How to Fuel Your DevSecOps in AWS

Dave Shackleford

SANS

Analyst Program

Whitepaper

How to Fuel Your DevSecOps in AWS

Written by **Dave Shackleford**

May 2021

aws marketplace

Introduction

As the pace of software development increases, there's a real need for organizations and their security teams to ensure application security is embedded in all phases of the development and deployment life cycle, as well as in the cloud during operations.

The software development life cycle (SDLC) has moved to an agile methodology that prioritizes collaboration and frequent, small updates to application stacks. Organizations should define and publish standards for code quality and security, as well as application workload configuration, so that all teams have something to measure throughout the entire application life cycle. Ideally, cloud workloads should be as locked down as possible, running a minimum of services, and configuration requirements should be revisited to help ensure resiliency of any cloud-based infrastructure.

In the spirit of increased collaboration, security teams should integrate with the developers who are promoting code to cloud-based applications. Security teams should impress upon development and operations that the series of tests and quality conditions they can apply to production code pushes will not slow down the development process. Security teams should work with quality assurance (QA) and development to define parameters and key qualifiers that must be met before any code can be promoted.

Much as with other areas of security, the responsibility for application security varies depending on the cloud model in use. In a software-as-a-service (SaaS) model, the provider is entirely responsible for application security in almost every case. With a platform-as-a-service (PaaS) model, the provider supplies the underlying systems and templates, so it has a significant degree of control and responsibility. However, the consumer has responsibility for any applications it develops, and that responsibility extends to security. And with an infrastructure-as-a-service (IaaS) model, entire workloads and their contents, including application components, are the responsibility of the consumer.

In addition, security teams will need to determine which operations tools can be integrated into the application pipeline and identify areas and controls that might need to be updated or adapted to work in a continuous integration and/or continuous deployment model. Protection for application workloads requires a dedicated commitment to security at many levels of any organization. A sound governance model that includes collaborative discussions about code quality, system builds, architecture and network controls, identity and access management (IAM), and data security are all critically important to developing the standards for controls and security posture.

In this paper, we will describe how to build a successful security automation strategy, beginning with a discussion on visibility into application development into cloud production environments. Next, we dive into what it takes for security teams to gain continuous visibility throughout all phases of the DevOps pipeline. We also explain application visibility in runtime and conclude with a discussion of automation in DevSecOps.

Visibility into Application Development

Application visibility relies on tracking events and behaviors at scale as workloads communicate within the cloud environment as a whole, as well as tracking the local application logs on individual systems and containers. True application visibility often requires feeding events into event management and SIEM platforms, and such systems have been well adapted for cloud environments, often via API integration. When using such platforms in the cloud, security teams need to enable logging and monitoring of code and system/container image repositories to help ensure that all access is tracked. Enabling AWS CloudTrail[1] logs within Amazon Web Services (AWS) will enable analysts to collect and monitor all access to images and code, as well as changes or other interesting events related to repositories.

Scanning code for vulnerabilities is another important task that should be wholly integrated into the DevOps pipeline. When code is checked into a repository, automated scans should be triggered that send development and security teams a report on vulnerabilities detected. Organizations should define application code security standards that outline the level of severity of vulnerabilities that is acceptable for continued build and deployment, with the level dependent on the criticality of the applications and data involved (as well as application exposure). An example of a cloud-native code review and assessment service is Amazon CodeGuru. Amazon CodeGuru Reviewer Security Detectors can improve code security and resilience by identifying security vulnerabilities in the OWASP Top 10[2] and recommending best practices to remediate critical issues and bugs during application development. During test and operations phases, Amazon CodeGuru Profiler monitors the runtime behavior of applications to detect anomalies as well as patterns of anomalous behavior. These capabilities provide significant visibility into both code development and application testing and operations.

Visibility into the DevOps Pipeline

Security teams should consider a number of different security controls and areas of emphasis to gain visibility into all phases of the DevOps pipeline. When integrating into a cloud-focused application development model, security teams should focus on the following:

- **Code security—**Ensure code is being scanned for vulnerabilities and monitor all code updates and changes, including details such who checked code in or out, and when.
- **Build and packaging tools—**Integrate monitoring for tools used to automate builds and packaging. These phases are optimal times to look for configuration errors and potential vulnerabilities in images planned for deployment.
- **Orchestration platforms—**For deployment and runtime maintenance, monitor and carefully secure orchestration tools such as Kubernetes.

