that containers offer, especially ‘composability’.



A container image can be used as a parent image for another container, an ability that is difficult—if not outright impossible—to offer with virtual machine images. Composability allows a workflow like the one below:

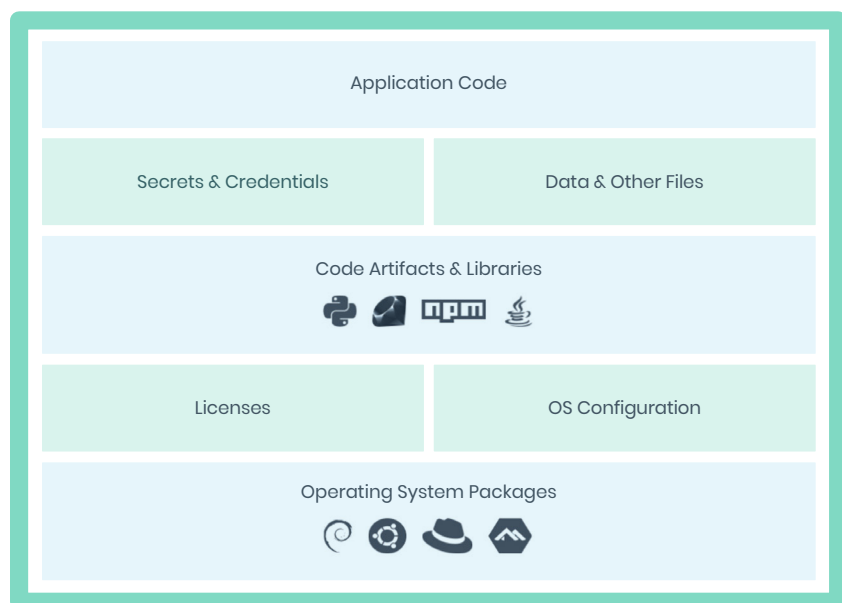
In this workflow, developers can pull a strongly versioned container image for their language, and then add another image layer with their development tools on top of this. Once they have finished coding, they can replace their development layer with a testing image and run comprehensive tests. The same parent container is used, from inception to deployment.

Image composition also allows developers and operators to collaborate and work on an identical environment. This collaboration makes a huge difference to support: completely negating that other, all too common refrain:

‘Now I need an environment to debug this in’

Extending Container Benefits to Security

Many of the strengths that containers offer for both developers and operators, can also provide serious security benefits. Containers provide a single versionable artifact that, with the right tools, can be used to generate a complete bill of materials. Every package, be it software, libraries or OS is visible within a container image and is immutable. Once checked and approved, the image will not change.



A **container image** is a single artifact that contains everything required to run an application or service.

This immutability also makes it difficult for attackers to gain a foothold within a containerized infrastructure. Generally, attackers establish beachheads within compromised servers to launch further attacks. With containers, that beachhead lasts as long as the container, and when it is restarted, the container is returned to a fresh state based on its immutable image. An attacker would have to repeat any attack that allowed them access and re-establish a base of operations. This costs the attacker time and gives more opportunities for detection. Although still not as mature as virtual machines in some aspects of security, the ability to create truly ephemeral workloads is one advantage containers do offer that virtual machines do not possess.

Collaboration is at the heart of the container workflow. Operators can create and stipulate the use of parent images that hardcode their best practices into every container. Developers can then easily consume these images to develop and release code: ensuring simple, repeatable, controlled consistency across development, testing and production environments. Each team can improve, iterate and develop in tandem without blocking the other.

Containers deliver a technical mechanism that can not only accommodate but also promote collaboration across previously disparate teams. And the security team can join the party too. The collaborative framework containers provide is what drives the mind-shift needed to turn DevSecOps from a buzzword into a corporate-cultural reality.

With containers, the security team has a single place to examine the OS package and configuration, code and dependent libraries. Since a container can easily be versioned and checksummed, security operatives can track what issues were found in which artifact. Even better, this assured immutability gives the security team an easy way to prescribe secure parent images for both developers and operations to use.

Collaborating around containers provides operators with fantastic new capabilities when it comes to deployment, opening the door to self-healing, elastically scalable infrastructure, and multi-cloud deployments. Again, containers and orchestration are able to turn a buzzword soup into working infrastructure, realizing the long-time goal for many operations teams.

The Security Challenge

Containers are no panacea and, along with their considerable benefits, come additional challenges. Many of these challenges are in the form of security and, ironically, some of the elements that make containers such a boon are also the very things that can make them a potential security nightmare.

