Opcito

Continuous Innovation Delivered

DevSecOps

Building Security into Your DevOps Processes

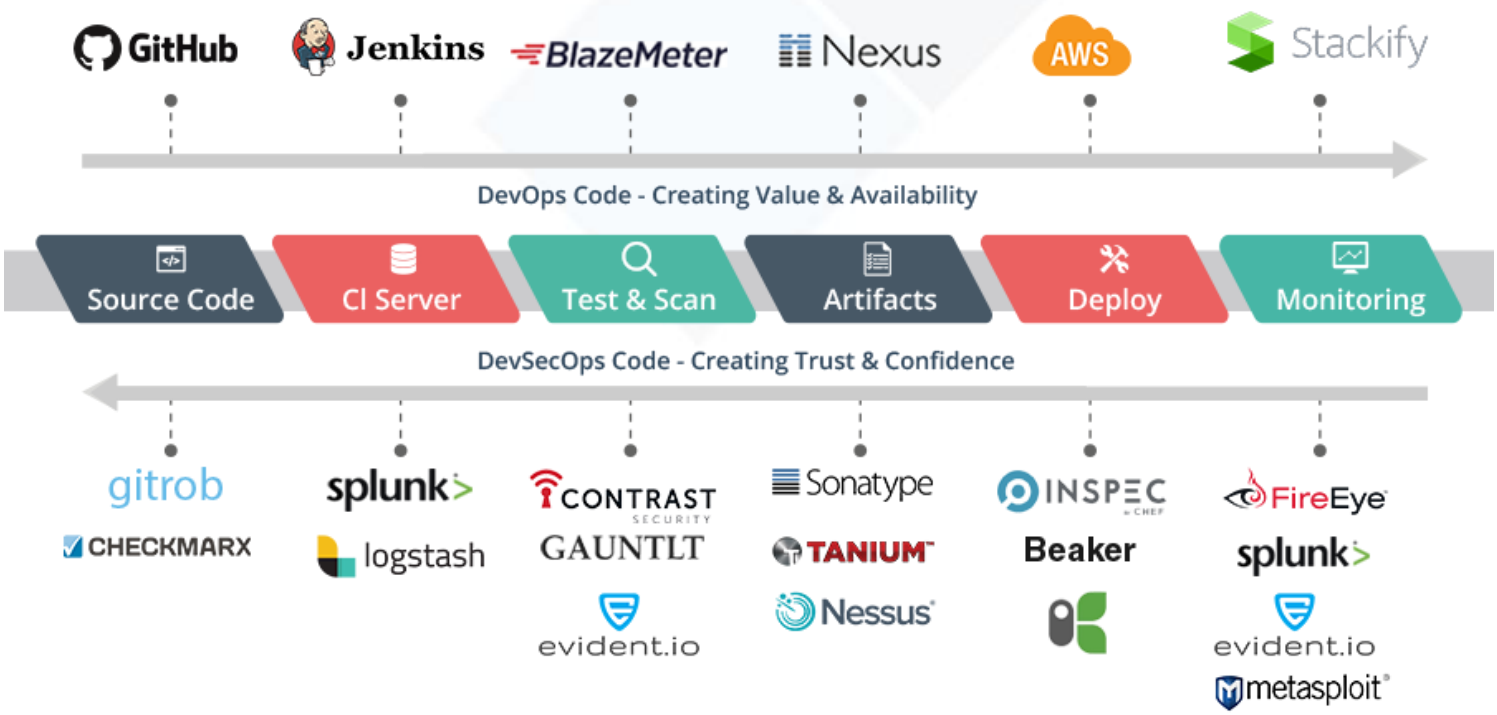
The DevOps movement has pushed for and succeeded in breaking down barriers and silos within organizations dividing teams into specialized functions of Development and Operations. DevOps enables organizations that embrace the movement and culture to be more competitive by enabling faster, more reliable software releases by leveraging automation to replace manual processes involved in shipping software.

A side effect of this speed is that security tools and processes need to move at the same pace to keep up. The idea driving DevSecOps or Rugged DevOps is to bake the security testing of the application under development into the process used to ship it. Automation of these processes takes people out of the chain and puts them in a different capacity. Instead of people being the process, tools and automation are the process and people monitor and respond to process failures. Thus, combining the strengths of both computers and people.

The image below lists some of the tools leveraged to automate the security testing and auditing of a DevOps pipeline.