The key security focal areas for all phases of a DevSecOps pipeline are described in the following sections.

---

[1] This paper mentions solution names to provide real-life examples of how cloud security tools can be used. The use of these examples is not an endorsement of any solution.

[2] OWASP Top Ten, https://owasp.org/www-project-top-ten

Code/Development

Ideally, your organization follows secure coding practices. Security and development teams should:

- **Discuss standards for languages and frameworks to make sure risk is acceptable before deployment.** This can be a tall order, because secure coding and development practices are still not commonplace.
- **Evaluate static code analysis tools that can automatically scan code that is checked in.** Many leading solution providers offer highly automated scanning tools in AWS Marketplace. You can also take advantage of native tools such as Amazon CodeGuru.
- **Ensure the code is secured within repositories.** Define check-in and check-out procedures and define and enforce role-based least privilege access controls.

Cloud providers often have options for code storage and management that include authentication with strong identity management and robust logging/tracking of activity. AWS CodeCommit is a fully managed source control service that hosts secure Git-based repositories and encrypts all files in transit and at rest, integrates with AWS Identity and Access Management (IAM) for controlling privileges and access to code stores, and logs all activity in AWS CloudTrail. Additionally, AWS CodeCommit has a wide range of APIs that can enable automation and integration with third-party static code analysis tools for code analysis and review by security teams. Code can be automatically scanned upon check-in, and bug/vulnerability reports can be sent automatically to the appropriate teams.

Build and Packaging

Code and workload stacks for cloud applications should incorporate automated and intelligent security controls, including:

- Validated code
- An approved build architecture and controls
- Automated build testing for compiled code

Besides the aforementioned automation and security controls and processes, automated reporting that goes to the proper parties for review will ultimately contribute to a more effective vulnerability management program across the environment. Much as with the previous phase of development (coding), the build phase can often be securely implemented within cloud provider environments. AWS CodeBuild is a fully managed, continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy. Managing encryption of build artifacts is critical, and AWS CodeBuild integrates with AWS Key Management Service (KMS). AWS CodeBuild also integrates with AWS IAM for control over privileges to builds and compiled code, and logs all activity in AWS CloudTrail.

Packaging is the phase of application development where the build is updated with additional software packages, whether open source or from in-house repositories. It's important for development and security teams to audit open-source modules for flaws and to discuss methods to protect code repositories automatically. Specialized package vulnerability scanning services and tools can be easily integrated into build and packaging features to assess libraries, packages, and binaries in images for vulnerabilities and potential exposure. Native services, such as AWS Systems Manager, can also be used to manage package repositories and secure build images with up-to-date patches and libraries.

Set a regular schedule for threat and vulnerability updates with the development and operations teams and incorporate it into defined processes.

Testing

The testing phases of a deployment can usually be highly automated and carefully monitored. With regard to testing, the following are best practices:

- **Use both static and dynamic tools depending on builds, and define security test cases ahead of time.** This way, a spirit of collaboration and cooperation can aid teams in defining acceptable outcomes that meet policy.
- **Automate all testing scenarios and tools, and teach developers/QA engineers to run them.** Security teams should hand off tools to the application developers wherever possible and not insert themselves into testing processes. Ideally, security should only perform penetration tests and continuous monitoring activities regularly once policies and standards are defined.

Using build testing tools, such as Test Kitchen and Vagrant, can simplify internal policy validation, before they are pushed and in an ongoing fashion. Vulnerability scans, penetration tests, and routine checks can provide testing visibility and validate policies' effectiveness by ensuring that only required network ports are open; that secrets such as credentials, encryption keys, and tokens are secured; and that privileges are assigned properly in the application architecture.

All elements of a configuration standard or expected vulnerability posture should be continually validated and assessed using automated orchestration tools and platforms. Many third-party dynamic application scanning and pen testing service providers are fully integrated into the cloud. These tests can be run automatically upon build check-in or image update, or manually as needed, with fully automated reporting sent to the appropriate teams.

Deployment

In the deployment phase, security teams should focus on the following areas:

- **Documentation—**Note any outstanding bugs, and document plans to fix them and when.
- **Communication—**Coordinate with development and operations teams to instantiate any controls needed for remediation or stop gaps.
- **Life cycle—**Ensure that an approved policy for bug remediation is in place and monitored for future release cycles.