Introducing Security at Full Speed

Software security relies on transparency at every level. To find and mitigate security issues and vulnerabilities, they must first be visible.

One key aspect is having the time to detect code changes, evaluate them and guarantee that they are safe. If this is done manually, speed becomes a challenge, and containers are all about speed.

Used from development to deployment, containers make it easy to introduce new libraries and even operating system packages into production environments. The sheer speed of change can leave security practitioners standing in the dust, especially if they lack effective automation.

The velocity of container based development and the speed with which new external components are added presents a significant new threat. Supply chain attacks and Zero-day exploits are on the rise, with [reports from Symantec](#) suggesting a growth of 78% in 2019.



Increasingly, breaches do not occur because an aggressor has forced entry to the network, but because a developer inadvertently invited the attacker through the front door, by including a compromised dependency.

These supply chain attacks can differ in complexity and sophistication:

Attackers can operate through libraries, distributing a compromised library that is deliberately named with a minor typo; say lftshift rather than leftshift. So long as the library functions as the developer expects, it is unlikely they will realize their mistake until it is too late.

Threat actors have even been targeting quieter open source projects, with less active or badly managed communities. Smaller, quieter projects, libraries and components can allow bogus contributors to rapidly build trust and gain the all-important commit rights to add new code. Once in, attackers have legitimized-access to a ready-made distribution mechanism for injecting malicious code or vulnerabilities into the project's user base.

Up to now, the majority of supply chain attacks have been through the realm of software libraries. However, with the rapid growth in the popularity of publicly shared containers, it is only a matter of time before attackers seize this new and obvious opportunity.

As discussed earlier, composability offers one of the strongest characteristics of containers for increasing the speed of development. It allows developers to iterate, building on and refining the work of others, to get where they want faster. If an existing public container image offers most of what they need, a developer can use this as the parent image and then simply apply the changes or additional components they require. In this way, production containers can often be defined by several layers of parent images.

However, composability also opens up new possibilities for attackers to add vulnerabilities to a publicly accessible container. With a well chosen, popular container, a breach could be spread rapidly into a large number of targets. In this way, it is easy to see why containers bring fresh new attack surfaces with them. Even if a container is free of exploitable software, it may still contain a host of other concerns.

Dedicated Container Best Practice

Containers also require their own security best practices. One important example is that processes within a container should not be run as the root user. This sensible precaution helps stop potential malware breaking out of a container by leveraging higher privileges on the underlying server. Likewise, it is not recommended to set OS components to update automatically, but rather to pin containers to use specific software versions.



Despite these and other best practices being well documented, most container tools do not emit warnings if they are not followed. Instead, it is left to the developer to apply them, either by having an encyclopaedic knowledge of container best practice or using some form of linting tools. As a result of this lack of automation, a terrifying proportion of public images do not follow best practices. This means that, even if internal developers follow best practice, there is no guarantee that the parent images they are using implement the same security standards.

Finally, containers can be challenging from a licensing perspective. Some organizations are averse to the risks and legal liabilities of specific software licenses.

Where developers can amend the OS tools and packages that ship with their container, it becomes straightforward for components with unwanted licenses to creep in. The offending component may not even be part of the intended package, and could have been dragged in unwittingly as a dependency of a dependency. Without automation and tooling, this is a licensing issue that is hard to spot. The only alternative is a time-consuming manual evaluation of an image.

Containers bring speed, sharing, some fantastic benefits, and potentially, a container-ship-sized headache for security professionals. Without automation such as [Anchore](#), a practitioner can never be expected to keep on top of emerging threats inside containers, without killing the very benefits they are meant to bring.