Some types of tools that exist already for security testing are:

- **Cloud infrastructure** – Tools to scan your cloud infrastructure and configuration of resources against best practices for that cloud. Examples are Azure Advisor and evident.io
- **Automated security tests** – There are now frameworks that let you write security tests for applications just like the traditional unit and integration tests. Gauntit is a framework that is gaining popularity to build and automate tests like these.
- **Static code analysis** – This set of tools can scan your codebase and open source libraries and find potential vulnerabilities. VeraCode is a popular tool to perform these kinds of analyses.



- **Runtime security analysis** – This set of tools runs alongside/within your application in production and can help identify and prevent security issues in real time. Contrast Security is one such tool that provides these features.
- **Vulnerability scanning for Containers** – Tools like Clair from CoreOS help you to scan your container images for vulnerabilities. The API interface for Clair makes it easy to add this to custom build pipelines.

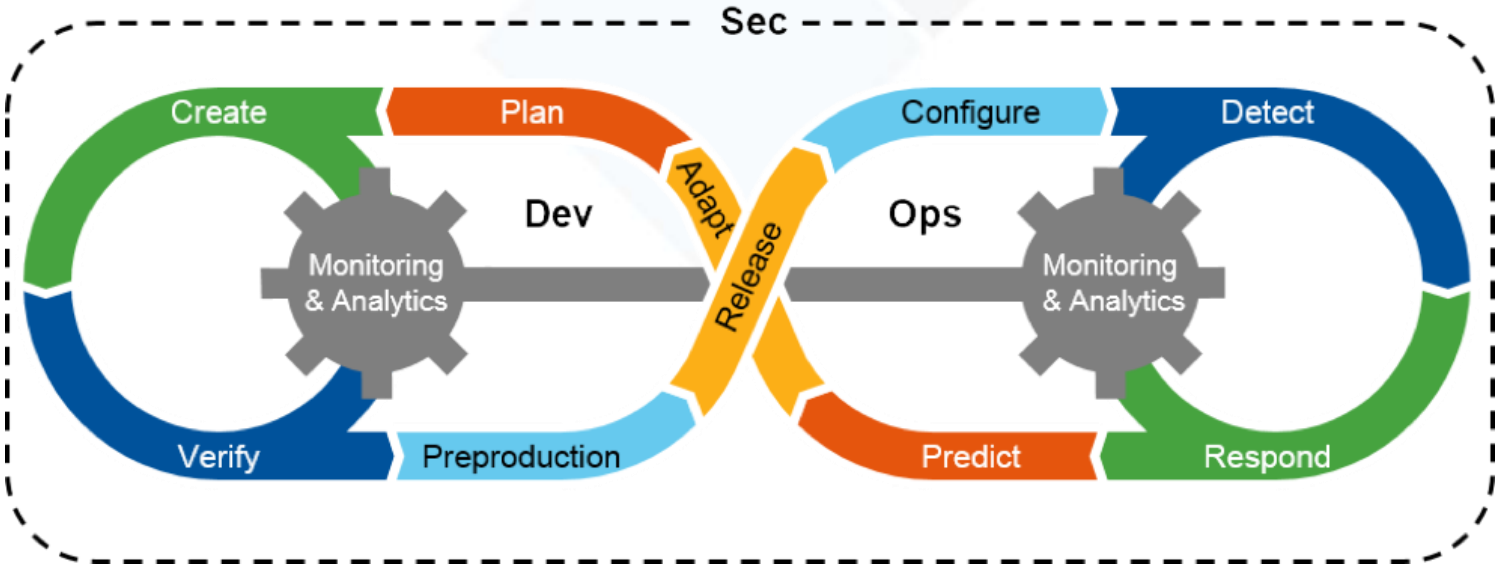
You can have a look at- https://github.com/devsecops/awesome-devsecops#tools for a more comprehensive list of DevSecOps tools for each stage of the SDLC.

Best Practices for Implementing DevSecOps in Your Organization

Automated and Programmable Security Controls

Just like DevOps relies on fast short feedback loops, DevSecOps also uses short feedback loops with the difference of having continuous monitoring and analytics at the core of both the development and operations ends.

Information security architects must try to incorporate security controls with a minimum to no manual configuration at every stage of the cycle in such a way that they are transparent and don't impede the hard-won agility DevOps brings while managing risk and regulatory compliance needs. This means that any security controls added need to be capable of automation to fit into DevOps toolchains. Automation brings in two benefits immediately. Automation reduces the risk of misadministration and mistakes which are the two leading causes of security breaches and unexpected operational downtime. Secondly, automation removes people from the process and makes them monitors of the process responding to process failures allowing a speed boost for security activities and making sure that the agility of DevOps environments isn't impacted.



Source: Gartner (September 2016)

When a security platform's capabilities like IAM, firewalling, vulnerability scanning, AST (application security testing) are available through a programmable interface, the integration and automation of these controls are easy within an automated DevOps pipeline. Security teams within an organization now can set policies which can be standardized and applied organization wide.

Specific best practices:

Choose security and management vendors who:

- Fully API-enable their platform services and expose 100% of functionality via APIs
- Provide explicit support for common DevOps toolchain environments, such as Chef, Puppet and similar automation tools
- Provide explicit support for containers and container orchestration and management systems (which are not necessary for DevSecOps, but help streamline service delivery from development into production).

Separation of Duties and Enforcement via RBAC and IAM

A major job for auditors and security architects is making sure there is a clear separation of who can do what in addition to where and when especially in terms of service deployment and development. Even within a single team responsible for a service there are clearly defined roles people take. The idea is not to lock down or unnecessarily hamper the individuals from doing their jobs but to give them the minimum set of permissions they need to achieve their goals. This will mean that they will be highly empowered within the areas they are responsible for. The scope and grant of capabilities can be managed by linking existing IAM systems and defining roles for each unique stage (development, staging, and production).

Specific best practices:

- Connect tooling to existing IAM systems to pull down identities and permissions, e.g., LDAP and Active Directory. Allow enforcement of security policies in tools and monitor all access to tooling and activities.
- Define unique roles for environments, e.g., development vs. production. An ideal situation is where all changes within production happen via audited and verified scripts with no people involved in making direct changes themselves.
- Make teams responsible and allow audits for changes to their service via a trust and verify basis. Verification can be achieved through audit logs and development changes logged into something, such as SCM, e.g., Git, Subversion, ClearCase.

Simplify Risk and Threat Modelling for Applications

A standard best practice for DevSecOps is having at least a basic risk based threat model. An easy way to start is having a simple questionnaire so development teams can assess the risk of the service they are developing at a high level. A sample questionnaire can include questions like:

- Is sensitive data being handled?
- What type of sensitive data?
- Are communications being encrypted?
- Is data at rest being encrypted?

Developer training should be conducted for basic security best practices and periodic communication around changes to these policies need to be circulated.

A list of basic security policies can include things like input sanitization, encryption of communication and data.

Specific best practices:

- Train developers in secure coding best practices and to write resilient code that sanitizes input and blocks common attack patterns, such as buffer overflows, SQL injection and cross-site scripting.
- Develop a simple threat-and-risk model assessment tool and implement it as a part of the planning and design process. Base the level of threat modelling on the risk of the application. Applications handling sensitive data or directly accessing the internet should require deeper threat modelling and collaboratively involve information security.
- Plan to mask, de-identify or synthesize data used in development for testing. Do not use raw sensitive production data in development.

Scan Custom Code, Applications, and APIs

Any custom code being written should be scanned for possible security vulnerabilities during development. Current tooling for traditional static application security testing (SAST) and dynamic application security testing (DAST) are not suited for the scale DevSecOps needs. They are too heavyweight and need a qualified security professional to run them.

There is a new breed of lightweight tools that integrate directly with a developer's IDE and allow for a quick check of security as a code is written. In addition to tools like these, automated scanning and security testing software should be a part of the continuous testing pipeline.

Interactive application security testing (IAST) is a great option if the platform the application is being developed on supports instrumentation (Java, .Net, PHP). IAST is a great fit for the highly-automated testing needed for DevSecOps.

A substitute for IAST is application security testing (AST) tools. Tools that can be driven via automation should be preferred.

Specific best practices:

- Evaluate and adopt IAST for applications that support it, and favour solutions using self-inducers for automated testing.
- Plan to fully automate any traditional static or dynamic tools or services that are used. For example, DevOps toolchain scripting tools can invoke automated testing. Do not make developers leave their native environment and toolchains.
- If SAST and DAST solutions are used, require vendors to support differential scans that test only the modified code and downstream-impacted modules.

- Acknowledge and accept that having zero vulnerabilities isn't possible. Reduce false positives (albeit with a risk of higher false negatives) and trim the output of AST tools and services to focus developers first on the highest severity and highest confidence vulnerabilities. Favour AST scanning tools and services that use machine learning and collective intelligence to trim results to only the highest confidence results.
- By policy, don't allow custom code with known critical vulnerabilities to enter production. Accept that vulnerabilities that represent lower levels of risk may or may not be addressed in future iterations. Approaches that identify and accept manageable risk are necessary.
- Work with DevOps managers to measure and motivate development teams to produce code with fewer vulnerabilities. Make security metrics a part of code quality metrics and hold development teams accountable.

Scan Open Source Dependencies for Issues

Most modern applications can be described as assembled vs. being built from scratch. Developers rely on a multitude of Open Source libraries and frameworks to accelerate the building of their application. This is a problem when there are vulnerabilities with these open source libraries and frameworks. All dependencies should be scanned and vetted for vulnerabilities during the build process and flagged for review and remediation.

Specific best practices:

- Prioritize OSS software module identification and vulnerability scanning in development.
- Scan all applications, system images, virtual machines and containers in development for unknown, embedded or vulnerable OSS components in the operating system, application platform and in the application itself.
- Implement an "OSS firewall" to proactively prevent developers from downloading known vulnerable code from Maven, GitHub and other OSS code repositories by policy.

Scanning for Vulnerabilities and Configuration at the Source – Development

The previous sections scan for discovered and known vulnerabilities in custom code and Open Source dependencies. As development continues and packages are built and integrated, it becomes important to scan the entire content of images (vms, amis or containers). This scanning should be built into the build pipeline and should be automated. These scans should also target the scanning of the configuration of the OS and application platforms and compare them to the best practices for that platform's secure configuration and hardening. An estimate from Gartner notes that through 2020, 99% of vulnerabilities that are exploited will continue to be known for at least one year. Finding these issues at the source during build time eliminates them from reaching production.

Specific best practices:

- Architect DevOps processes to automatically scan the contents of all system images, including the base OS, application platform and all containers for known vulnerabilities

and configuration issues as part of the continuous integration process. By policy, don't allow systems to leave development with known critical vulnerabilities.

- Require developers to remove unnecessary modules and harden all systems to industry standard best practices.
- Integrate with anti-malware scanners (such as VirusTotal), network sandboxing and algorithmic malware detection (such as Cylance) to scan systems to ensure the malicious code hasn't been introduced to the image during the development process.

Expand the Definition of Sensitive Code to Include Scripts, Recipes, Templates, and Layers

DevOps promotes a process of Infrastructure as a code. This allows you to version audit and automate the deployment and configuration of infrastructure, essentially making your infrastructure programmable. The first strategy talks about making security tools programmable.

Since infrastructure is being treated as a code, these artefacts need to have security coding principles applied to them. This means that the templates, scripts, recipes, and blueprints need to be secured and audited.

We've discussed how automation can reduce the chance of a mistake, but a poorly written script can magnify a mistake if released into production. An example of this is a recent S3 outage that was caused by an engineer mistakenly removing production servers instead of replicas. This action through the script caused Amazon's S3 service to grind to a halt. The script didn't have checks or rate limiting around how many servers it could affect.

Incidents like these drive home the need to make sure that configuration files and scripts, like source code, need to be scanned for mistakes, possible vulnerabilities, and excessive risk.

Earlier we discussed how high levels of automation can reduce the chance of a mistake. All configuration that can be expressed in text files should be held in a central repository like Git that allows changes to these configurations to be recorded. Git can record not only what change was made but also when and by whom. Eventually, all infrastructure should be treated like source code with version control rollback audit, logging and alerting based on usage. Making this the organizational standard means that no changes to infrastructure can be made without being recorded and audited. This process will be valuable to auditors and get their buy-in in implementing a DevOps expansion to not just application but also infrastructure.

Specific best practices:

- Ensure that DevOps teams have implemented good version control practices and tools to maintain clear accountability and traceability for all the application software that is deployed into the live environment.
- Extend the scope of the version control and automated deployment tools to the configuration, infrastructure setup and monitoring configuration.

- Use automation scripts to deploy to the staging environment for final tests (may be an automated test in advanced DevOps environments).
- Scan scripts for errors and embedded risk, such as embedded credentials, encryption keys, API keys and so on, that represent a significant and avoidable risk.

System Integrity and Configuration Compliance in Production

Let's move the focus of this paper to the best practices in production. The priority for any production system is that the infrastructure and services installed and running are what we need and are configured correctly.

Any tampering with the images against standards introducing vulnerabilities should be detected and isolated. We should measure as many system elements as we can (H/W, virtualization, images VMs, Containers). This measurement should extend to the validation of any container assembly layers used in container management solutions.

Specific best practices:

- Implement system integrity measurement on systems as they are booted, including the hardware-based root of trust measurements of the basic input/output system, bootloader, hypervisor and OS on systems you own.
- Store VMs at rest encrypted and hashed, if VMs are used in the DevSecOps workflow. Verify against tampering at boot.
- Use a container management system (if containers are used) that supports hashing or other techniques to measure and verify system integrity when loaded.

Use Whitelisting on Production Systems, Including Container-Based Implementations Leveraging Whitelisting in Production

One of the most powerful security controls for a workload is whitelisting and subsequent monitoring and enforcement of all its interactions. The use of whitelisting to prevent any thing not explicitly listed as safe prevents the running of malicious files and workloads.

Whitelisting can extend well beyond just what is allowed to run on a system. This technique can be used to whitelist network connectivity, user access, administrative access, and file system access. Historically this has been difficult to achieve but with the automation of infrastructure through DevOps configuration tools this is now straightforward. The declarative nature of DevOps templates lend themselves really well to whitelisting.

Specific best practices:

- Disable runtime-signature-based anti-malware scanning and implement a whitelisting model on server workloads. Antivirus scanning provides little or no value on well-managed servers and is a waste of resources in a DevSecOps environment.
- Automatically configure whitelists from the declarative sources of DevOps toolchains and containers.

- Require vendors to support whitelisting approaches for containers if containers are used.

Don't Assume Perfect Protection

Organizations in this day and age face advanced and targeted attacks and preventing them all isn't possible. It's better to assume they will be compromised and engineer to minimize the impact of such a compromise. This means everything in your environment needs to be monitored continuously and any unusual behaviour that is indicative of a breach should trigger alerts and automated responses. Machine learning and advanced analytic techniques to identify patterns and deviations can be leveraged here.

Specific best practices:

- Design for pervasive monitoring of critical applications, user logins/logouts, transactions, interactions, network activity and system activity.
- Use the monitoring data to establish baselines of "normal" for the application in order to detect meaningful deviations. Share monitoring data across DevOps or product teams, platform teams and security operations center teams, as unusual activity may be caused by a hardware failure, software failure, bug, insider threat or attack.
- Deploy deception and decoy services automatically to more easily identify attackers as these technologies mature over the next several years.

Restrict and Lock Down Access to Production Infrastructure and Services

Making sure that automated tools are the only way to make changes in production allows the standardization of remediation and makes all actions auditable. The exclusive use of automated tools also lets you perform rollback of any changes attempted. Rapid iteration and immediate feedback for problems and vulnerabilities at the Development end means your security posture is improved.

Specific best practices:

Information security architects should collaborate with DevOps teams to:

- Restrict changes to only being made via automated tools and scripts. Disable remote administration via Secure Shell (SSH) and Remote Desktop Protocol (RDP) to force access via APIs and scripts.
- Adopt an immutable infrastructure mindset (where possible) and automate all changes to the environment using DevSecOps-style workflows. Out-of-date workloads should simply be replaced with newer images in an automated, systematic way.
- Require privileged access management systems to manage credentialed access in the rare cases when direct administrative access is needed.

Containers – Security Limitations

While containers aren't necessary for a DevOps

transformation, they are extremely popular because of the consistency and streamlining they provide for developers as their code moves from development to production. Containers do introduce several security issues that need to be acknowledged. Containers share a common OS; this means that isolation is provided by the OS and not the hypervisor. Network traffic is visible to all containers on the same host OS without the addition of any additional tools. This means that any attack that is successful on the OS kernel will expose all containers on that host.

These are the reasons why it's recommended to use containers on workloads of similar trust levels and using hypervisors as isolation by running the containers within VMs. The use of lean stripped down speciality OSes developed to run containers is also a recommended measure.

Conclusion

DevSecOps aims for a goal of having security checks and controls applied transparently and automatically within a rapid-development automated DevOps pipeline. Shifting security left to start at development makes sure that DevSecOps is effective and it follows the journey of service throughout its lifecycle.

Above all, successful DevSecOps initiatives must remain true to the original DevOps philosophy: teamwork and transparency, and continual improvement through continual learning.