> **Before pushing applications and systems out the door, fix any bugs that meet the levels of severity your team defined.**

Deployment involves more on the operations side. Ideally, controlled and automated deployments will be coordinated and controlled by operations, with input from the application development teams involved. In terms of security, this means:

- Nothing new is added or changed once approved builds are ready.
- Deployment is done to the appropriate location/endpoints.
- Deployment is performed over a secure channel for cloud (TLS/SSH).
- Checks exist to ensure that a failed deployment rolls back.

At this stage it is critical for security teams to be invested and involved. They should establish secure network channels for any deployment activities, which likely involves the use of dedicated circuits, such as AWS Direct Connect, VPN tunnels using IPSec, and/or secure certificate-based HTTPS with strong cryptographic TLS implementations. Image validation should also occur at this phase, which will heavily rely on automation and a combination of vulnerability scanning and host-based agents that can validate all libraries, binaries, and configuration elements used in the application workloads. For some of these tasks, orchestration engines such as Puppet, Chef, and Ansible, as well as cloud-native tools such as AWS OpsWorks and AWS Systems Manager, can reliably and securely handle the configuration and assessment of application images.

Operations and Runtime

This stage primarily focuses on protection of applications with tools such as network access controls and web application firewalls (WAFs), as well as monitoring, logging, and alerting. Define security use cases for the following production operations:

- Specific events that should trigger alerts
- Specific events that should trigger automated remediation
- Event severity that may determine follow-up priority

For starters, teams should define attributes of deployments that can be monitored continuously. Examples of quick wins for monitoring include the following:

- Types of instances allowed to be deployed (e.g., size and build)
- Image tags that determine deployment eligibility
- AWS IAM users and roles invoked in operations

These attributes should be known and relatively inflexible, and they can easily be used as simple trigger points for alerting or even automated rollback or preventative actions. For example, if an instance type of `m1.small` is deployed and when `t2.micro` is the only approved type, this event could be cause to shut down the workload entirely—or, ideally, prevent deployment in the first place. Cloud-native or third-party WAFs, such as the AWS WAF and others, can easily be set up to block unauthorized application events such as SQL injection, cross-site scripting (XSS), and others. In addition, they can perform manual or automated blocking of IP addresses based on threat intelligence that incorporates reputation analysis. WAFs can also generate detailed logs, which security teams can then stream back to a central analysis engine, such as a SIEM platform.

Additional Considerations

Along with core security controls and practices in each major phase of a modern development pipeline, security teams should consider some additional topics and concepts. A critical aspect of managing security in a cloud environment is the need to carefully limit and control the accounts and privileges assigned to resources. All users, groups, roles, and privileges should be carefully discussed and designated to resources on a need-to-know basis. The best practice of assigning the least-privilege model of access should also be applied whenever possible. Any privileged accounts, such as root and the local administrator accounts, should be monitored very closely (or, ideally, disabled completely).

In addition to establishing privilege management in configuration definitions, application development teams should ensure that no sensitive material, such as encryption keys or credentials, is stored in definition files, on systems that are exposed, or in code that could be exposed. Because encryption and data protection strategies are increasingly automated along with other development activities, it's critical to ensure protection of the proverbial keys to the kingdom at all times. In the cloud, this can be easily accomplished with a variety of native tools, such as AWS KMS and AWS Secrets Manager, or third-party tools such as Hashicorp Vault.

Strong privilege management is a necessity in fast-moving application pipelines. Integration with secrets management tools and a granular IAM policy engine such as AWS IAM is crucial, along with federation capabilities and integration with directory services such as Active Directory. Security teams should help define the appropriate least-privilege access models for all stages of application development and deployment, and then implement this in a centralized tool/service whenever possible. A comprehensive privilege management and IAM implementation strategy increases operational oversight of users, groups, and permissions, so a single policy engine should be used if at all possible.

Visibility of Cloud Applications in Runtime

Logs and events generated by services, applications, and operating systems within cloud instances should be automatically collected and sent to a central collection platform. Automated and remote logging is something many security teams are already comfortable with, so organizations implementing robust cloud security designs simply need to ensure that they are collecting the appropriate logs, sending them to secure central logging services or cloud-based event management platforms, and monitoring them closely using SIEM and analytics tools. In the case of containers and container management tools, many new and well-known providers of vulnerability scanning and configuration assessment services have adapted their products to work in the cloud, granting deep visibility into both container image configuration and runtime event monitoring.