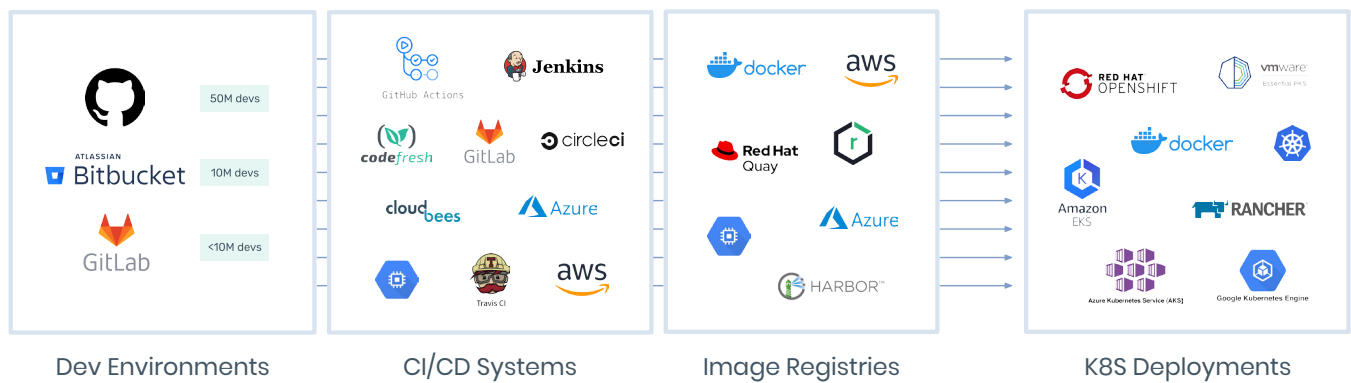
Don't Mess with My Tools

Suite Nothings

On the surface of things, there are two ways to approach the selection of security tooling. The first approach is to look at a comprehensive suite or 'ecosystem'. This ecosystem will typically offer broad functionality and claim many amazing and integrated benefits. Buying into an ecosystem is simple from the point of view of purchasing and installation: one supplier, one support line, one tool, and a straightforward set of packages to maintain. If the right price can be agreed upon, this might seem like the logical choice. But stop for a minute...

We all know the sacrosanct nature of someone's tools. If you have ever worked together with skilled tradespeople, artisans and craftspeople; if you have ever had a father with a shed; you know, you just don't mess with someone's tools. A bad worker may blame their tools, but not nearly as much as a proud worker will blame you, if you tamper with their favoured widget or gadget.





Choosing your own tools is a basic human right of anyone who takes pride in their job. We all choose our work tools to fit with our idiosyncrasies and favoured way of working. And, over time, we even refine our ways of working around our tools. In short, our tools can almost become an extension of our professional selves.

Developers are no different. And why should they be?

The reality is that overarching product suites rarely deliver on their promise, tending to compromise on features to offer broader scope. Compromise is inevitable when you have a product that offers, for instance, an artifact store, security scanning, CI/CD pipeline, container auditing and more. Not all features can receive the same level of development attention, and outside of the core product, features can be cursory. All too often, features can even be added in situations where it may make no sense in that workflow.

The Swiss Army knife is a tediously overused metaphor in the IT industry. But if you were to hand a comprehensive, top-of-the-range Victorinox to an electrician and ask them to rewire your kitchen with it...?

Moreover, because larger security suites are integrated, the products within the ecosystem are typically rigidly integrated with each other. Adding in a third-party product to the ecosystem is normally difficult and external tools can rarely be integrated fully. So, developers really are stuck with just our little red helvetic friend.

When this impact starts to reach beyond security into the developer's own toolchain, things move beyond a joke. Security software has a profound impact on every team that interacts with it. We no longer live in a world where you buy some software in black and yellow striped packaging, install it, and forget about it.

Security must involve every part of the software development lifecycle. And effective security tools may need to interact with each development tool, and every developer, involved along the way. This can force changes to organizational workflows as a result: it is often seen as easier to follow the path of least resistance and use the entire ecosystem than retaining tools that cannot integrate with them.

This is where security really starts ‘messing with the developers’ tools’.

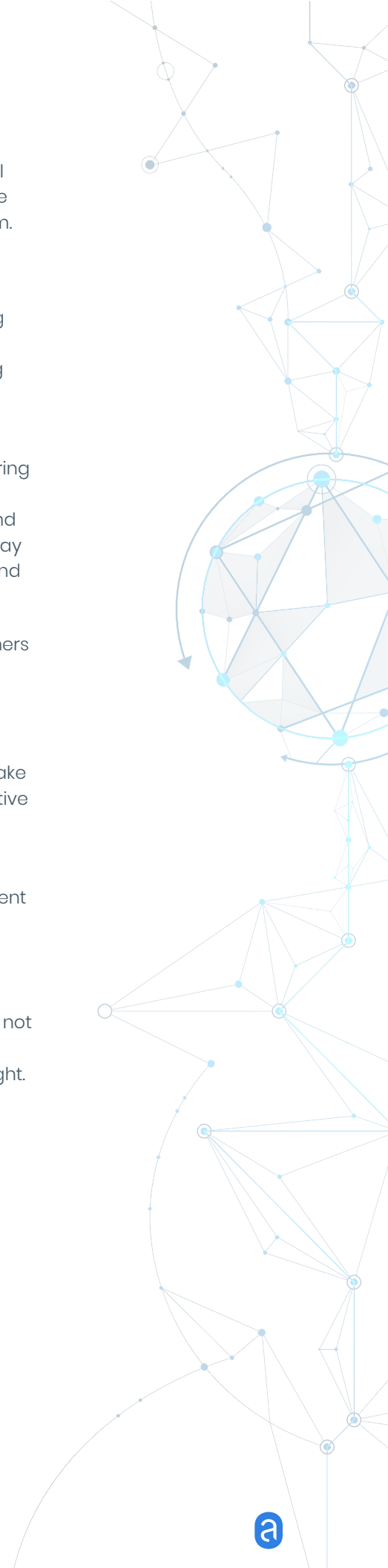
And while we may joke, these changes to the development process can be jarring if they replace generally accepted and liked working practices. It can create resentment between teams and it can easily be perceived as one team imposing poor working conditions on another: generally, either the security or operations teams imposing on the developers.

This not only causes ill feelings, but can dramatically reduce productivity, both during and after the rollout of the new ecosystem. Moreover, like any worker at the top of their game, real developer talent will look for roles where they have the respect and freedom to choose their own tools and shape their working environment. Take away this respect and freedom, and companies will inevitably find it harder to attract and retain talented individuals.

In addition, many of the larger tool suites simply do not integrate well with containers and the pace at which they work. Security software broadly works in two modes; inspection or gating. On their own, tools in inspection mode are near-useless. They rely entirely on the user understanding the inspection data produced by the tool, on their understanding of the relevance of this data in the context of their organization’s security policy, and on the user having the time and inclination to take the appropriate action. Conversely, tools in gating mode can be massively disruptive if they are poorly implemented.

Often, security-focused tool suites are designed with audit, gatekeeping and operational stability in mind. They either side-line or outright ignore the development process. This massively inflates the iteration loop that developers are forced to undertake, and can cause serious harm to the developer cadence.

To be effective, tools must offer inspection and gating, with multiple integration points along the development chain. However, they must do this in such a way as not to disrupt the development cadence. It is vital that security tools do not waste the hard-won advantages that DevOps working practices and containers have brought.



The New Role for Best of Breed

Few would dispute that the superior alternative to the suite or ecosystem is to take the best tools for each part of the development cycle and form them into an optimal toolchain. However, in the past, this approach has often been perceived as too costly and time-consuming: requiring the teams using these tools to install and manage complex software and solve issues integrating them.

Containerization has changed this. Where disparate tools would once have been massively complex to install, run and maintain, they can now be deployed using a container. Kubernetes is an excellent example of this approach. Each Kubernetes distribution is a collection of tools fulfilling the Kubernetes API, and even the underlying networking layer can be swapped out.

Kubernetes 'operators' are the next evolution in this process, offering to abstract complexity even further. They allow Kubernetes administrators to describe the configuration for different products using Custom Resource Definitions. It is no longer the case that creating your tooling needs to be involved, expensive or hard to maintain.

Creating a tailored security toolchain allows you to select the best tool for each aspect of security. **Anchore** has been designed to fill the role of container security scanning and policy enforcement. It is not a static code analyzer, nor a replacement for network security tools. It is a best of breed container scanning tool, incorporating inspection and gating, that can be easily integrated into workflows and used alongside tools that specialize in other aspects of security.

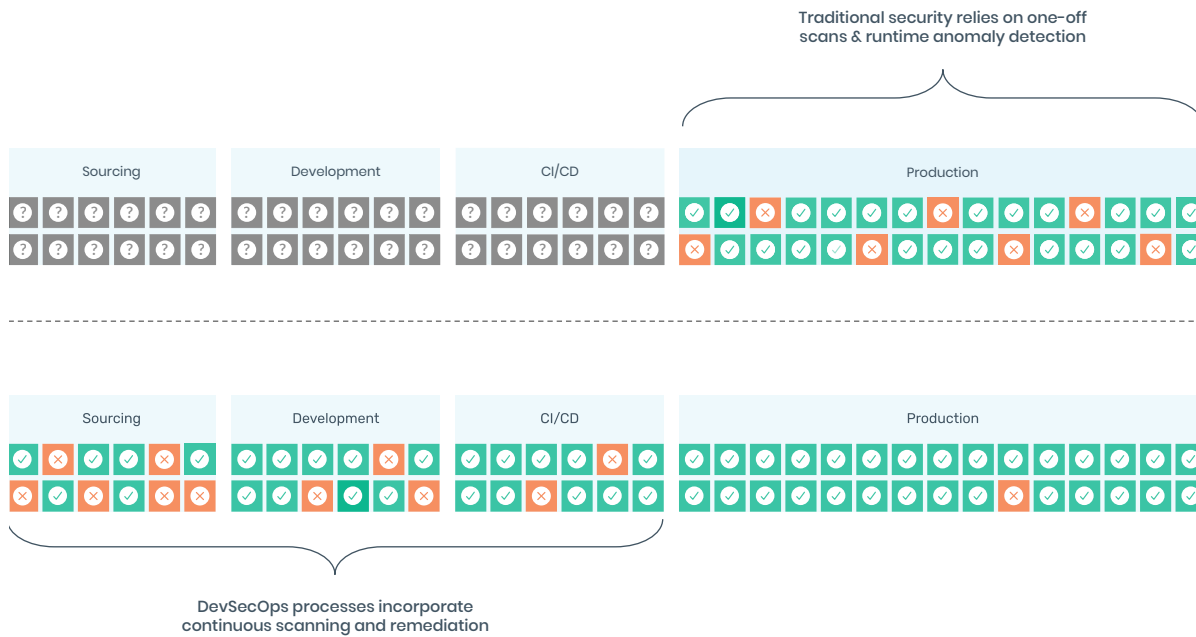
Anchore is fully automatable, allowing it to be integrated with practically any working practice, toolset and step of the software lifecycle. **Anchore** does not interrupt processes and demand instant change. It fits around what is there and encourages subtle changes over time, as developers and security admins alike learn what does and does not work. Security becomes a frictionless extension of existing development techniques.

Frictionless Security

For security to become frictionless, the security team must make allies of everyone who takes part in the development of software, allowing them to quickly and easily identify and remediate security issues.

By giving development teams the information they need to fix security issues early on in the development process, it frees up the security team to focus their time on more complex and more interesting aspects of security. Catching issues quickly and at the right part of the software lifecycle, ensures there is no tedious 'grunt work' left to be done. Security at full speed spreads the load, allowing developers to feel part of the security picture, rather than having it imposed upon them.

Just as operations benefit from the greater levels of collaboration and openness found in DevOps, so can security. This increasingly collaborative approach has led to the creation of DevSecOps as an extension of the DevOps methodology.



Much like its progenitor, DevSecOps pushes left, ensuring that Security, as well as Operations, is considered from the very beginning of the development lifecycle. Much like DevOps, DevSecOps is a cultural development, but it is being enabled by containerization and CI/CD. DevSecOps is driven by the adoption of new automation and tooling that helps foster collaboration.

Anchore is a security tool that has been built for DevSecOps and which encourages the necessary cultural change. It is focused on making security frictionless, does not require constant maintenance and, once implemented, fades into the background.

By contrast, the older generation of security tools cannot be expected to fit with this newer security paradigm. Awkwardly retrofitting tools to a newer generation of DevSecOps processes will almost inevitably create friction, standing in the way of collaboration and leaving DevSecOps out of reach, as an ideal rather than a reality.

Frictionless security demands tools that do not break the development flow: no use of custom GUIs or other out-of-band ways to deliver insight and information. Where possible, security tools must melt into the background, simply enriching existing developer alerts to add detailed security information. Flexibility, automation and ease of integration with multiple third-party developer tools are the key. This allows security insight to be taken into account and implemented across different stages of the CI/CD pipeline and in every part of the software development lifecycle.

Anchore is driven by security policy expressed as code. Policies allow **Anchore** to be both centralized and decentralized at the same time.

Policy is written and maintained by security specialists, providing them with the flexibility to define different and context-appropriate levels of enforcement. These policies are then consumed by **Anchore** wherever it is run: be that the developer's laptop, or a Kubernetes cluster. Policies are a powerful way to deliver a nuanced approach to enforcement, creating a frictionless developer experience without endangering security in production environments.

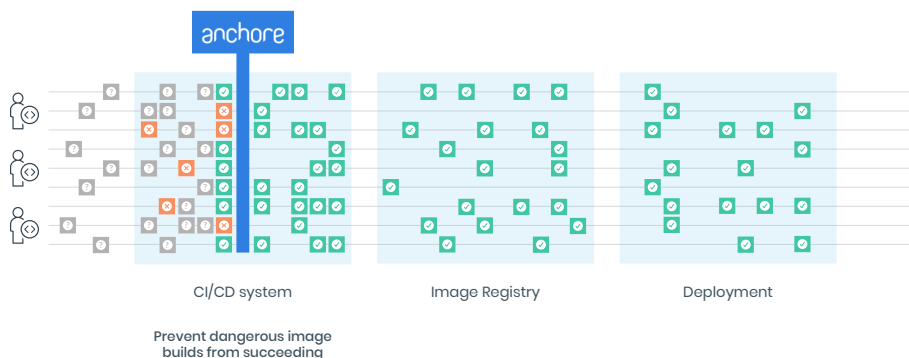
Security at full speed means that security should never slow down or block development, but equally, development imperatives should never compromise security. These two tensions can be balanced using a DevSecOps approach and pushing ownership of security left. In the case of **Anchore**, this means using it as close to the developer as possible: integrating security feedback at multiple points across the development cycle, and using appropriate policies to inform, but not block development.

Security at Every Layer

Integrating **Anchore** with developer workflows means integrating with their local toolings such as their IDE, the CI pipeline, or both. **Anchore's** flexible policy framework then allows for increasing levels of both scrutiny and enforcement as artifacts pass through the development pipeline.

On the desktop, a lightweight scan can be used to catch top-level issues and to check that containers are using best practices. This immediate feedback is comparable to a developer running a unit test. It can be integrated into the workflow either using a make file, IDE integration, or tools such as Githooks. This integration allows the developer to catch high-level issues right inside their workflow as they are working, without making them wait for a report.

Policies become a helpful, non-invasive tool for education, promoting best practice for secure development. The comfort of the tooling effectively trains the development team to work more securely from the outset.

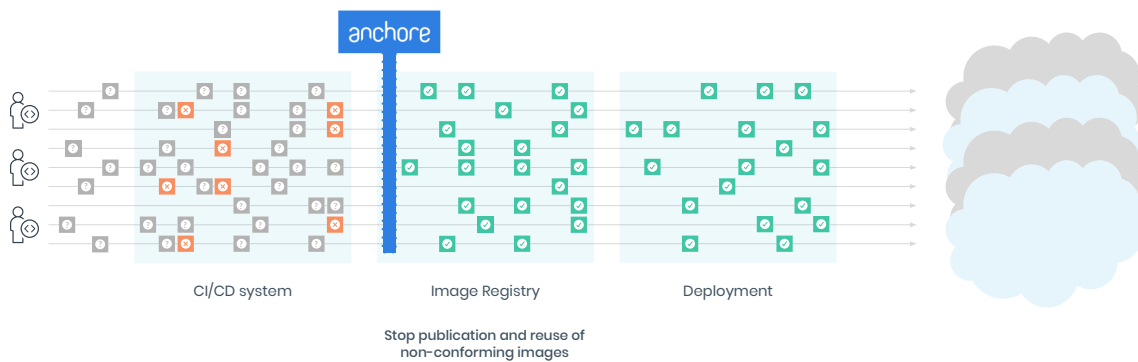


Once a new software change is ready, it can be pushed into the CI/CD pipeline. Here, a second set of policies can be set to run a more detailed scan, looking deeper into software libraries, and integrating with other tools that can enrich the security picture. Rather than blocking further merging activity, the policy can be set to report security issues.

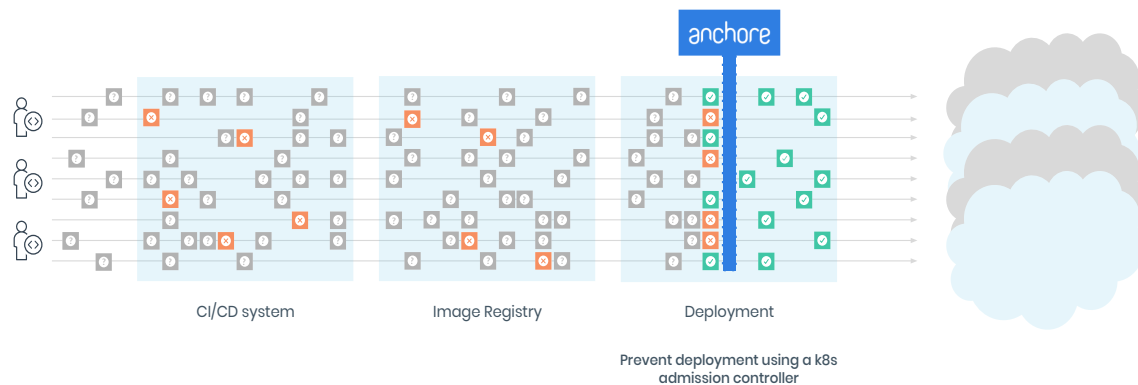
In this way, the code can be further tested without the need for immediate action. At the end of the pipeline, the developer has an actionable report of any security issues, as well as any integration problems that may have been encountered. The developer receives the same familiar testing feedback in the way they always have, but with added details around security.