Traditional Server Workloads

Logs from your instance OS should be collected, just as you would in your own data center. This includes Syslog, Windows Events, and all the other logs you'd normally collect for security and operational reasons. The basic mechanics of generating logs and sending them somewhere might be the same, in general, depending on your deployment model. However, due to volume and cost, sending them to an in-cloud log collector or event management platform likely makes sense. This process is distinct from logging within the CSP environment, where you will focus on API calls and access to the admin console for your cloud environment.

To enable consistent workload monitoring and logging, many organizations will need to create and enable a central cloud log repository to store logs generated within workloads. Although there are many ways to accomplish this, AWS has a unique Amazon CloudWatch agent that can be installed into Amazon Elastic Compute Cloud (Amazon EC2) workloads. It forwards Syslog and other standard events to a dedicated Amazon CloudWatch logging group. From there, these logs can be parsed and analyzed, or streamed to a different event management and monitoring solution through streaming services, such as Amazon Kinesis Data Firehose.

For many organizations, the data export costs associated with large volumes of workload logs might prove somewhat prohibitive to simply sending all logs back to on-premises data collectors and SIEM tools. While this may work with a small volume of cloud services and workloads, large organizations will eventually want to enable cloud-native log collection and analysis tools instead.

Containers and Container Management/Orchestration

Containers are rapidly becoming a common means of quickly deploying application workloads in both internal and cloud environments. Containers are created on a shared OS workload, and both the runtime container image and the underlying OS platform need to be secured and maintained, much as with other images described earlier. Having a secure repository for container images such as Amazon Elastic Container Registry (Amazon ECR), as well as orchestration tools that can be used for starting, stopping, and managing container deployments securely, such as Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS), is important for any enterprise using containers in the cloud. Encryption and IAM controls for images, as well as strong logging for all activities, should be security priorities. Logging within containers and their underlying host OS should be considered mandatory when possible. It is important to make sure that no secrets are embedded in containers and container files.

Serverless Applications and Security

A final type of technology that many application development teams are employing is serverless, which offloads the entire workload (container and OS instance) to the provider's backplane, enabling developers to create microservices applications that only require application code to be uploaded and operated within the cloud provider environment. Serverless security should involve static code review (numerous third-party providers can integrate into serverless environments such as AWS Lambda to scan the code), privilege and permission control over all serverless applications with IAM, and complete logging of all serverless application updates and execution using tools such as AWS CloudTrail.

Orchestration Services

Kubernetes, while extremely powerful, is also known for being highly complex. Bad actors can potentially gain unauthorized access to Kubernetes environments in numerous ways, ranging from exposed APIs to admin consoles with weak IAM or misconfigured security policies.

Logging in Kubernetes can be complex (handled in numerous places, and not always simple to enable). Logs for each container can be sent to traditional files in **`/var/log`**, but Kubernetes best practices suggest not doing this due to how scattered this may be with ephemeral containers and performance issues. Instead, direct container logs to **`stdout`** and **`stderr`**, and Kubernetes will aggregate them for analysis and access through **`kubectl`** logs commands.

Kubernetes API Server audit logs are likely the most interesting to security teams. Although Kubernetes audit logs are not enabled by default, it's a good idea to leverage this feature to debug issues in your cluster. Kubernetes events are key to understanding the behavior of a cluster. This is done via an audit policy, and you can enable logs at different levels for varied types of resource requests (see Table 1).

Within cloud environments, enabling a continuous monitoring and visibility strategy is paramount through the entire DevOps pipeline, but there are more tools and services available than ever to help accomplish this goal. APIs are plentiful to enable automated logging and security scanning of code, packages, images, and runtime behaviors.

Table 1. Resource Requests for Enabling Audit Logs

Resource Request	Definition
`None`	Don't log events that match this rule.
`Metadata`	Log request metadata (e.g., requesting user, timestamp, resource, verb), but not request or response body.
`Request`	Log event metadata and request body but not response body.
`RequestResponse`	Log event metadata, request, and response bodies.

Automation in DevSecOps

One of the most important goals of DevSecOps is to help automate security controls within the pipeline as well as within the cloud operating environment. By implementing automation in cloud development and deployments, organizations are working to eliminate manual, repeatable tasks; reduce or eliminate human error; and simplify complex or time-consuming tasks.