Meanwhile, the security team can continuously update the developer policy, allowing them to fine-tune and adapt the issues that are surfaced at each point of development. The developers, in turn, don't need to be aware of this; from their point of view, the testing is being seamlessly updated to ensure their code is properly scanned for any issues.

Once the deployment cycle is finished, the developer can push their code into a container repository. Here, **Anchor** can be configured with a third set of policies, performing deep scanning and auditing of the entire container registry. As new threats emerge, **Anchor** is able to report on existing assets within the registry without requiring a complete rescan. Instead, it utilises the bill of materials stored from previous scans and checks if the new vulnerabilities are present in the registry, giving security teams automated feedback on their security posture, and allowing them to take action to remediate any issues.



Finally, a fourth level of policies can be applied at run time using the Kubernetes ingress controller. The ingress controller is a perpetual watchdog that runs within the Kubernetes cluster, inspecting every container at launch and checking against a policy what actions to take if vulnerabilities are found.



These controller policies can then be further tuned and refined to fit the security posture of a given environment. In a test cluster, it may be enough to scan and flag any issues found with a container, but allow it to launch. This approach enables security teams to work with developers to fix issues, but not act as a blocker for further functional testing or performance testing. The pre-production cluster can ensure that only containers that have been scanned in the test environment are allowed to launch.

For production environments, policies can be set at their most rigorous. Here, only whitelisted containers that have previously been scanned, and passed tests should be allowed to launch. This severely hampers an attacker's ability to run arbitrary workloads in production. It applies a significant amount of extra runtime security, without hindering the velocity of developers.

At the same time as delivering a frictionless relationship with developers, **Anchore** also allows the security team to collaborate with the broader business. Often, security teams are deeply involved with wider business auditing and are tasked with supplying reports on current security issues, possible license issues, compliance levels and any recent security activities. **Anchore** produces reports in a variety of easily ingestible formats such as CSV. This can be integrated into existing workflows, giving the security team the freedom to add context and information around the data, rather than spending all of their time collating it.



Conclusion

Containers have finally enabled organizations to realize the DevOps and CI/CD vision. And in doing so, this has fundamentally changed all aspects of the software lifecycle. Within the context of this root change, existing software security tools cannot just be marginally tweaked to fit.

Perhaps the greatest overarching change, and the driver for a large proportion of enterprise adoption of containers, is in the dramatic increase in cycle speeds from development to implementation. It is therefore vital that software security solutions can handle the very real implications of this increased development speed and, equally, that they do not diminish or destroy the very speed and efficiency gains that containers have provided.

Effective container security is about much more than scanning the contents of a container. It has to be about integrating security policy and insight along the development cycle - from start to finish. In this way, container security is both mandating, and partly enabling, DevSecOps as the next evolution in enterprise CI/CD software development.

Security at full speed pushes security left and allows software development teams to fully enable the DevSecOps approach. Rather than making Security front and center, it makes it an integrated and seamless part of existing software development workflows. DevSecOps needs to integrate security insight and information into the developers' toolchain, not burden them with more tools.

For DevSecOps to work, it must be implemented with respect for the developers' ways of working. Container security tools cannot force screen hopping to out-of-band applications, require cross-collation of information between tools, or cause disruptive blockages in the existing DevOps workflow. Organizations need security tools that can be fully automated, using an API or command-line tool, so that they can be integrated into any development workflow and with any tool. This capability ensures that software development teams can use security insight and information within their existing workflows.

By integrating security information, in context, throughout the developer pipeline, developers can be guided to implement best practice as part of their normal processes. Implementing security policy becomes the new development norm. And as developer practices improve, policies can even be hardened, taking the organizations from meeting security guidelines and regulations, towards exceeding them.

Containers demand a new approach to software security. But adopting this new approach dramatically increases the efficiency of security implementation, in the same way that DevOps has increased efficiency in development and IT operations efficiency. By smoothing interaction and removing much of the historical friction between developers and security teams, this new model of DevSecOps or security at full speed, allows organizations to achieve more and raise security standards to new levels.

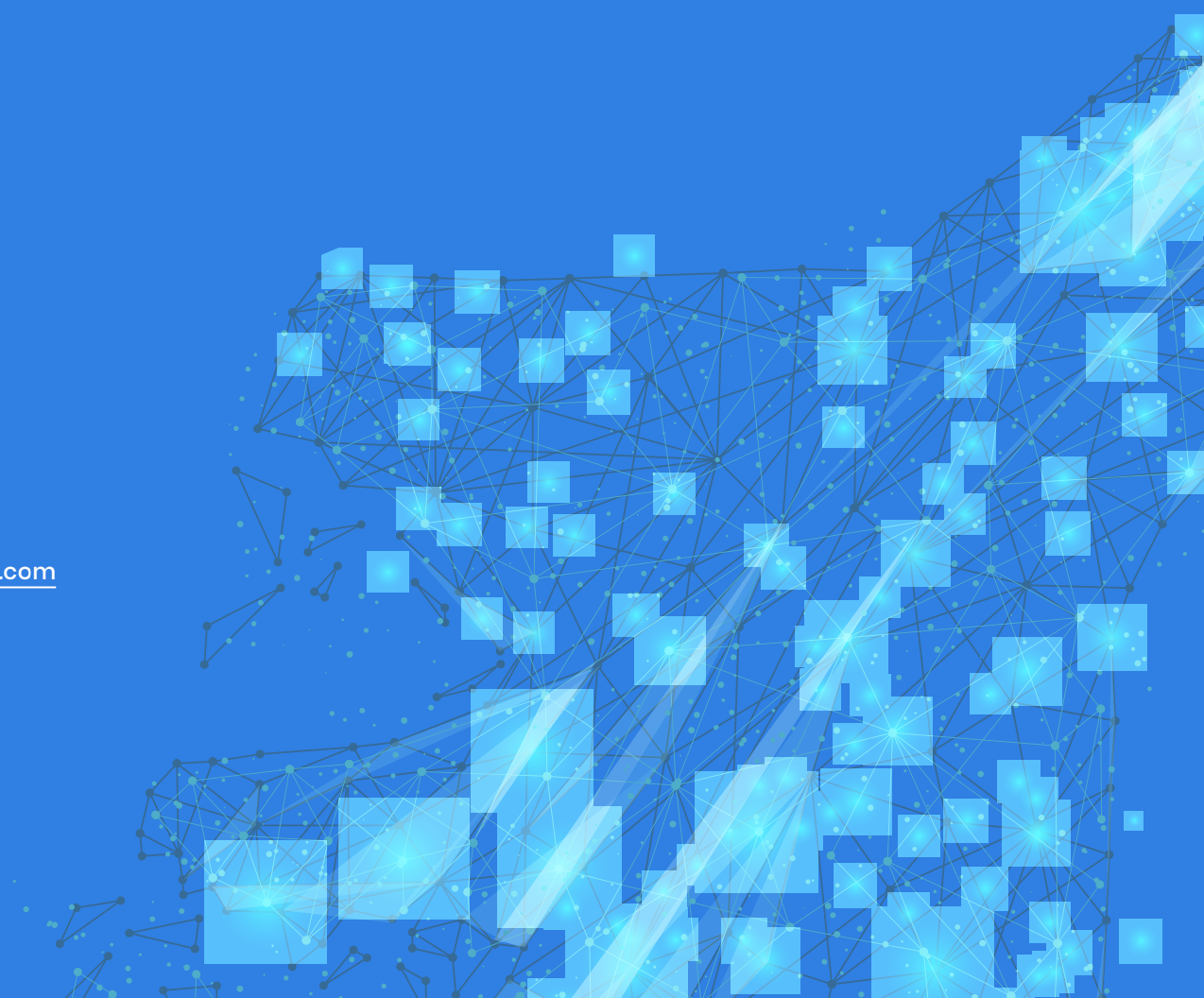
About

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

An abstract geometric pattern in the bottom right corner of the page. It consists of numerous blue squares of varying sizes, some of which are connected by thin, dark lines, creating a network-like structure. The squares and lines are set against a solid blue background, and the overall effect is a complex, crystalline or molecular structure that appears to be growing or expanding from the bottom right towards the center of the page.