Automating vulnerability scans of various types within the pipeline and in cloud operations is an achievable goal. Cloud providers offer a plethora of APIs for integration, allowing automated scans to kick off at several key inflection points:

- When code is checked into a repository or updated
- When container or workload images are checked into a repository
- When packages are loaded into a registry
- When new workloads start running in production

Another common focal area for security automation is automated event generation and monitoring of cloud resources. Organizations implementing DevSecOps should:

- Collect the appropriate logs, such as workload logs and cloud control plane logs like AWS CloudTrail events.
- Send logs to a cloud-based event management platform that can be used to trigger automation sequences and tools.
- Monitor events closely using SIEM and analytics tools.

While event data should be monitored using SIEM or other analytics tools, the key to automation often lies with the central collection engine in the cloud environment, allowing security teams to initiate AWS Lambda functions and alerting that help automate detection and response capabilities. In AWS, this breaks down as follows:

- **Phase I: Learn—**In this phase, you monitor for events occurring in the environment. With AWS, this would likely come from AWS CloudTrail logs, Amazon Virtual Private Cloud (Amazon VPC) Flow Logs, Amazon CloudWatch logs, and so on.
- **Phase II: Trigger—**Based on some pattern matching, using Amazon CloudWatch alerts or even a SIEM, you then trigger some sort of follow-up action.
- **Phase III: React/Respond—**The final phase is the actual action triggered during the automation. This could be an AWS Lambda function that performs an action, a vulnerability scan, or an alert sent via SNS or other method.

Let's consider a simple example of automation sequences for DevSecOps.

Example: Pipeline Scanning

As part of a cloud-centric DevOps pipeline, a developer checks code into AWS CodeCommit. Upon check-in, an automated static code scan is initiated using Amazon CodeGuru or a third-party solution through API integration. The results of this static analysis are sent automatically to DevOps and security teams, who can verify that all code checked in meets standards for acceptable levels/severity of bugs before being promoted to the build and packaging phases.

New container images are pushed to Amazon ECR, where automated vulnerability scans are triggered to look for known issues with packages and installed components. The results of this scan are automatically sent to DevOps and security team members, as well. Based on the organization's policy, only images with severity vulnerabilities of medium and lower can be assigned tags identifying the image as acceptable for production operations.

When new workloads in Amazon EC2 are started, the Amazon Inspector agent is already installed as a part of AWS CloudFormation template definitions, and an automated vulnerability scan is initiated immediately. If any high-severity vulnerabilities are detected in AWS CloudTrail events (by sending these to an Amazon CloudWatch logs group and initiating an AWS Lambda function that looks for this event detail), the workload is automatically terminated. This demonstrates event-driven automation of response actions and processes using a triggered serverless function that can then perform remediation as needed. Similar capabilities are available through AWS Config rules that detect misconfiguration.

Next Steps in Security Automation

When building a security automation strategy for the DevOps pipeline, teams should consider the following:

- **Security of the pipeline—**All platforms and services that constitute the DevOps pipeline (e.g., code and image repositories, and build tools) should be monitored and carefully controlled through privilege allocation.
- **Security of code in the pipeline—**All code should be automatically scanned upon check-in and modification for vulnerabilities.
- **Security of what comes out of the pipeline (builds and images)—**All builds and images should be automatically scanned for package and component vulnerabilities.
- **Cloud operations—**All running cloud assets should be monitored through cloud fabric logging, and event-driven automation can then be used to remediate or alert on issues.

For application pipelines today, a plethora of tools is available from cloud providers and third-party companies to help automate strong security controls through the entire development and deployment process. A strong governance structure is critical to ensure all stakeholders are involved and on board with the new tools and processes needed, and security operations teams will need to help define standards for code and images, as well as build strong protective and detective controls in the cloud environment.

About the Author

Dave Shackleford, a SANS analyst, senior instructor, course author, GIAC technical director and member of the board of directors for the SANS Technology Institute, is the founder and principal consultant with Voodoo Security. He has consulted with hundreds of organizations in the areas of security, regulatory compliance, and network architecture and engineering. A VMware vExpert, Dave has extensive experience designing and configuring secure virtualized infrastructures. He previously worked as chief security officer for Configuresoft and CTO for the Center for Internet Security. Dave currently helps lead the Atlanta chapter of the Cloud Security Alliance.

Sponsor

SANS would like to thank this paper's sponsor: