Modern App Development
and Enterprise DevOps Series

Microsoft

sogeti
Part of Capgemini

Securing Enterprise DevOps Environments

Table of contents

Summary 03

Introduction 05

Secure the developer environment 06

Secure the Enterprise DevOps platform environment 17

Secure the application environments 30

Secure Enterprise DevOps in practice 44

Closing thoughts 51

How Microsoft & Sogeti can help 52

Summary

Securing your DevOps environments is no longer a choice – **The hackers are shifting left too.** Nefarious and creative hackers have started compromising developer boxes, infecting release pipelines with malicious scripts, and gaining access to production data via test environments.

In this Ebook, you'll learn to fortify all three attack surfaces of enterprise DevOps environments and implement the culture changes necessary to thrive in our dangerous new world. We'll explore the ideal secure and regulatory-ready setup of Enterprise DevOps tools and practices, focusing on three specific areas:

Secure the developer environment

Secure the DevOps platform environments

Secure the application environments

To build a truly secure Enterprise DevOps platform, each environment needs careful attention. This Ebook will help guide you through the process—providing an example breach and detailing best practices to fortify your Enterprise DevOps security.

Below is a visualization of the developer, DevOps platform environment, and application environments covered within this Ebook, along with the potential threats for each respective environment. Notice how the connections between environments and to external integrations expand the threat landscape and lead to increased opportunities for hackers. Leveraging the strategies discussed in the Ebook, will help you properly prepare your enterprise to deal with next-generation threats from shift-left hackers.

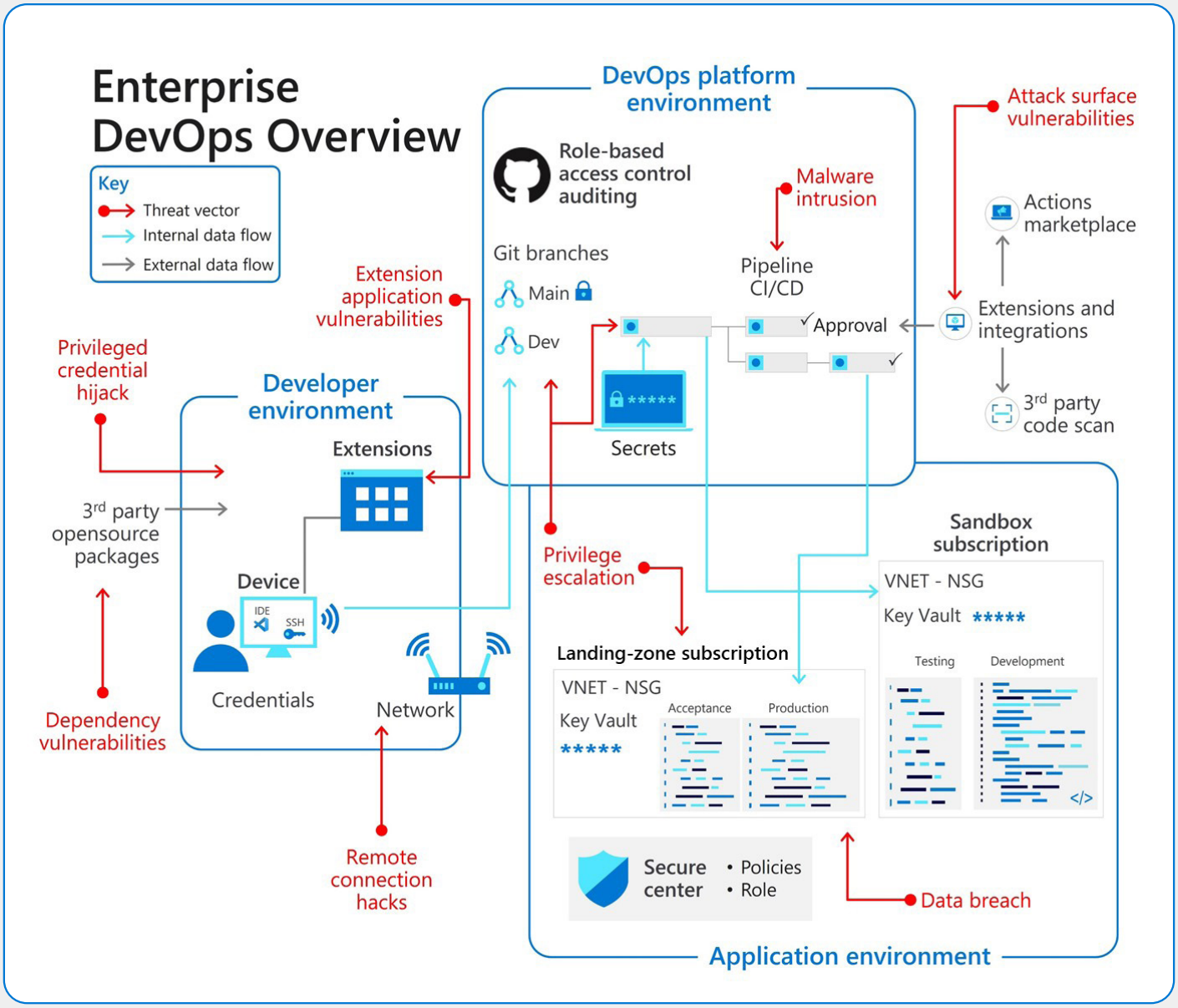


Figure 1 - Enterprise DevOps Environments Overview

Introduction

Shifting security left further into development is hardly a secret. Hackers now shift left too and use DevOps environments to gain access to the enterprise. These new attacks stretch across the enterprise and unlock new dangers. Some of these new attacks may inject malicious code, assume powerful developer identities, and steal production code, along with the typical breadth of cyber security breaches.

The attack surface of enterprise DevOps environments covers three areas: The developer environment, DevOps platform environment, and application environments—occurring throughout the application development, and maintenance lifecycle.

The developer environment includes everything a DevOps team member uses to produce code, test, and document, from the laptop to the software. As companies transition to a ubiquitous, work-from-anywhere styled approach, the control of these devices suffers greatly. Often, cyber security offices lack a consistent understanding of where and how the code is secured and built. Hackers are taking advantage, with an uptick in remote connection hacks and developer identity thefts.

Another key target for hackers is the DevOps platform environment. All tools that help enterprise DevOps teams to function represent key entry points for attackers from pipeline automation to code validation, and code repositories. A common example occurs when company code is infected by hackers before it reaches production systems and thereby passes through cyber security checkpoints.

Hackers have sieged production environments for decades and generally there exists cyber security practices to prevent them. However, now the environment has widened significantly to include all supply chain tools and products that enterprises incorporate into their system. One breach at a third-party open-source tool can now lead to a global cyber security pandemic.

This Ebook will guide you with best practices and platform capabilities to harden your enterprise's security and defend against shift-left hackers in all three environments. It also closely examines real world examples of hacks for each scenario. Additionally, your enterprise needs ways to adopt practices, improve knowledge transfer, and cross-functional team awareness. This Ebook will also help to realize secure enterprise DevOps in practice.

Secure the developer environment

Are your developers happy and productive? Developer velocity relies on developers' ability to work how and where they want, maximizing their impact on business outcomes. Generally, powerful, customizable machines top developer wish lists. And in most cases, developers often require root or administrator access on their work environments. These developer demands also run contrary to compliance regulations enterprises must adhere to, specifically the need to audit and control what is accessed and stored on 'private' employee environments.

Adhering to diverse developer needs taxes enterprises. The thought of unmanaged machines connected to the organizational network conjures up a nightmare for security teams, procurement, and the governance board. Even the best-case scenario, providing developers with default and hardened employee environments, creates disdain from both sides. This solution limits the capabilities of Enterprise DevOps team members and undoubtedly makes difficult to access developers unhappy.

Explosive growth in remote working has further complicated this challenge. With employees connecting from anywhere, these vulnerable Wi-Fi networks present an open door for cyber-attack. Physical loss or theft of a developer machine is also a major concern as no-one is safe when data is on the street.

Unfortunately, enterprise vulnerability extends to the integrations developers use in their development environments.

Unfortunately, enterprise vulnerability extends to the integrations developers use in their development environments. Many development tools are rich in their extensibility capabilities, featuring marketplaces with many unmaintained integrations. For example, IDE marketplaces often contain thousands of community-made extensions. And extensions endanger more than just main development tools and can be found in all kinds of tools developers use. In a worst-case situation, a malicious extension can result in a company-wide breach.

How do you give DevOps team members flexibility and control, without opening the door to malicious attacks? This is the fundamental challenge for many security offices today.

To update your Enterprise DevOps cybersecurity, implement the following measures to help secure the developer environment:

- Control the developer environment with a cloud environment
- Secure the developer environment with containers
- Configure least privilege access
- Limit who can change and approve code with branch security
- Adopt only trusted tools, extensions, and integrations

Let's dive into each practice with several technical explanations that highlight the necessity of each security principle.

In the visualization of the developer environment below, notice that the environment connects to the DevOps tools environment to affect the Git branches. It's also widening the environment surface through its connection to 3rd party open-source packages and application extensions. These extensions present new attack vectors for hackers in dependency vulnerabilities and extension application vulnerabilities.

In this chapter, we'll take a look at how to prevent hackers from compromising those connections along with defenses against privileged credential hijacks and remote connection hacks.

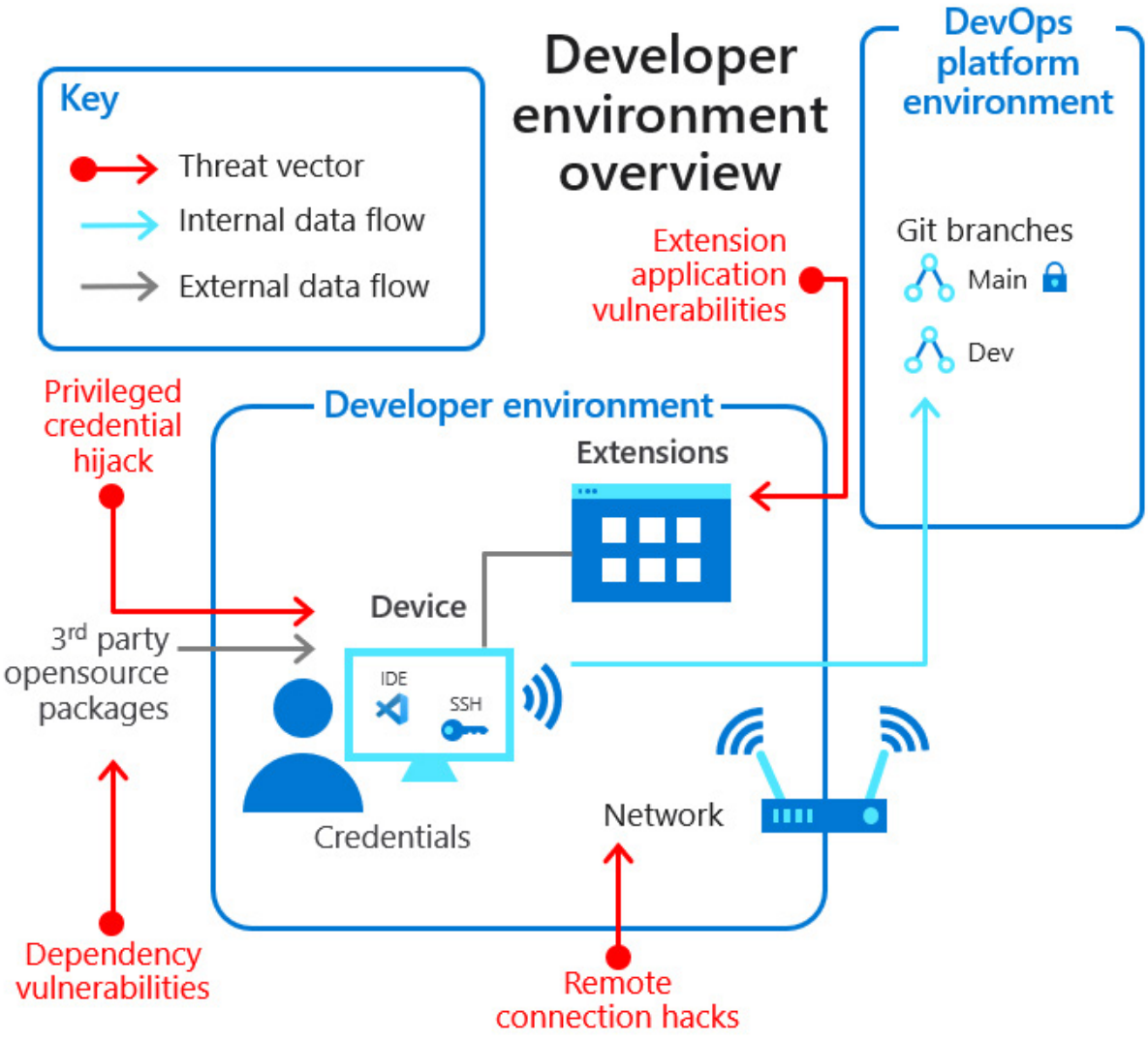


Figure 2 - Developer Environment Overview

The real life (hack)

March 2021 - Hackers Infecting Apple App Developers with Trojanized Xcode Projects

Attempts to steal customer information are reaching new levels of forethought. Developer machines have become a key attack vector for hackers looking to install backdoors and download entire system's source code.

What happened?

Many developers for Apple systems use the developer environment XCode, and this knowledge prompted hackers to produce a "XcodeSpy," a trojanized Xcode project. This is a tainted version of a legitimate, open-source project available on GitHub called TabBarInteraction that's used by developers to animate iOS tab bars based on user interaction. However, in this case, XCodeSpy installs a customer variant of the EggShell backdoor on a macOS computer along with a persistence mechanism.

Going further, this malicious project contained a backdoor which, after installed on the developer machine unlocks the ability to record information from the victim's microphone, camera, and keyboard. It's feasible that XcodeSpy could be targeted at a particular developer or group of developers, but with the capabilities to hinder other high-value victims. While only one US firm reported a confirmed payload of the EggShell payload there may be many unacknowledged cases where hackers are simply gathering data for future use.

Breaches often occur over time and hackers will group interesting targets and data for future campaigns. Every hour wasted without identifying malicious code is another hour where company data is being stolen. Here, the hackers most likely plan on leveraging the AppleID credentials taken in this operation to install malware with valid Apple Developer code signatures.

In this chapter, we'll take a look into how the attack could've been stopped with more secure practices.

Control the developer environment with a cloud environment

Centralized control and templates in a cloud environment form the backbone of efficiency and central management for developer environments.

Centralized control and templates in a cloud environment form the backbone of efficiency and central management for developer environments. For example, in the early days of cloud, Sogeti created its OneShare solution. This self-service portal allows DevOps teams to control enterprise environment needs via the cloud. Teams can create developer machines and test environments based on predefined templates on any public cloud. Self-service support enables tight cost control by initiating shutdowns, deletions, and improving security through opinionated templates and hardened images. Centralizing controls offers a layer of governance to DevOps teams when they must adhere to further developer requirements be it networking, access, credentials, and/or certificates.

Many enterprise DevOps teams turn to public clouds compute resource consumption for the control and almost unlimited power to flexibly meet business needs. Teams can recycle machines on a daily basis and build them up again from scratch automatically, giving hackers very little time to investigate the developer environment. When a new template is needed, new hardening requirements can be easily updated for all teams to start using.

Multiple technologies enable teams to manage, create, and control developer environments from the cloud to meet their needs. A solution like Azure Virtual Machines gives developers the freedom to create and switch the machines on and off themselves, to more supported and controlled scenarios via solutions like Sogeti OneShare or Azure DevTest Labs.

How to

Azure Virtual Machines (VMs) is one of several types of on-demand, scalable computing resources designed to meet the needs of Enterprise DevOps teams. Typically, VMs perform well in situations where you need more control over the computing environment. But, when creating a VM there are many elements to consider, prominently how it's created, and how it's managed.

An Azure VM delivers the flexibility of virtualization without having to buy and maintain the physical hardware that runs it. However, every VM requires maintenance tasks, such as configuring, patching, and installing the software that runs on it. Azure VMs offer a quick and easy way to spin up a developer environment with specific configurations required to code and test an application. For example, many enterprises use a Virtual Machine with Visual Studio on it to create and update secure environment policies. And Gen2 VMs on Azure can even Secure Boot using a virtualized TPM (i.e. no need for a hardware root of trust). This is a highly recommended feature to protect dev machines from unauthorized access. After VMs are set up correctly, teams can leverage solutions like Azure Policies to dynamically update policies with each new threat discovery.

There are two challenges when using Virtual Machines on the cloud for development and test activities. First, many team members may not be required to have the expertise to create, start, stop, and delete a Virtual Machine from the cloud—creating a knowledge gap. Another challenge is the awareness of cost control implications on Virtual Machine cloud usage.

To address the knowledge gap, one solution is to provide expanded ease-of-use through cloud management platforms. Sogeti's cloud management platform, OneShare, allows teams to manage and monitor application development environments, test environments and infrastructure resources in a systematic way. An integrated toolset on top of Microsoft Azure, OneShare gives developer teams the ability to load, use, manage and monitor their environments. Advanced capabilities grant enterprise DevOps teams granular control, which may include schedules that configure the start/stop of Virtual Machines and enforced geofences that switch on or off Virtual Machines when the team member enters a specified area.

To help DevOps teams learn and manage VMs varying cost, many solutions guide users with a visualized projection tool aligned to company usage. For instance, in the screenshot below, notice how Azure DevTest Labs delivers visualized usage trends and configurable costs thresholds that activate notifications and automatic shutdowns when needed.

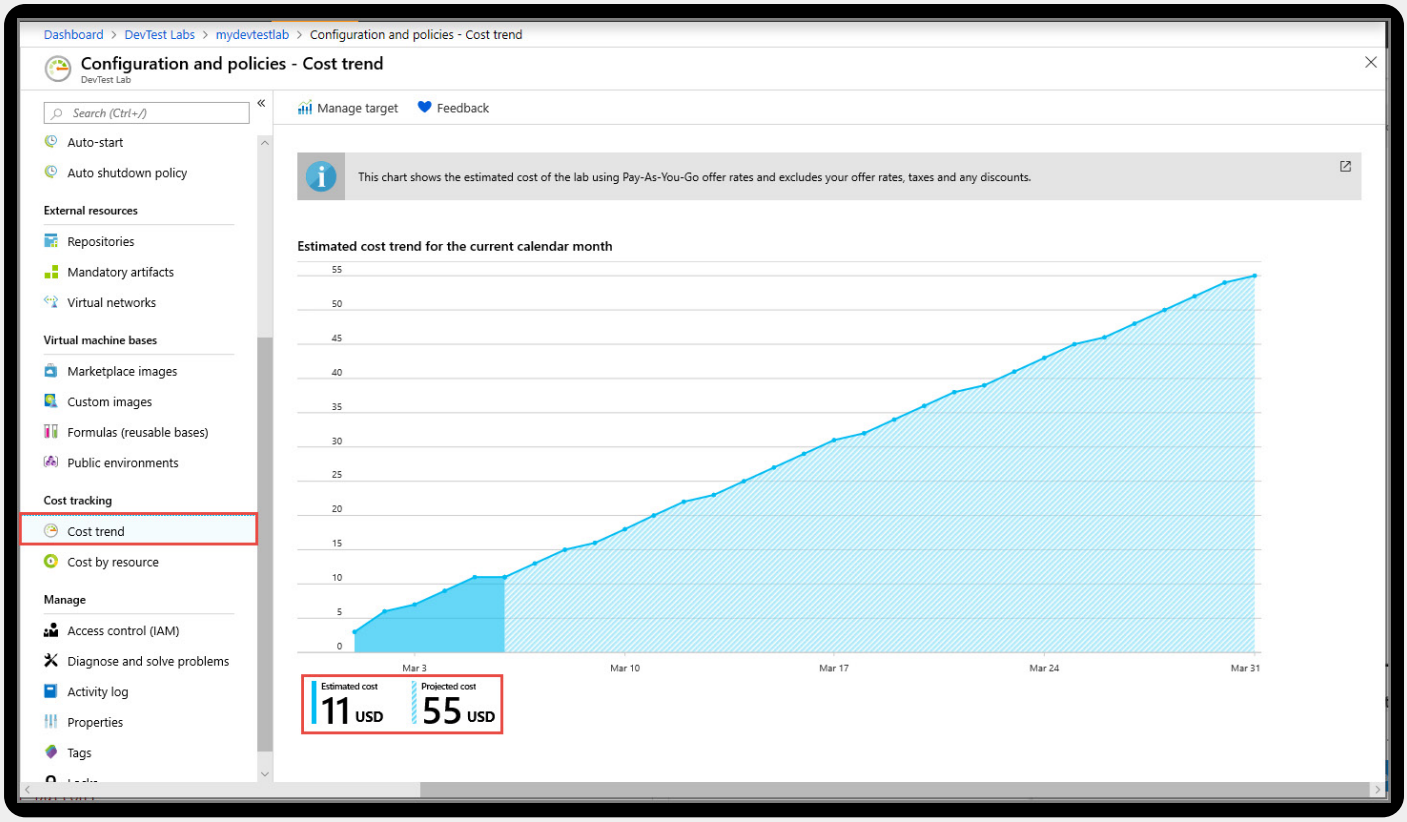


Figure 3 - Azure DevTest Labs Cost Visualization

Secure the developer environment with containers

While virtual machines may replicate hardened images or secure ARM templates with artifacts, they live and die in the cloud. With container technology, developers can run their development environment in the cloud and on their device. Although a developer machine in a container can be limited, the power and usages of local and cloud compute resources make them as powerful as Virtual Machines. These powerful, portable machines provide more secure options to enterprise DevOps teams looking to spin up developer environments, least of all being easy recreation. While there is some security risk associated with containers, they do provide teams with options to limit the scope of their development environments and projects. This limited scope provides teams with segmented and reproducible environments that keep damage contained and update automatically.

Ultimately, the scale and control of containers means DevOps teams can easily meet developer needs while ensuring an added level of security. It's now relatively easy to create a complete collection of developer environments with containers and maintain them in a container registry with all the necessary security boundaries and validations around it.

For both this situation and the Virtual Machine scenario, remember that base images must be maintained and hardened at the organization level and not be taken from public resources.

It's now relatively easy to create a complete collection of developer environments with containers and maintain them in a container registry with all the necessary security boundaries and validations around it.

How to

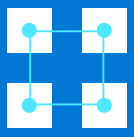
Innovations with container development environments provide developers so many new ways to build, configure, and craft efficient and secure applications. GitHub recently launched GitHub Codespaces, enabling developers to create an environment in a container with the UI accessed through a browser. Visual Studio Code lets teams create developer environments based on a .devcontainer (https://code.visualstudio.com/docs/remote/containers) folder which lives next to the application code.

There are so many ways to start building developer environments on top of containers, make sure to select the service that best fits your needs. For example, vscode.dev and github.dev provide file editing access to your files within a GitHub repo in a developer environment. While this is a not a full developer environment it offers a way to access and edit files securely.

Configure least privilege access

Most developers want to have administrator privileges to the environments and tools they use, but with great administrative privilege comes a great security challenge.

Although developers may study and believe in their ability to notice malware, phishing, or other breaches on their environments, this is not always the case. First, the large size of the developers' environment threat surface makes it unrealistic for a developer to maintain omnipresent system knowledge. When a developer environment outfitted with administrator access to all systems is compromised by a hacker is finally discovered, precious remediation may have already passed. This opportunity is no secret to hackers as software developers are the role most targeted by hackers.



The use of least privilege and just-in-time access is not only a good practice for system administrators and end users, but also for Enterprise DevOps

The use of least privilege and just-in-time access is not only a good practice for system administrators and end users, but also for Enterprise DevOps. It's best that team members maintain only minimal access to environments for the shortest amount of time required. This covers not only the administrator access rights on the main device, but also access to the DevOps tools, release pipelines, code repositories, environments, secret stores, databases, and so on. For Enterprise DevOps teams, the base requirement is a connection to the organization's identity store. Using identity federation for integrating with SaaS environments avoids duplication of identities on third party platforms, to reduce the risk of its exposure.

In addition, to access SaaS-based DevOps tools, secure practices for Enterprise DevOps teams span access to the code repository, either via SSH, HTTPS, or a personal access token, and a configuration that can control on which developer device (local, cloud, container) the systems code can be downloaded (cloned). One thing to consider is that it is strongly recommended not to use a personal access token for source code access. When you access a SaaS-based environment, you need to have clear instructions for how the access principles dictate who can clone repos and from which device. OneDrive is a good example of this where you can synchronize folders on a local and cloud perspective. When an unmanaged device attempts this, it's prevented, and unmanaged devices are never allowed access.

How to

Generally, there are three ways for a developer to connect to a SaaS environment, either via HTTPS with an identity, a personal access token, or connecting with a SSH key. In all scenarios a connection needs to be made with the enterprise identity store. With GitHub, except for GitHub EMU users, your identity is and always will be your public identity, a connection with the enterprise identity store is required to control access via SSO, just like in the screenshot below.

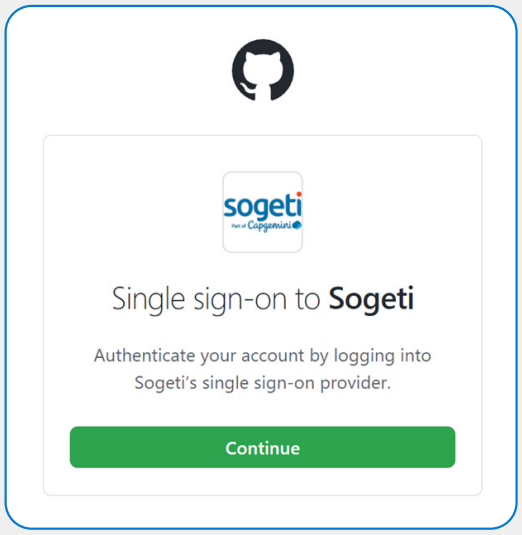


Figure 4 - Screenshot of Sogeti Single Sign-On

GitHub recently implemented EMU user accounts which are standardized and synchronized with each user's corporate identity. This means that GitHub usernames, emails, and display names are defined directly in your company's identity store, making it easy for users to identify with their collaborators, even if they've never met face-to-face.

With an SSH certificate authority, organizations or the enterprise account lead may provide SSH certificates members can then use to access the resources with Git securely. An SSH certificate is a mechanism for one SSH key to sign another SSH key. If you use an SSH certificate authority (CA) to provide members of your organization with signed SSH certificates, you can add the CA to your enterprise account or organization to allow people to use their certificates to access organization resources.

GitHub Enterprise Cloud supports SSH certificates to give enterprises and organizations more control over how their members access their repositories. Admins can upload the public key of their SSH CA and begin issuing certificates for their members to use for Git authentication. Certificates can only be used for accessing repositories belonging to that enterprise or organization. Additionally, admins can require members to use certificates when accessing their repositories like in the screenshot below.

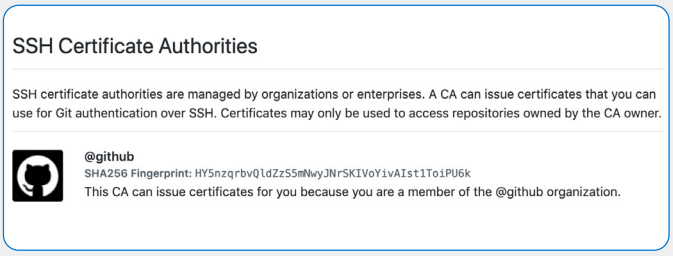


Figure 5 - Screenshot of SSH Certificates within GitHub

Lastly, for access, tools like VS will have built in support for shared windows identities and VS Code will defer to a Git credential manager. The usage of a Git credential manager is strongly recommended to harden access to your code.

Limit who can change and approve code with branch security

It is crucial that the repository used to store your system code is adequately secure. The worst-case scenario occurs when hackers gain access to the code repository and modify code without the teams noticing and then study how the system is secured.

To prevent this, implement a branching strategy (see image below) to establish control over code changes, protected branches with code reviews give DevOps teams control over code changes and auditing advances.

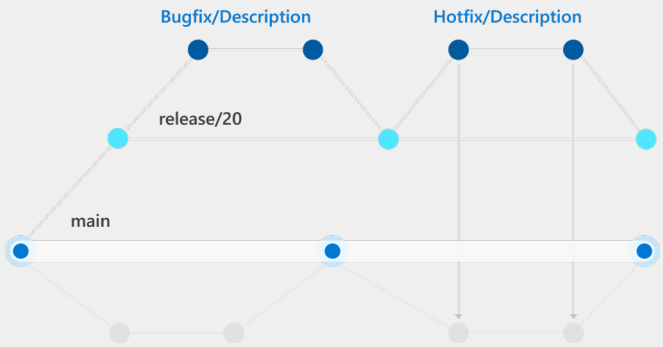


Figure 6 - Branching Strategy Visualization

The visualization above articulates a controlled flow of changes that delivers a clear chain of command and blueprint for addressing code changes. There are many different approaches for the branching strategy, but one commonality is that protected branches serve as the source for new releases to production.

There are many different approaches for the branching strategy, but one commonality is that protected branches serve as the source for new releases to production.

How to

The control mechanism of branching strategies is in the approval workflow. Protected branches require certain validations, reviews, and approvals before they can accept changes. Approval authorizations should be controlled by the administrator of the GIT repository.

One option is to create a branch protection rule to enforce certain workflows for one or more branches, such as requiring an approving review or passing status checks for all pull requests merged into the protected branch shown in the example below.

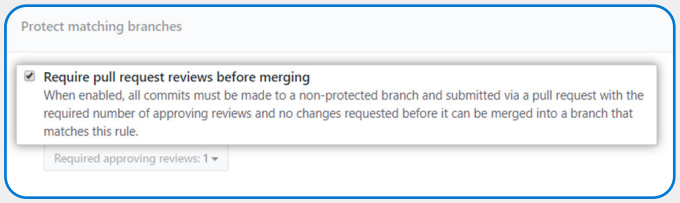


Figure 7 - Pull Request Policies Example

Branch policies help teams protect their important branches of development. Policies enforce your team's code quality and change management standards. This example below showcases how to set reviewer requirements on a Git branch within an Azure repo.

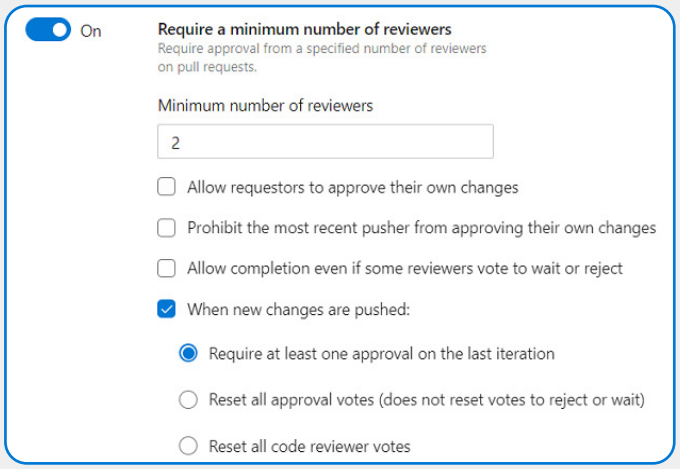


Figure 8 - Branch Reviewer Policies Screenshot

Adopt only trusted tools, extensions, and integrations

Extensibility in integrated developer environments (IDE) is so productive that it's essentially a mandated feature. Every developer relies on the ability to leverage and curate extensions within the marketplace of that specific IDE to make their optimal work environment. The VSCode marketplace shown below has thousands of extensions to make developer's lives easier. However, whenever your teams are interested in adopting new tools or extensions, sometimes the most important aspect is verifying the trustworthiness of publisher. Make sure your teams take the time to integrate only tools from both trusted marketplaces and publishers.

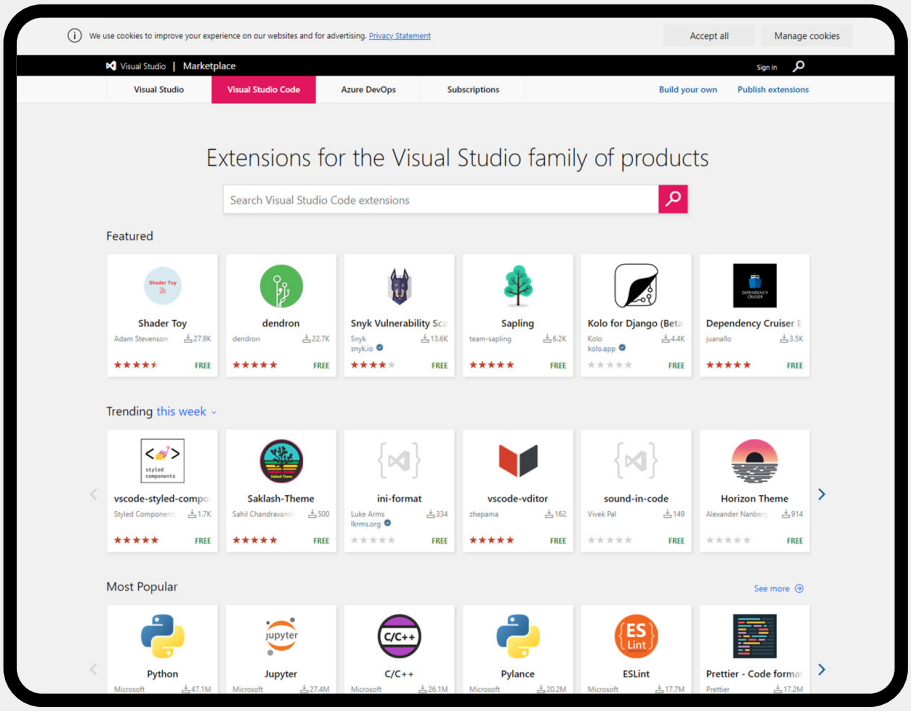


Figure 9 - Screenshot of the VSCode Marketplace

Make sure your teams take the time to integrate only tools from both trusted marketplaces and publishers.

Most IDE extensions require approving certain privileges to function. Often this is a file with read permissions on the system to analyze code. Some extensions also require connections to cloud environments to function, as is common in many metric tools. Approving all these additional functionalities on top of the IDE opens up enterprises to more threats. It's best to set up secure practices to control the extension use to limit the attack surface of developer environments.

The guide on the next page offers some ways that the VSCode marketplace helps to ensure the safety of the extensions within it. Take this approach and review the safety of each extension you're allowing into your codebase.

Can I trust extensions from the Marketplace?

The Marketplace runs a virus scan on each extension package that's published to ensure its safety. The virus scan is run for each new extension and for each extension update. Until the scan is all clear, the extension won't be published in the Marketplace for public usage.

The Marketplace also prevents extension authors from name-squatting on official publishers such as Microsoft and RedHat.

If a malicious extension is reported and verified, or a vulnerability is found in an extension dependency:

1. The extension is removed from the Marketplace.
2. The extension is added to a kill list so that if it has been installed, it will be automatically uninstalled by VS Code.

The Marketplace also provides you with resources to make an informed decision about the extensions you install:

- **Rating & Review** – Read what others think about the extension.
- **Q & A** – Review existing questions and the level of the publisher's responsiveness. You can also engage with the extension's publisher(s) if you have concerns.
- **Issues, Repository, and License** – Check if the publisher has provided these and if they have the support you expect.

If you do see an extension that looks suspicious, you can report the extension to the Marketplace with the **Report Abuse** link at the bottom of the extension **More info** section.

Figure 10 - Marketplace Policies Overview

It's also important to track, on a developer machine how many extensions are used and what the maturity of those extensions. This helps to understand the potential attack surface your allowing into your own environments. Try to incorporate only VS Code marketplace extensions that come from verified publishers. When you're installing extensions of third-party application with VSCode, you must regularly check the extensions that you're running with the command line: code --list-extensions --show-versions. These steps will allow a better understanding of all the extensible components you're running in your developer environment.

Secure the Enterprise DevOps platform environment

Modern enterprises rely on DevOps platforms for deployment, including the pipelines and production environments that developers require to be productive. Traditional application security methods didn't consider the increased attack surface that these pipelines and production environments represent for hackers. But now, with hackers shifting left and targeting these upstream tools, a new approach is needed to secure DevOps platform environments.

But now, with hackers shifting left and targeting these upstream tools, a new approach is needed to secure DevOps platform environments.

How and why are pipelines currently targeted?

Pipelines and production environments are extremely attractive to hackers because of how removed these environments are from standard application security practices and processes. These environments typically require high-level access credentials, so when compromised, these credentials allow deep and meaningful access to attackers.

While new attack types are being found each day, some of the most common attack vectors for pipelines are extracting runtime variables or argument injection. Pipelines can also be targeted by scripts that retrieve service principles or credentials from pipelines. A third scenario is the misconfiguration of personal access tokens, opening a world of problems and allowing anyone with the key to access the DevOps platform environment.

On top of pipeline and personal access token scenarios, enterprises need to verify the security of their third-party tool integrations, which are often required to fulfill automation system requirements. Most integrations, like test frameworks and static/dynamic code testing require access to the code, often read-only, but sometimes also require write access. A vulnerability or misconfiguration in the third-party integrated tool can lead to serious security incidents.

To help defend against these incidents, your teams will need to fortify the DevOps platform environments. Start by ensuring granular control and audit trails are available across each environment. You'll also need to implement least privilege access when you can and ensure the right level of read/write permissions. The goal is to build a secure setup, minimizing exposure of secrets and parameters.

In this chapter, we'll review how to implement the following aspects for securing the DevOps platform environment:

- Ensure no team member has access to secrets and certificates
- Equip every DevOps platform environment with audit trails
- Secure the software supply chain
- Automate scans for Infrastructure-as-Code (IaC) templates
- Automate approval workflows
- Allow only verified DevOps tool integrations

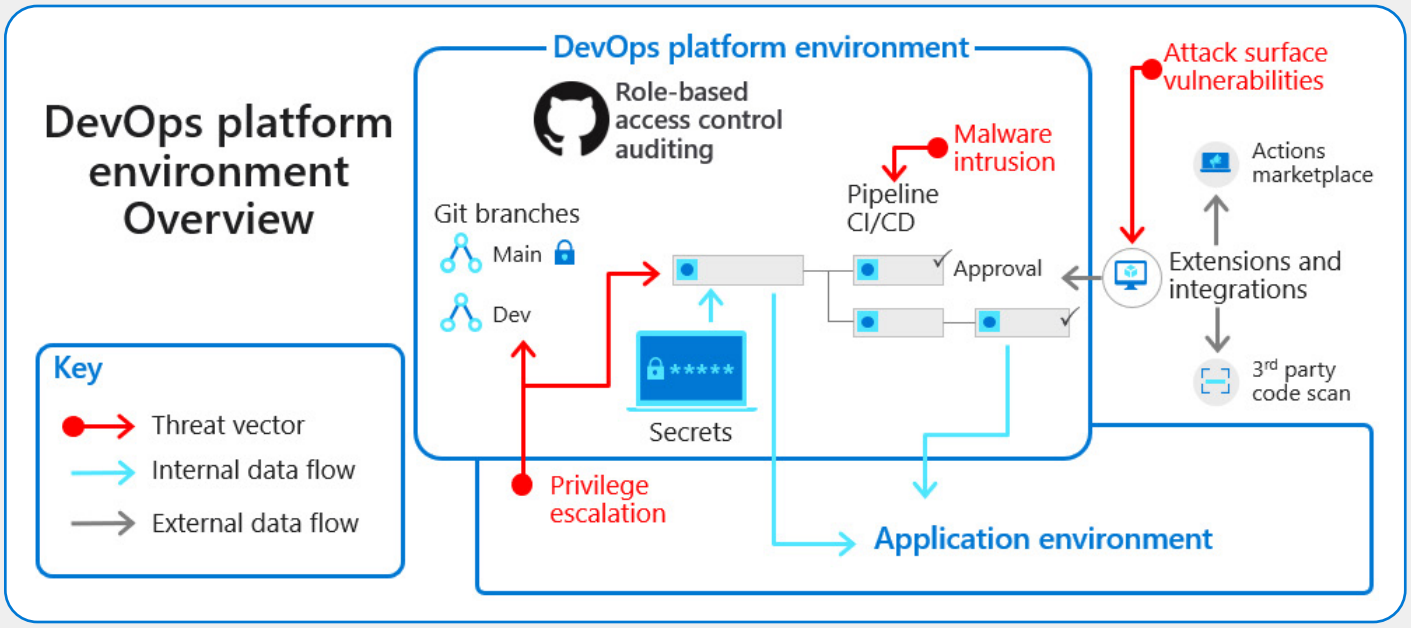
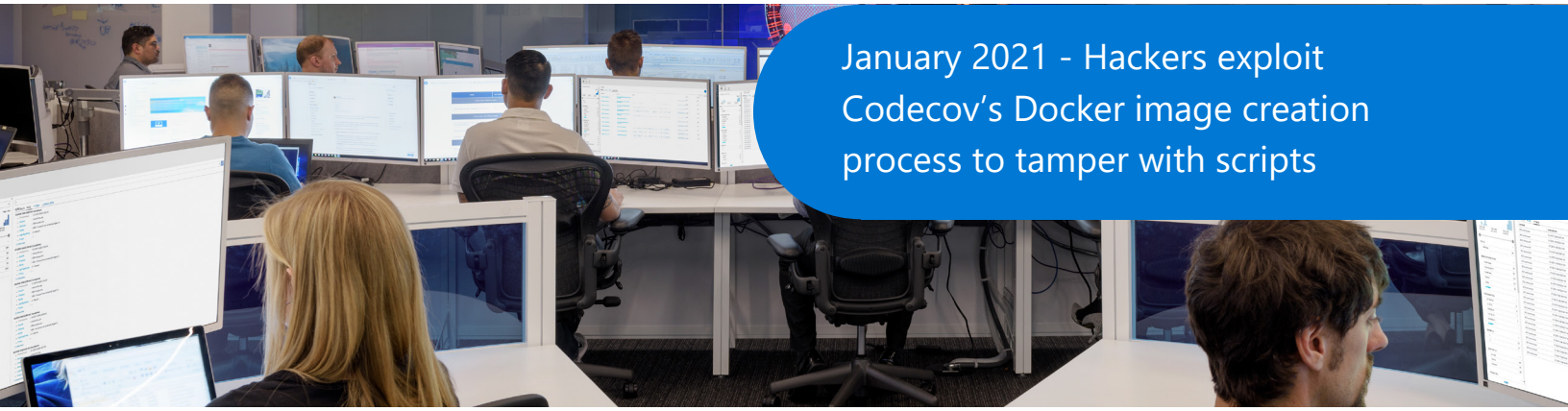


Figure 11 - DevOps Platform Environment Overview

In the visualization of the DevOps platform environment above, notice that the environment connects to the application environment as well as to CICD pipeline extensions. These extensions present opportunities for hackers to engage in privilege escalations stemming from the application environment along with increased attack surface vulnerabilities from the extensions. For the DevOps platform environments, it's critical to defend against those threats along with malware intrusions.

The real life (hack)



January 2021 - Hackers exploit Codecov's Docker image creation process to tamper with scripts

Based in San Francisco, Codecov offers code coverage and software testing tools, so that the DevOps cycles can deploy healthier code. However, in this example we'll take a look at how an attacker was able to exploit an error in Codecov's Docker image creation process to tamper with the Codecov Bash Uploader script.

This type of supply chain hack is aimed to not only exploit Codecov software, but also to use the organization as a springboard to compromise a huge number of corresponding customer networks. The end-goal in this scenario is the theft of credentials, tokens, and keys running through client CIs, as well as "services, datastores, and application code that could be accessed with these credentials," according to Codecov. In addition, URLs of origin repositories using the Bash Uploaders may have been exposed.

With over 29,000 enterprise clients, many startups, and open-source community projects, Codecov's damage radius immediately grew to an enormous scale. The initial exploit in the Bash Uploader impacted Codecov's full set of "Bash Uploaders" including the Codecov-actions uploader for GitHub, the Codecov CircleCI Orb, and the Codecov Bitrise Step.

What happened?

To recover, Codecov imposed strict security processes including rotating internal credentials and pulled in a third-party cyber-forensics firm to conduct an audit. Codecov's also adopted a new monitoring system to prevent such "unintended changes" from happening in the future.

Throughout this chapter, we'll take a look at even more ways enterprise clients can prevent a downstream hack from occurring in the future, such as tighter access control and securing the software supply chain.

Ensure no team member has access to secrets and certificates

Avoiding a catastrophic breach can be as simple as effective secret management. Every stage in the application lifecycle now uses secrets and certificates that must be stored securely. The graphic below visualizes effective secret management where every secret, every password, access token, and certificate must be managed.

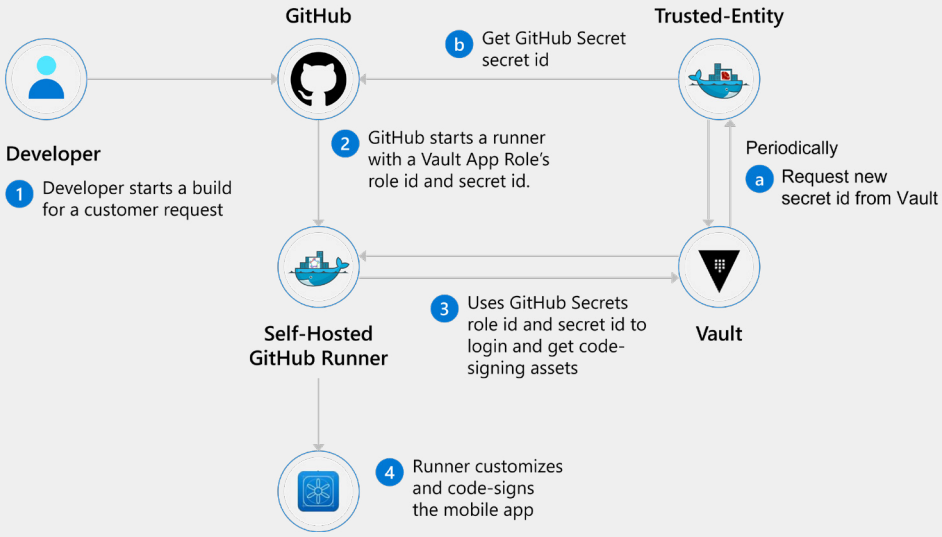


Figure 12 - Secret Management Example

The secure development practice here is for the Enterprise DevOps team to develop always as if it is an open-source project. Ensure that teams are storing no secrets anywhere in the code or on team environments, rather they should be kept within key vaults.

Ensure that teams are storing no secrets anywhere in the code or on team environments, rather they should be kept within key vaults.

How to

You'll need to start by selecting a key vault that best fits your enterprise, Azure Key Vault is a great choice for storing secrets and certifications. Access to this vault is organized via network security and/or via access policies for Azure Active Directory entries.

Azure Key Vault helps in securely managing and storing the secrets, keys, and certificates. With Azure Key Vault, it's easy to set up granular access control to pair keys in the vaults to Azure identities, groups, or roles. This helps ensure control of the secrets while removing any direct access to the secrets within the application. Once users enroll, it automatically renews certificates from supported public Certificate Authorities. Azure Key Vault also supports monitoring and auditing of key usage with Azure logging. Later we'll discuss the importance of sending logs into security information and event management (SIEM) solutions for more analysis and preventative threat detection.

Teams can limit CI/CD deployment access with Azure Key Vault through constraints on access for the antifactory to upload the build artifacts and restrictions on database access. This solution also helps fortify security by generating encryption keys, only allowing authorized users to maintain read/write access. A big advantage is, that you can rotate, change, update these secrets in the Key Vault, without worrying about exposure. Experiment with Azure Key Vault variables (shown below) to harden data confidentially through encryption and limit exposure.

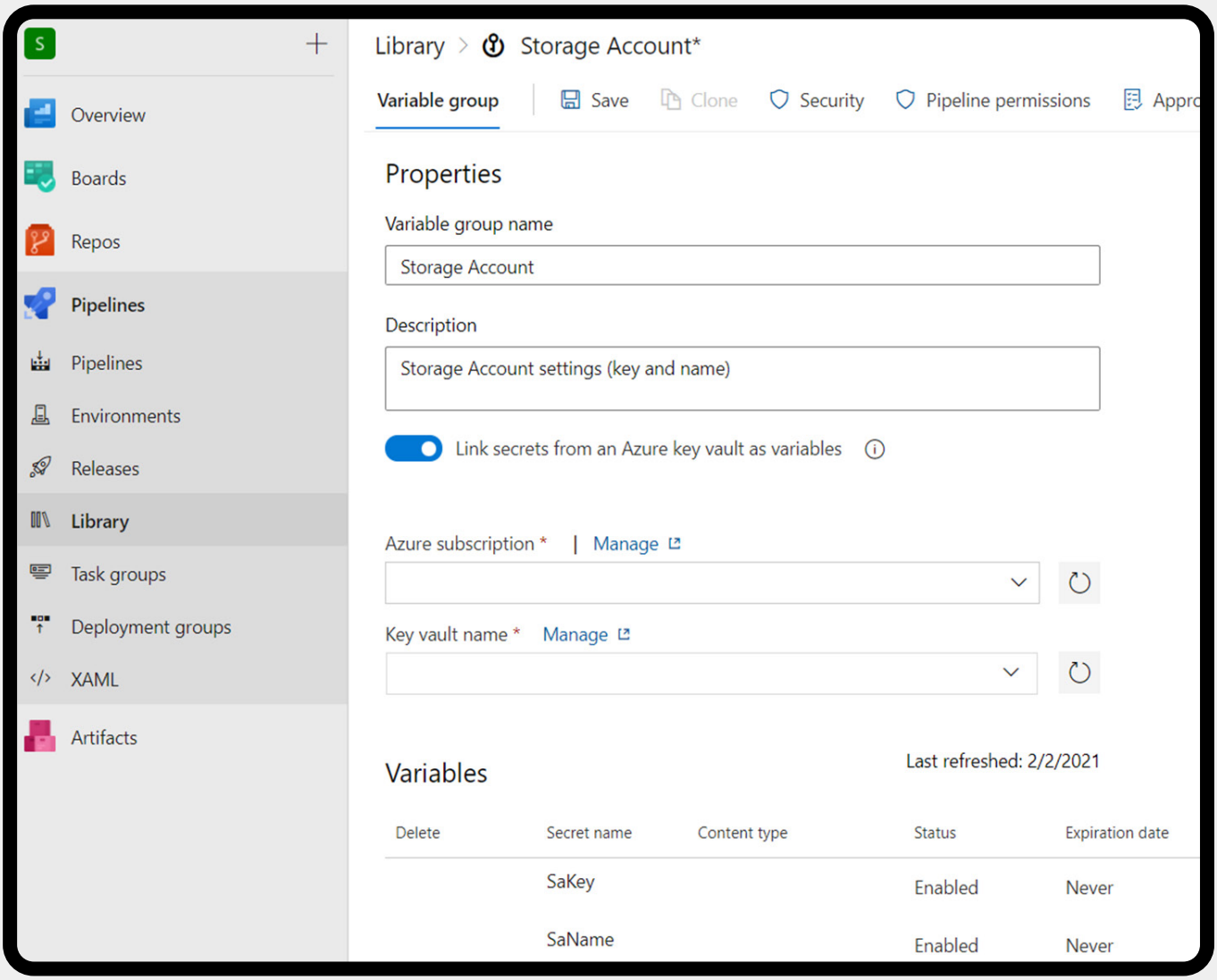


Figure 13 - Screenshot of Effective Azure Key Vault Variables

Automate scans for Infrastructure-as-Code (IaC) templates

As 'everything' becomes code, how we automate processes and checkpoints needs to evolve along with the relatively new ways of working. DevOps teams use a combination of tools and languages to get their job done, but how do they verify their code is running safely? To raise the maturity of your security posture and ensure compliance, it's necessary to automate scans for IaC environments. With IaC environments, it's even easier to scan for misconfigurations, compliance audits, and policies issues. If left unchecked, potential issues can lead to an easy entry point for a potential hacker. One such danger occurs when a VM is deployed to a cloud, without the right access controls, any malicious actor could connect to this instance.

Below is an example IaC automated scan using the Sogeti CloudBoost Library. This is helpful in displaying highlighted issues for remediation along with a security posture score for the entire environment.

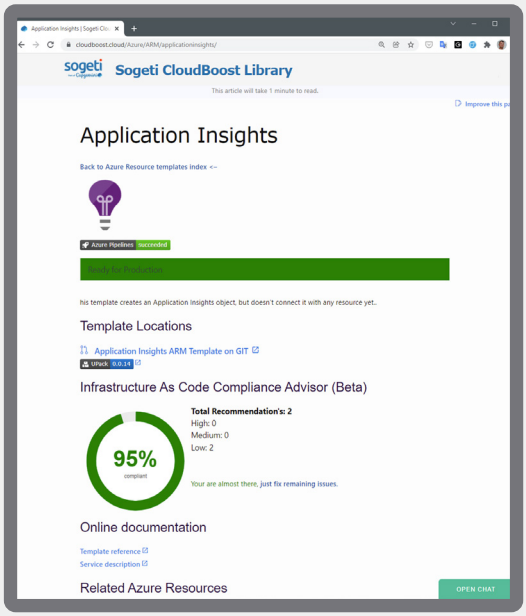


Figure 14 - Screenshot of Scan Results within Sogeti CloudBoost Library

How to

There are 3 key components of any IaC template scan: Access controls, static code analysis, and compliance checks. You'll also need to validate the cloud IaC templates before you provision with static code analysis. Furter, implementing compliance checks and access controls raises the security posture of your entire infrastructure.

It's important to establish a first line of defense with automated checks from trusted tools. There are many tools available, but our recommendation is to use Open Policy Agent (OPA) based tools. The language for writing the OPA policy, rego, is easy to adopt for untrained developers. Some companies create ready-to-use policies and share them online in open-source projects. One example, Kics.io, is loaded with the best practices for numerous languages. An example running against Terraform sample code is shown below:

```
Storage Account Not Forcing HTTPS, Severity: HIGH, Results: 1
Description: See that Storage Accounts forces the use of HTTPS
Platform: Terraform

        [1]: terraform-example1/storageaccount.tf:4

                003: ######
                004: resource "azurerm_storage_account" "storage" {
                005:   name                     = var.storage_account


Key Expiration Not Set, Severity: HIGH, Results: 1
Description: Make sure that for all keys the expiration date is set
Platform: Terraform

        [1]: terraform-example1/keyvault.tf:37

                036:
                037: resource "azurerm_key_vault_key" "wvd" {
                038:   name         = "wvdkey"
```

Figure 15 - Terraform IaC Example Code Scanning Sample

This scan found several issues including a missing HTTPS enforcement on a storage account, which is seen as a high severity case. Additionally, this scan flagged a few keys in a Key Vault without expiry.

Now that you've seen an example, make sure to adapt this practice to your own enterprise's needs. Adding your own best-practices or compliance requirements is easy, making the checks automated and safer than a manual check in peer review.

Equip every DevOps platform environment with audit trails

While already a default security practice to track changes made in production environments, it's time to extend those same logging processes to all Enterprise DevOps environments. Ensure you're tracking who gained access, what change occurred, and the date/time for any active system. This specifically includes DevOps platform that teams are using with CI/CD pipelines that flow into production. Audit trails for DevOps tools provide robust ways to remediate threats quicker, find and alert on suspicious activities to possible breaches or vulnerabilities, and to find potential data or privilege misuse. Audit trails are a backbone of secure DevOps environments.

Audit trails are a backbone of secure DevOps environments.

How to

It's crucial to ensure everything is logged for analyzing what happens in the Enterprise DevOps platform. This includes configuration changes of any DevOps tools to capture any edits or additions to access security. This helps guarantee no one has gained unrestricted access to areas of the DevOps platform. Secret edits and views should be logged to understand who gained access or has changed secrets that potentially can be used to access production environments. Release pipelines must be monitored to understand what moved to production and who triggered the execution.

In general, it's important that all Enterprise DevOps platforms have audit logs and audit events available for administrators to review actions preformed on any DevOps system. Incorporating audit logs into the daily workflow for teams and enterprises responsible for cyber security is more challenging. Audit logs from Enterprise DevOps tools are often plain text JSON formatted log files, these logs can be queried by administrators. Note that many events may be already tracked by default within the audit log feature. Within GitHub, have administrators set up audit trails for:

- The organization an action was performed in
- The user who performed the action
- Which repository an action was performed in
- The action that was performed
- Which country the action took place in
- The date and time the action occurred

One example is logging the addition of a member, through the (add_member action) GitHub Action. More advanced configurations stream audit logs to an external data management system like Azure Event Hubs. Logs can then be stored presentably, set up with automated notifications, and with a serverless Azure function.

An even more advanced implementation of auditing and comes from integration with Microsoft Sentinel. In this setup, GitHub data is collected and pushed to Microsoft Sentinel. From there, your team can create a baseline for threat modeling process security scenarios which intake anomaly detection alerts. Then Microsoft Sentinel takes the prescribed actions and sends notifications to the security team. To go deeper on this subject, check out the Microsoft Sentinel GitHub repository (Azure/Azure-Sentinel) example workbook to see how to make this integration possible.

To tighten collaboration between DevOps and Security teams, start by integrating logged DevOps events with security tools. Better collaboration and logging also leads to earlier threat detection and faster remediation of breaches and vulnerabilities. Another integration between the DevOps teams and cyber security is described in the next chapter where container security scan results are pushed from the release pipeline to Microsoft Defender Security Center, giving cyber security insights and the capability to act in what is deployed to production.

Pipelines function as the place to build, test, and deploy code from repositories to production. Generally, when pipelines are written as-code and stored in the same Git repository as the application code, teams are quicker to embrace new code review expectations. This team review process follows the branching structure and pull request review processes configured in the DevOps tool. **Team reviews are one of the most crucial security practices to include for pipeline security.** Many teams also dictate an additional security check to introduce an approval step in the deployment workflow.

Next to the integration pipelines and Microsoft Security Center, pipelines can also be used to audit the DevOps environment. For example, this GitHub Action below can perform an audit of member authorizations in the GitHub organization; Membership Audit Action for Enterprises and Organizations.

```yaml
on:
  schedule:
    # Once a week on Saturday 00:00
    - cron:  '0 0 * * 6'

jobs:

  audit_log:
    runs-on: ubuntu-latest
    name: Membership Audit Log

    - name: Membership Audit Log Action
      uses: svanboxel/org-audit-action@v1
      with:
        enterprise: 'goodcorp'
        token: ${{ secrets.TOKEN }}
        issue: true
```

Figure 16 - GitHub Actions - Membership Audit Action

This action provides a CSV file listing all members and their respective authorizations on each repository. You can perform an issue with the result and affect it to a responsible who can check it.

To tighten collaboration between DevOps and Security teams, start by integrating logged DevOps events with security tools. Better collaboration and logging also leads to earlier threat detection and faster remediation of breaches and vulnerabilities.

Automate approval workflows

For any approval workflow to push code into production, certain automatic or manual checks must confirm the security, business value, status, and quality of each request. These checks function as a gate between development and production. Without checks, hackers can render your environments offline with denial-of-service attacks or easily inject code into production environments without flagging or triggering an alert.

More automation in security reduces the risk of human error and provides efficiency gains, but some automated actions depend on approvals or non-IT human actions. Imagine you want to open a new feature and you need to advertise beforehand. Then, you need a process to trigger the installation in the production environment once it is advertised. To do this, try tools like ServiceNow to automate the approval request and process the response.

How to

When setting up automatic CI/CD pipeline checks, there are many programs and tactics to leverage. In the example below, we'll look at integrating a change management system like ServiceNow. While some deem integrated gates as a 'step back' in the DevOps world, this can help speed up approvals. Here is an example of Azure DevOps using a standard ServiceNow Change Management gate:

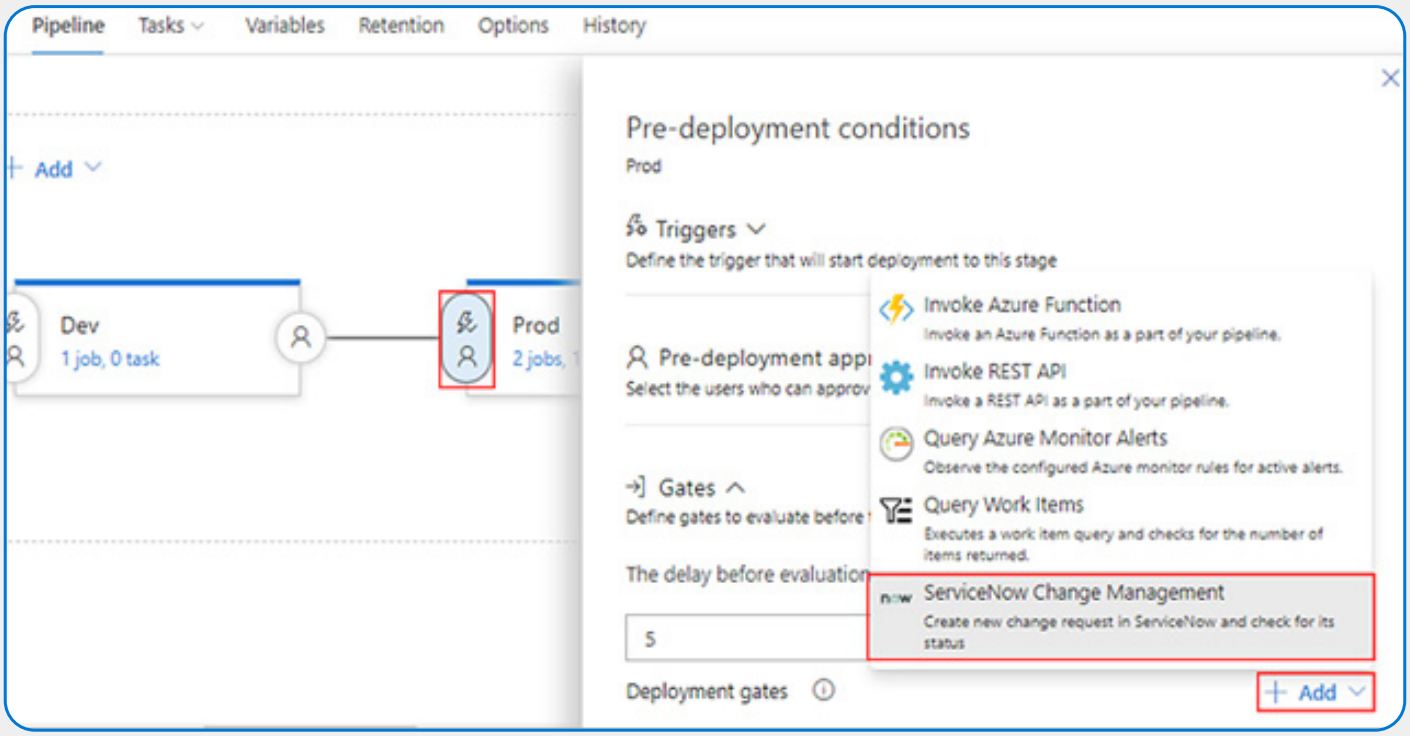


Figure 17 - Screenshot of Azure DevOps with a ServiceNow Change Management Gate

Often updates are tracked using change requests in ServiceNow. Automating the link between your DevOps tool and ServiceNow eliminates manual labor, updating issues, and reduces tickets. When using environments in either GitHub or Azure DevOps (ADO), teams can configure approvals for deployments to certain environments. A secure practice is to select environments per stage to contain credentials allowing for connecting and deployment to an Azure resource group. This extra step ensures that only the specified workflow is able to deploy to that environment—minimizing the attack surface when credentials are breached.

Once the ServiceNow or change management system is integrated, have your DevOps teams follow the practices below for secure deployments across environments:

Apply approval checks/gates on ADO environments instead of individual pipelines to manage all gates in one place.

In approval gates, approvers should grant approval for Infrastructure or Application deployment. Limit approvers to reviewers within a particular ADO group. This will ensure that the deployment gets reviewed before its deployed to a critical environment.

Set time out for approvals process to ensure they are completed within a specific, predetermined period, after which, the pipeline will not remain in queue.

Allow infrastructure and application deployment only from the main branch. Ensure that no deployment via any feature or development branch is allowed in Pre-Prod/Prod environments.

Set constraints so that the pipeline executor cannot approve his/her own pipeline run.

Secure the software supply chain

With every library you bring into your codebase, you expand the software supply chain and inherit dependencies from each open-source project or tool. These downstream dependencies are often the Achilles' heel of an application.

With every library you bring into your codebase, you expand the software supply chain and inherit dependencies from each open-source project or tool. These downstream dependencies are often the Achilles' heel of an application. **The first question you should ask yourself is, do I really need this dependency or library?** If the answer is no, it's best to remove the library and reduce the attack surface of your software supply chain. However, it's also best to exercise caution. In a worst-case scenario, single-line libraries cause havoc when revoked from a store or repository, breaking entire platforms. Another part of programming is the use of open-source components. Several studies have shown that as much as 90% of applications contain open-source components, let alone other libraries and frameworks. Keeping track of vulnerabilities and updates can be a daunting task.

Unfortunately, building everything yourself is not always the answer. Frameworks like .NET or Spring can be a solid ground to build upon, but it is extremely important to understand what version you are running and if there are vulnerabilities in your dependencies. Unknowingly running an older version or something with a known danger invites dangerous scenarios for any enterprise.

Vulnerabilities stemming from dependencies, libraries, and open-source projects included in the software supply chain are top of mind for governments too. President Biden's Cyber Security Executive Order details the danger in section 4 below:

THE WHITE HOUSE

v. Identifying relevant compliance frameworks, mapping those frameworks onto requirements in the FedRAMP authorization process, and allowing those frameworks to be used as a substitute for the relevant portion of the authorization process, as appropriate.

Sec. 4. Enhancing Software Supply Chain Security.

a. The security of software used by the Federal Government is vital to the Federal Government's ability to perform its critical functions. The development of commercial software often lacks transparency, sufficient focus on the ability of the software to resist attack, and adequate controls to prevent tampering by malicious actors. There is a pressing need to implement more rigorous and predictable mechanisms for ensuring that products function securely, and as intended. The security and integrity of "critical software" – software that performs functions critical to trust (such as affording or requiring elevated system privileges or direct access to networking and computing resources) – is a particular concern. Accordingly, the Federal Government must take action to rapidly improve the security and…

Figure 18 - Selected Passage from President Biden's EO on Cybersecurity

How to

To best track and update dependencies, there are several options for both GitHub and Azure DevOps. Dependabot helps GitHub teams with automated dependency updates and will warn you about updates regarding high or critical severity issues in your dependencies. GitHub shows these as alerts in a dashboard and emails the repository owner of these open issues:

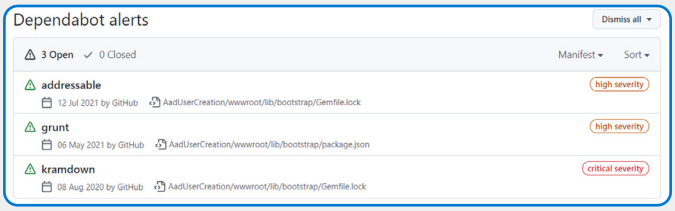


Figure 19 - Screenshot of Dependabot Alerts

Another tool to keep track of dependencies is WhiteSource Renovate. For GitHub there is an app available for use, and for other systems an open-source version is available. The great thing about this tool is that it will create pull requests automatically, enabling you to fix and update almost instantly. Note that Dependabot also creates PRs in the same process.

An example of such a pull request is shown below:

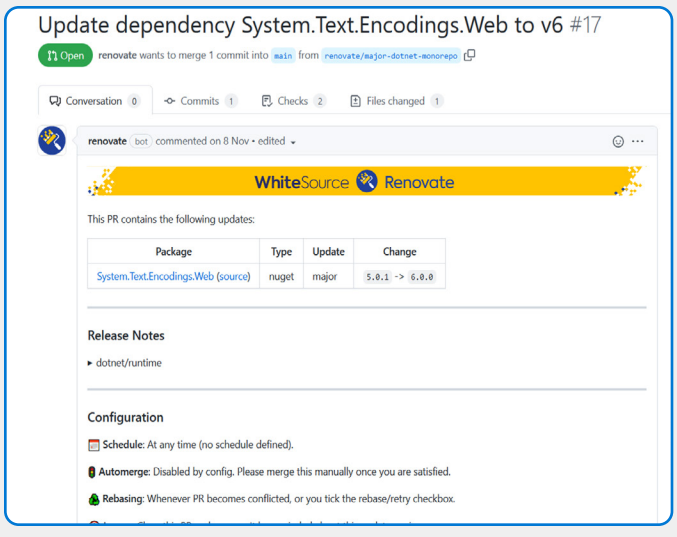


Figure 20 - Dependencies within WhiteSource Renovate

In this example, the update is a major one, which means you'll need to ensure application still works as expected. Assuming all the unit, integration, smoke, and acceptance tests are fine, you can merge this and be updated instantly. This style of proactive issue resolution leads to a shift in mindset. No longer can you simply ignore updates or major issues. Alerts will now force you to think and address issues.

Sharing packages and artifacts across your organization is something that should be set up properly. There are numerous ways teams can share code, from sharing actual binaries saved in Git or opening their source code. Often these practices grow organically, exposing your company to several risks. Using a package manager solves a lot of the plumbing and reduces complexity and risk. Examples of package managers are GitHub Packages and Azure Artifacts. These package managers will allow you to share the precompiled binaries in a uniform way (maven/nuget/etc.) and will abstract away a lot of the logic, simplifying the pipelines and setup of the consuming teams.

Another benefit of using your own package manager is that you control the deletion of files. For example, keeping every public source package you use—including packages from npmjs and nuget.org—safe in your feed where only you can delete it. Historically, the removal of a package from a public feed can cause many issues downstream.

Allow only verified DevOps tool integrations

As in developer environments, DevOps tools come with many extensions and integrations to make the DevOps team efficient and secure. A secure practice is to only allow verified integrations that require the least privilege possible to execute their work.

GitHub Actions and Azure DevOps offer extensive marketplaces with additional capabilities for CICD pipelines. When the team doesn't control or monitor an extension, it's a clear security risk. A common practice is that teams only use whitelisted and verified actions. GitHub organizations can be configured to follow this practice, as shown in the example below:

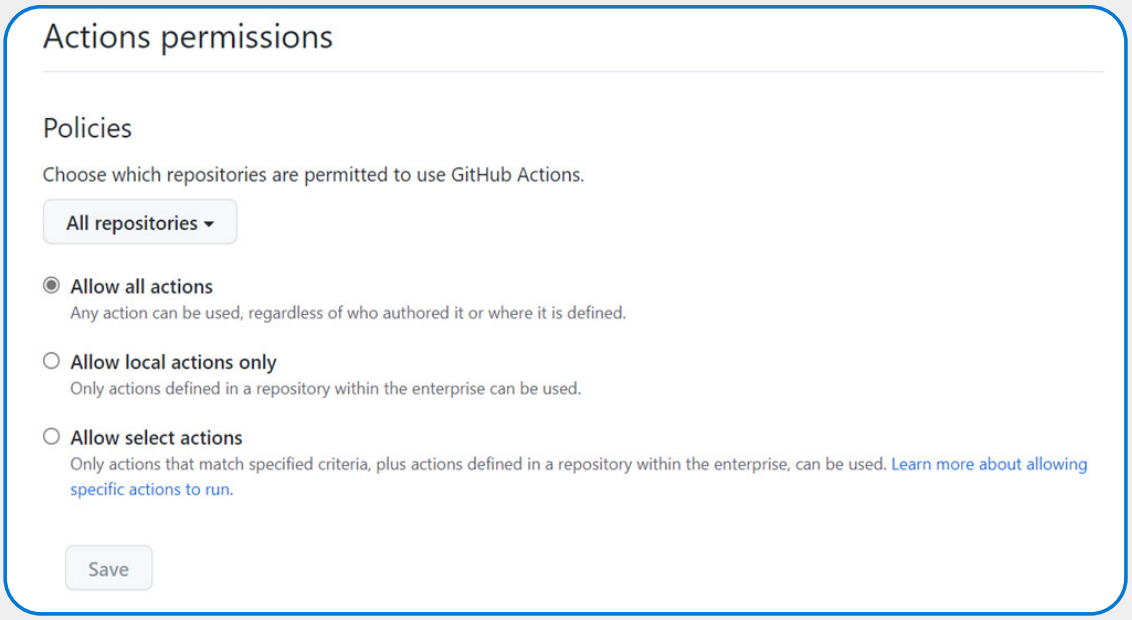


Figure 21 - Example of GitHub Actions Permissions

The same secure practice should be followed for integrations. Within GitHub, these integrations can be deployed inside a GitHub application. **Only enable integrations that are both validated and configured with least privilege access**. Check out the GitHub Application overview for Sonar Cloud below, set up with read and write access to code repositories and write access to pull requests.

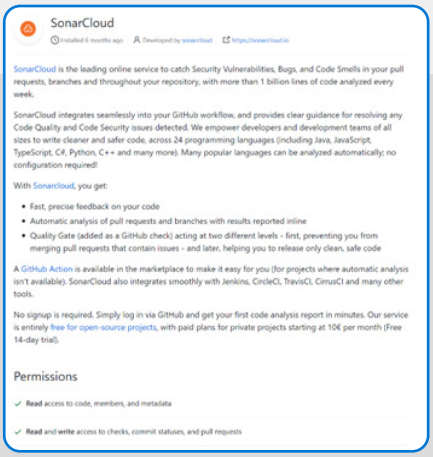


Figure 22 - GitHub Application Overview for Sonar Cloud

Secure the application environments

While test and development environments serve a different purpose from production environments, they too, can be open to the outside world and introduce risk if not secured. Chances are, the data you're storing, analyzing, or processing in non-production environments is just as sensitive as the data you push out to production. Application environments, when not secured, present a dizzying array of opportunities for hackers, including configuration drift between updates, open ports, access escalation, and vulnerability unawareness. So why skimp on security here just because it's not a production environment?

Companies often focus their privacy and data security teams on their production workloads because they contain the most sensitive and valuable data. Often attractive for intruders, the breach of this data imposes the most risk for financial loss, reputation loss, and regulatory fines. An insecure software development environment leads to breaches, but even insecure non-production environments can leave a company open to corporate espionage, sabotage, and theft of private consumer data.

Security matters equally in every environment. Adopting the same security practices and implementing security controls for every environment should be a default requirement for any DevOps team. What's more, the application environment is home to many of the strongest remediation tactics for any enterprise. Whenever a breach occurs, the best outcomes belong to teams that have enabled practices like environment standardization, automated deployments, extensive monitoring, software bill of materials (SBOM) creation, network segmentation, and repeatability of provisioning.

An insecure software development environment leads to breaches, but even insecure non-production environments can leave a company open to corporate espionage, sabotage, and theft of private consumer data.

In this chapter, we'll explore how to secure your application environments to defend and anticipate next-generation threats, including:

- Leverage Well Architected Framework (WAF) for landing zone provisioning
- Lock down environments with segmentation
- Provide visibility into delivery pipelines to security teams
- Remediate vulnerabilities quickly with a SBOM

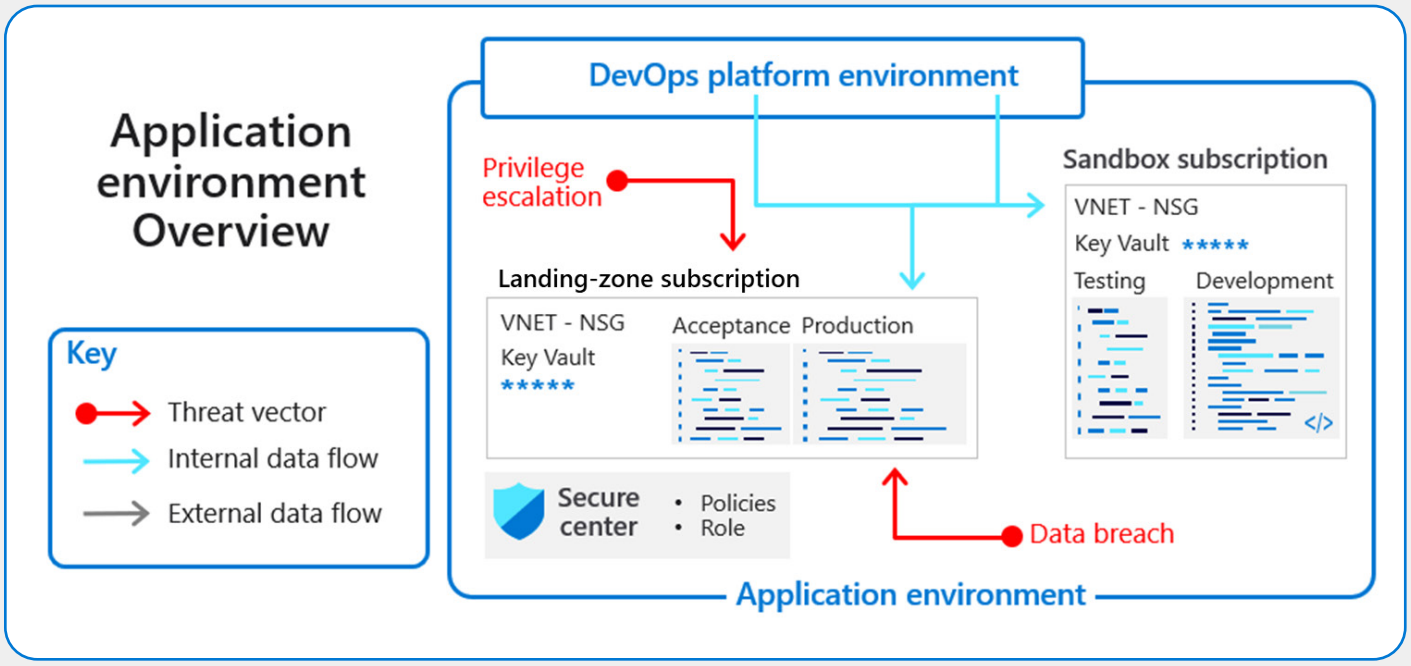


Figure 23 - Application Environment Overview

In the visualization of the application environment above, notice that the environment inherits inputs from the DevOps platform environment. The two largest areas of concern for this environment are privilege escalation hacks and data breaches. Both attack types target the landing zone due to the high volume of secrets, subscriptions, and company data located there. For this next chapter of the Ebook, let's explore some strategies on how to secure your application environment.

The real life (hack)

In securing application environments, many of the best practices covered in this chapter serve to create the quickest response to an incident that occurs in any environment. In many breaches, attackers don't need to gain access to production because the DevSecOps team uses production data to validate and find bugs in the test environment.

For example, let's examine a supply chain hack like the SolarWinds hack in 2020. In this instance, malicious code was inserted into the SolarWinds solution and deployed from there to its many customers. Worldwide, this affected thousands of organizations, not directly via production environments but through an automated update to customer management environments. While this is an example of a supply chain hack, the main lesson here is that **with the proper application environment protections in place, enterprises can limit the blast radius of a hack** and quickly trace, remediate, and secure themselves once again.

How can an organization prepare for when their software supply chain is compromised?

Limiting access and separating environments through security practices like least privilege and segmentation respectively are two immensely powerful techniques to limit damage from a breach. In this hack, malicious actors depended on high-level access to impose their will on threatened organizations. Confining access where possible directly hampers this potential danger. Hackers also required the ability to traverse across environments, which proper segmentation would have prevented.

Limiting access and separating environments through security practices like least privilege and segmentation respectively are two immensely powerful techniques to limit damage from a breach.

It is also important to provide continuous security monitoring across teams so that enterprise security teams receive up-to-date reports and visualizations of their environments. The longer a hacker spends unnoticed by security teams, the greater the cost of the breach. Fast remediation depends on an enterprise's capability to monitor and report threats.

A potent combination of monitoring, segmentation environments, and careful policy implementation through least-privilege access principles would have limited the damage facing exposed enterprises in this SolarWinds example.

Leverage Well Architected Framework (WAF) for landing zone provisioning

A great place to start with securing application environments is by building a secure landing zone. Public cloud providers build opinionated Well Architected Frameworks (WAF) to help guide this process. WAFs are a set of guiding principles with practices that can be used to run cloud workloads in an efficient and secure way. Most Well Architected Frameworks contain five architectural pillars: Cost Optimization, Operational Excellence, Performance Efficiency, Reliability, and Security. Where security is embedded in every pillar, it has a special focus on protecting systems and data from threats.

The graphic below shows the security pillar of the Microsoft Azure Well Architected Framework's practices, guidelines, and documentation:

Figure 24 - Microsoft Azure WAF's Security Practices, Guidelines, and Documentation

The result is an Azure configuration across multiple Azure subscriptions for scaling, securing, governance, networking, and identity management with built-in security practices. From there, application teams can 'land' their application resources on top of the landing zone to be sure enterprise and security concerns are met. There are multiple landing zone designs, from starter setups to more advanced enterprise configurations, review all the options provided by the Security Benchmark team to see which option best fits your organization. The example below is a detailed visualization of a WAF in practice:

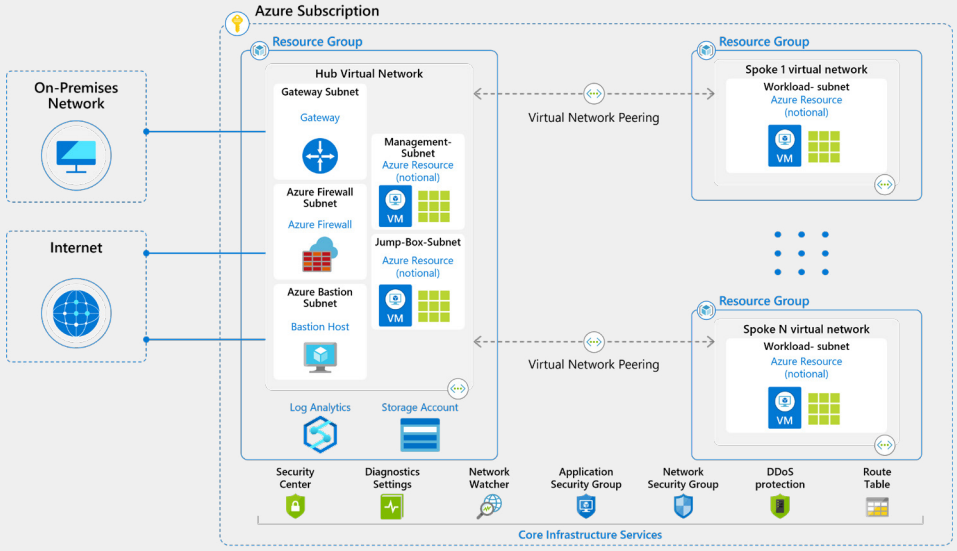


Figure 25 - Example of WAF in Practice

Cloud landing zone deployments need to be reliable, predictable, and gated with security at every deployment.

Cloud landing zone deployments need to be reliable, predictable, and gated with security at every deployment. This means relying on automation to eliminate human mistakes as often as possible. To protect your WAF designed landing zone against configuration drifts, misconfiguration, security risk, human mistakes and so on automation must be in place for all components. Automation both quickens routine deployments and ensures conformity to security protocols.

The system is designed to conform with industry standards and principles via the Well Architected Framework. Still manual actions and choices create holes in the configuration which are exploited by attackers faster. In 2019 for example, a hacker breached Capital One and gained access to production data via a misconfigured web application firewall.

As landing zones become more automated, you'll need to introduce even more standardization. In turn, more standardized environments lend themselves to implement business functionality more easily. This high level of automation also helps teams to be more productive. Now, it's easy to create landing zones for workloads, automated configuration of CI/CD tooling, deployment pipelines, and workload requirements. Automating the deployment helps to accelerate the deployment and creation of application workloads with the correct access and resources built in.

To be in control and compliant with the guiderails, one option is to leverage this automation to apply tollgates or quality gates prior to deployment. As most pipelines deployments start with the same basic steps, enterprises can then create standardized pipelines for code deployment with mandatory stages with tollgates. Pipelines are then triggered with code changes or updates but also at set time intervals. This practice applies to any code with landing zone provisioning process, even automation scripts.

How To – Automate the landing zone

For any cloud, it is important that the foundation is extensible and maintainable, because foundational changes to the landing zone will happen. Automation brings consistent quality, control, and the ability to update every day in a controlled manner. Any update to the landing zone must be supported by quality gates. If the landing zone is highly secure, anything built upon that foundation exists on top of secure services. This automation creates a desired state where teams can reapply and reapply, including the required security and quality with each update.

Let's examine a real-world example of these automated quality gates and automated landing zones which Sogeti teams built for an energy company. Below, the screenshot overviews that every firewall change results in the execution of thousands of tests to validate the new state of the network.

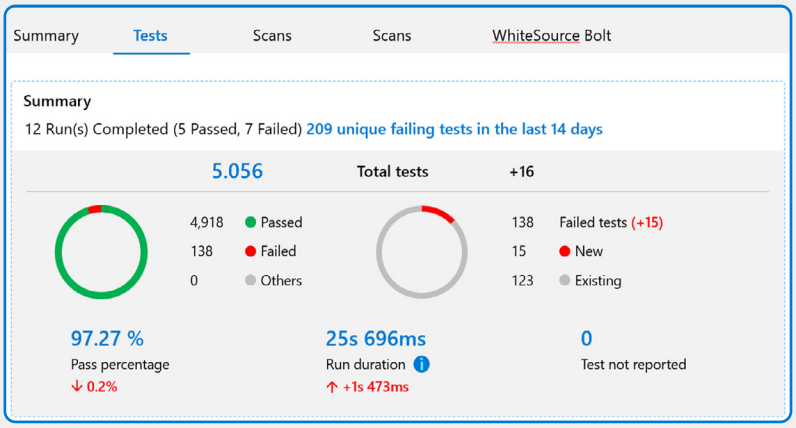


Figure 26 - Quality Gates Example

10 tips to automate WAF deployment and guidelines to automate our WAF deployment

1. Automating the base deployment helps to accelerate deployments and as well as the deployment of application workloads.
2. Ensure that when automated tasks run, fire and forget scripts are not allowed. Instead have them handled and deployed as code.
3. There is no a silver bullet, every deployment tool/script language has challenges.
4. Perform monitoring and drift detection on WAF, WAF code, pipelines, IAC tooling and automation scripts using your DevSecOps deployment process.
5. Automate pipelines from day 0 and build from a desired state.
6. Work in a modular model, small blocks enable you to create a complete package where features can be disabled or enabled. Not every workload needs the same features.
7. When using tooling, set it to always update to the latest version and downgrade when needed.
8. Backup, restore, and monitor tooling and configuration as part of your standard pipeline process. Do the same for each quality review, security review and even cost management.
9. Create and define a management structure, implement guidelines, and access controls at a high level. Ensure the deployment of the guiderails is also automated.
10. The WAF landing zone or foundation should enable and accelerate teams. If your framework is not secure, how do you expect the teams/workloads to be secure?

How to - Application team support with landing zones and InnerSource

Without default definitions and example of what a landing zone should look like, it might be difficult to ensure every DevOps team conforms to the security guidelines. **By defining default landing zones for your applications, it is easier for teams to do the right thing.** Maintaining a library with cloud 'building blocks' containing snippets of code or cloud resource templates compliant with the Well Architected Framework recommendations accelerates and makes the deployment secure without losing speed.

For example, Sogeti maintains libraries with cloud resource templates ready to be used by any team. Visualized in the diagram below, templates are automatically validated and maintained continuously. Used with pipelines as code to provision full landing zones (2) for security and compliance. DevOps teams (3) use the templates to provision cloud resources and conform them to company governance on top of the landing zone.

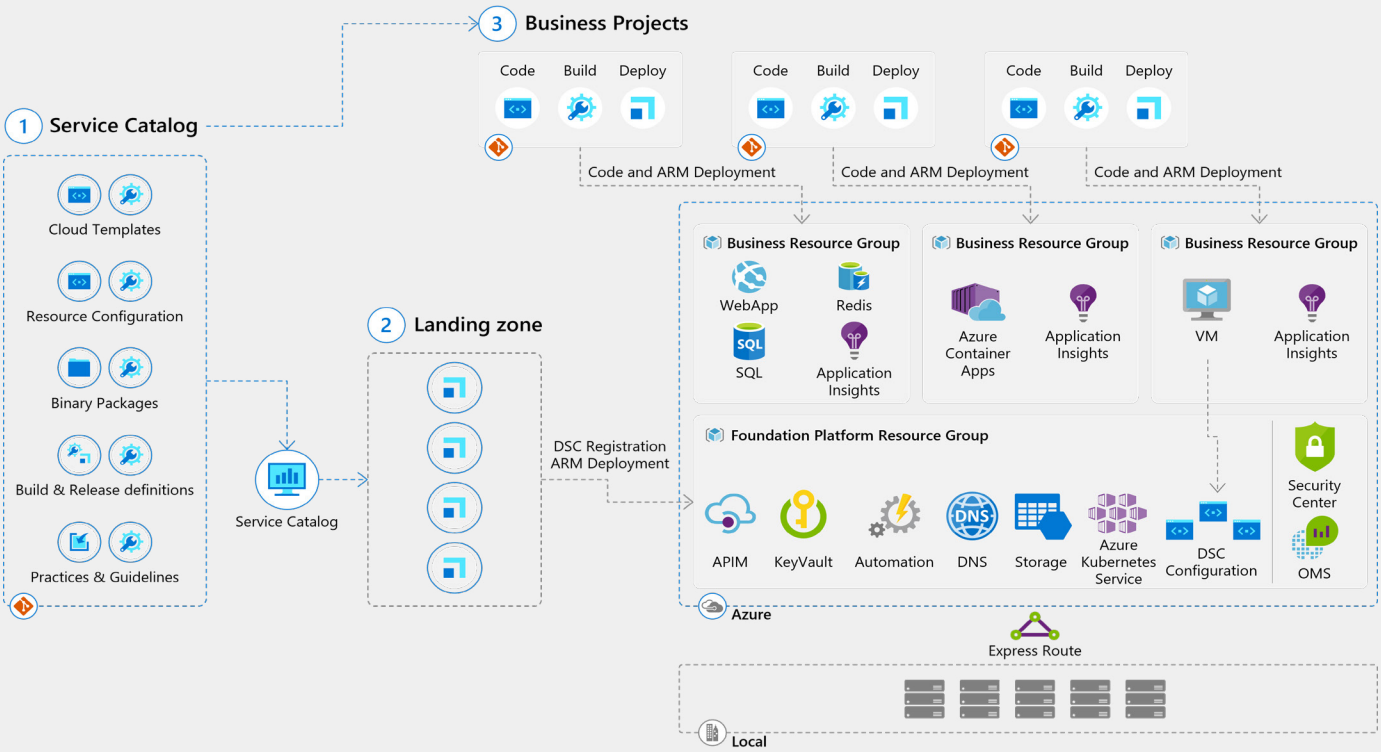


Figure 27 - Example Maintenance of Cloud Resources and Libraries

It's essential to run this store in an InnerSource model where everybody can contribute to keeping the code and templates evolving. Remember to pair this with automated processes so that any update is automatically validated and tested.

Lock down environments with segmentation

To reduce cyber security risks, segmentation creates boundaries that hackers cannot pass. The primary function of segmentation reduces the attack surface when a system is hacked. Segmentation is accomplished by carefully configuring networks and identities, where networks are physical boundaries with controlled access between them.

Segmentation limits the impact or blast radius when an application gets compromised. If an application consists of several components, compromise of one component doesn't imply compromise of the entire application. This can only be achieved if each component authenticates themselves with each other while communicating using a definite port and protocol. Further, components must be grouped in different security groups based on their sensitivity. It will ensure that these components will only get selective access and limit the impact of any compromise. For example, a database residing in different security group will not get compromised with the front-end application as they were in two different security groups. In our earlier example regarding Codecov, hackers would not have successfully exfiltrated secrets and compromised their system if their environments were properly segmented to prevent this type of supply chain attack.

Segmentation is accomplished by carefully configuring networks and identities, where networks are physical boundaries with controlled access between them.

As there are many levels you can apply segmentation to, we've set apart some of the scenarios for you to start with. There are segmentation capabilities at a management level all the way down to the actual resources.

How to - Setup management groups

The highest level of segmentation on an Azure subscription is a management group. Management groups are a logical hierarchical container which can contain other management groups and subscriptions. As your cloud estate grows, so too will the number of subscriptions within it. It's a good practice to enforce policies on Azure subscriptions using management groups from the beginning so the ensuring policies are set properly everywhere.

Below is a visualization of how management groups are configured across an enterprise, with some management groups housing several other groups and subscriptions.

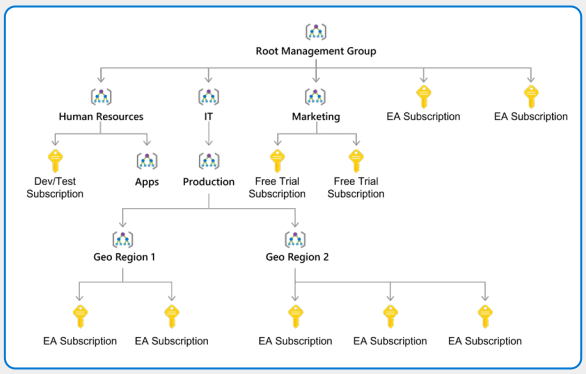


Figure 28 - Example of Management Group Configurations

Analogous to a fire alarm where users are instructed to "break the glass in case of emergency", a break glass mechanism provides access to the system in an emergency, bypassing the authorization. This can be useful for recovering from unforeseen circumstances.

A typical example is fixing the failure of the policy-based authorization system that results in large-scale incorrect denial of access. The engineer needs to circumvent the authorization system via the break glass mechanism to fix it.

How to – Isolate environments

Another deeper function of segmentation is isolation at the environment level. Follow the architecture principles of a hub and spoke model to bring workload isolation to cloud resources. The hub virtual network acts as a central point of connectivity to many spoke virtual networks. The hub can also be used as the connectivity point to on-premises networks. The spoke virtual networks peer with the hub and are used to isolate workloads. These spokes can be setup in isolation and without the ability to talk to each other. This way the workloads cannot talk to each other—reducing the blast-radius if something goes wrong.

Below is an example of this hub and spoke model within a typical Azure environment. In this diagram, the designer took careful consideration to separate the resources from other virtual networks and placed access controls where necessary.

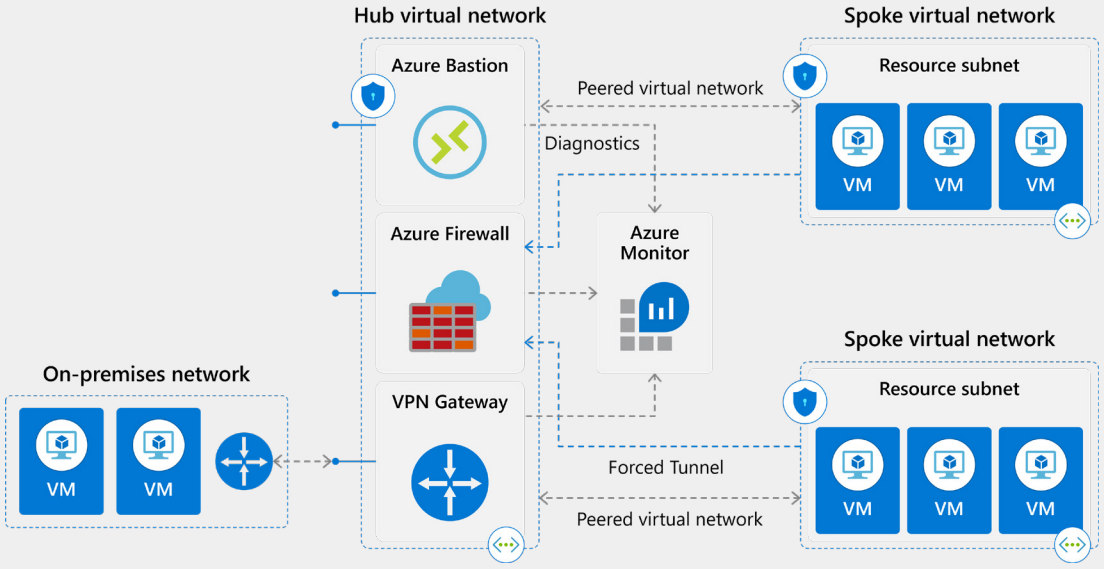


Figure 29 - Hub and Spoke Model Visualization

How to – Segment application workloads

Even within the spoke and hub model, having a resource group that separates your application from the other spokes is only the first step. Whichever spoke houses your application should also segment different parts of the application within it. If your application for example has four components as shown below, you'll want to make sure that only the services that need to communicate can.

In this situation there are two subnets, and only one of the workloads in subnet 1 should be able to connect to a specific instance in subnet 2. Using application security groups, segmentation is done on a network level, but you can group the actual network rules from a logical perspective, making it easier to implement and understand. For this level of segmentation, it helps to understand the connectivity and grouping of resources within your setup.

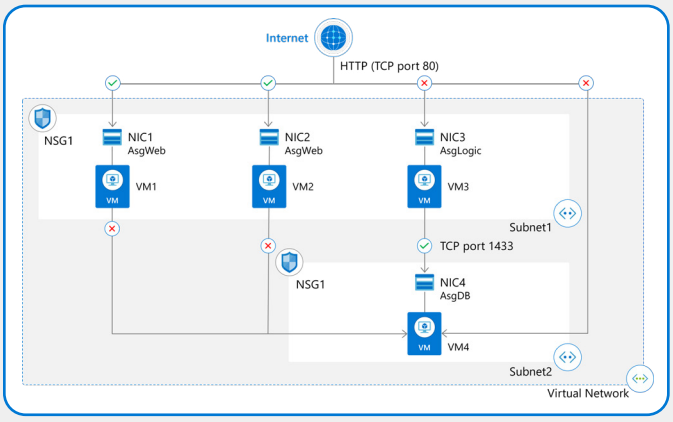


Figure 30 - Network Segmentation Example

Provide security teams with visibility into delivery pipelines

Security teams normally live in the production environment, securing the production data from being compromised with their specific tools like Microsoft Sentinel and Microsoft Security Center. When DevOps teams share security scans results, pipeline incidents and deployments with security teams, not only does the barrier between security and DevOps teams decrease, but security issues are also found earlier. Security teams can then use the functionality of their security tools to help find security issues before a system is deployed to production.

For example, pushing security scans of container images into Microsoft Security Center brings enhanced visibility and control. Security teams can then access a holistic, 360-degree view across CI/CD pipelines and runtime resources through CI/CD scan assessments in Azure Security Center. This helps create a greater shared insight into development practices, potentially vulnerable code, containers, and deployments.

How to

It is possible to configure security scanning into CICD pipelines with many different tools and publish the result to any dashboard. Below is an example of an integration between GitHub and Microsoft Security Center. Once enabled, the security scan results appear in security dashboards and help configure alerts for developers and environments to operate more securely. CI/CD vulnerability scanning gives much needed visibility into container images and the GitHub workflows that are pushing these images. Here is an example dashboard:

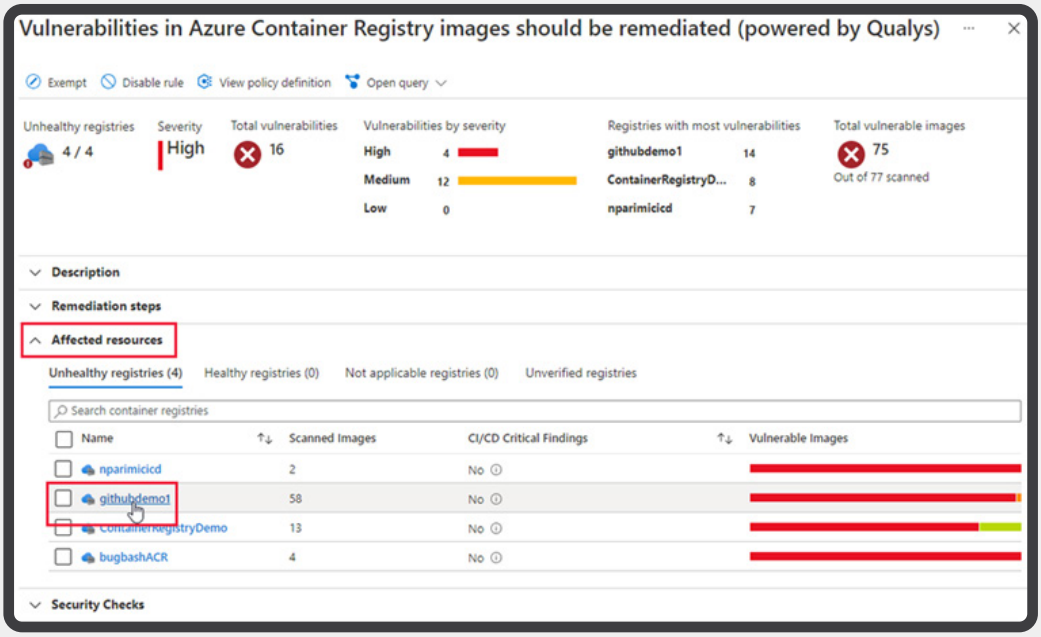


Figure 31 - Example of Dashboard Scan Results

Alignment and collaboration between teams naturally grows as their tools share data. Sharing security scan data between security and DevOps teams brings both worlds together. This collaboration is the hard part when integrating. Only when the enterprise security team receives insights to the possible vulnerabilities deployed into production can they act on it and help improve the DevOps team practices. This requires trust between teams and a shared goal—unlocking improved remediation time and strengthening cloud security posture.

The above example only overviews security scan results shared from container scanning. Security teams require more information, like which external packages are used along with which version, when and by whom a pipeline is executed, who changed it and what were the components installed, which configurations were changed. This is a similar practice to audit trails with Microsoft Sentinel covered in the last chapter.

Another integration between DevOps and security teams can be accomplished with Azure Security Benchmark. With Azure Security Benchmark SecOps teams can continuously validate if new and existing deployments conform with security and compliance policies. Company specific controls can be created and applied to all running workloads on all environments.

Azure Security Benchmark reports on, and continuously validates if Azure resources are configured to conform to security and compliance policies. Organizations can use Azure Security Benchmark to consistently and evaluate Azure deployments against these industry standards such as CIS Controls v8 and v7, NIST SP800-53 Rev4 and PCI-DSS v3.2.1.

Below is an example scan in Microsoft Security Center that displays the compliance findings across environments. Visualizations like this immediately indicate where problems occur so security teams can act swiftly. Powerful capabilities like this rely on an intentional setup of visibility across environments, teams, and tools.

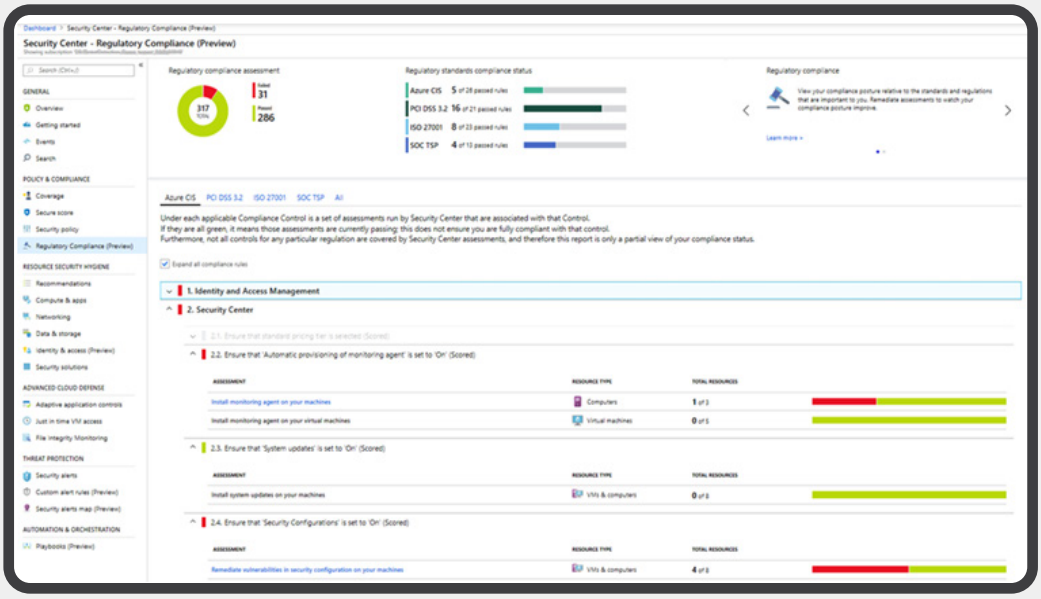


Figure 32 - Screenshot of Microsoft Security Center Scan Results

Remediate vulnerabilities quickly with a software bill of materials (SBOM)

Remediating vulnerabilities as soon as they occur is one of the easiest ways to fortify application environments. However, this often requires a full redeployment of production systems if a vulnerability is found.

Teams with fully automated release pipelines can redeploy the system almost instantly after the necessary fixes are made. But not all systems have an automated deployment, let alone a list of systems running in production. What's more, without that production system knowledge, it is nearly impossible for teams to keep track of downstream vulnerabilities as well. This lack of awareness brings security risks, even to systems which typically run stable in production. A software bill of materials (SBOM) helps ensure companies remain cognizant of all software and software dependencies running across environments.

One real-world example of this is the recent log4J vulnerability, which impacted hundreds of companies. For companies with SBOMs and automated release pipelines, this vulnerability was fixed within days after it was found. Teams with capabilities enabled like GitHub Dependabot received automatic notifications with pull requests to update the version and automatic deployments updated the environments.

In many other cases, organizations couldn't easily respond to vulnerability due to the unknowns within their system. Not knowing how many or which systems, and where they exist in production, presents a significant security risk for many organizations.

How to

To bring insights into production systems, DevOps teams leverage DevOps tools with automatic audit trails, along with workflow approvals and integrations with ITSM tools. Additional insights from production systems such as components, dependencies, and packages require additional security practices to be adopted. Once sufficient insights are gathered, DevOps teams should then extend automation with a generated SBOM and store it within easy access to the security team.

Once this is set up, creating a software bill of materials every time the system is deployed to production is a rather simple task. There are open-source and commercial tools available that can connect to the release automation to trigger the SBOM creation.

OWASP and SPDX are the main two standards that drive the way systems communicate for SBOMs. OWASP CycloneDX is a lightweight SBOM standard designed for use in application security contexts and supply chain component analysis. SPDX is another widely used SBOM standard, but it is instead ISO licensed and certified. In both standards, there are a wide variety of tools to couple them with. The screenshot below details some of the ways to integrate and generate SBOMs using tools and actions from the GitHub Actions marketplace.

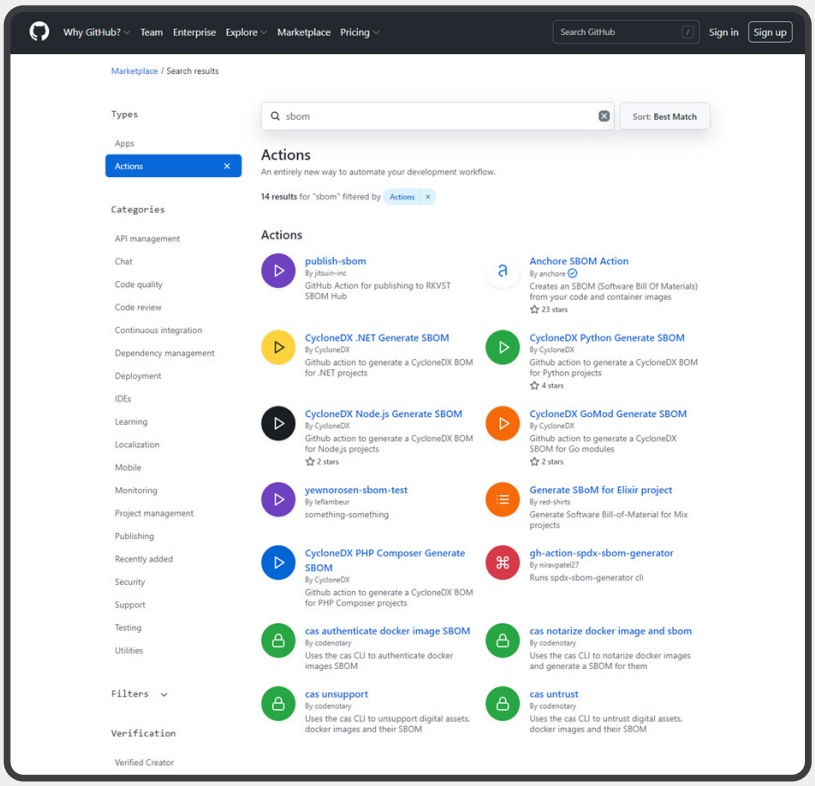


Figure 33 - Screenshot from GitHub Actions Marketplace

Once a SBOM is created, it's time to put it to use! A SBOM is effective only when set up to trigger alerts based on fragile dependencies, compromised packages, or when required updates are found. Generally, the trickier part comes when integrating these alerts and insights within the workflow of security teams. Missing any of these insights or suggested actions may result in triggered panic behaviors. For example, when a vulnerability appears that's spread over multiple environments like Log4J, a panic behavior will trigger ad-hoc and inefficient activities due to the lack insights—losing precious time.

While a fantastic tool for any enterprise, a SBOM with insights on packages, versions, licenses, vulnerabilities can be even more valuable. To decrease security risks even more, start tracking how projects are maintained, what best practices they follow, and which tools do they use. For example, it's possible to bring more cyber resilience to a system when relied upon packages are maintained by multiple people. Fragile dependencies occur when packages rely on only one maintainer.

Once you've located insecure sources, there's another step to ensure they're addressed. It's crucial to both execute a static security test and additionally perform a digital signage to confirm packages and build releases. Think of this rich security information as the recipe alongside the ingredients for each package. Most importantly, guarantee all of this information is readily visible for security teams to audit the safety and resilience of environments.

With all of this information successfully documented, how can you verify the safety and security of each SBOM? You can actually grade your SBOM against security scorecards, which initially started as a Google project in partnership with GitHub. These security scorecards check and report automatically the security practices a team follows against industry standard practices.

Scorecards are an automated tool that assesses a number of important heuristics ("checks") associated with software security and assigns each check a score of 0-10. You can use these scores to understand specific areas to improve in order to strengthen the security posture of your project. You can also assess the risks that dependencies introduce, and make informed decisions about accepting these risks, evaluating alternative solutions, or working with the maintainers to make improvements. Check out more on the OSSF Scorecard Repository

A great way to start is running scorecards against a Git Repo to give insights into how the repo compares to industry standards. The easiest way to use scorecards on GitHub projects you own is with the Scorecards GitHub Action. The Action runs on any repository change and issues alerts maintainers to view the results in the repository's Security tab. The scorecards GitHub Action is free for all public repositories. Private repositories are supported if they have GitHub Advanced Security. Private repositories without GitHub Advanced Security can run scorecards from the command line by following the standard installation instructions. With this Action you'll receive a "report card" like the screenshot below that details how secure your repository is compared with industry standards.

```
RESULTS
-------
Aggregate score: 7.9 / 10

Check scores:
```

SCORE	NAME	REASON	DOCUMENTATION/REMEDIATION
10 / 10	Binary-Artifacts	no binaries found in the repo	github.com/ossf/scorecard/blob/main/docs/checks.md#binary-artifacts
9 / 10	Branch-Protection	branch protection is not maximal on development and all release branches	github.com/ossf/scorecard/blob/main/docs/checks.md#branch-protection
?	CI-Tests	no pull request found	github.com/ossf/scorecard/blob/main/docs/checks.md#ci-tests
0 / 10	CII-Best-Practices	no badge found	github.com/ossf/scorecard/blob/main/docs/checks.md#cii-best-practices
10 / 10	Code-Review	branch protection for default branch is enabled	github.com/ossf/scorecard/blob/main/docs/checks.md#code-review
0 / 10	Contributors	0 different companies found -- score normalized to 0	github.com/ossf/scorecard/blob/main/docs/checks.md#contributors
0 / 10	Dependency-Update-Tool	no update tool detected	github.com/ossf/scorecard/blob/main/docs/checks.md#dependency-update-tool
0 / 10	Fuzzing	project is not fuzzed in OSS-Fuzz	github.com/ossf/scorecard/blob/main/docs/checks.md#fuzzing
1 / 10	Maintained	2 commit(s) found in the last 90 days -- score normalized to 1	github.com/ossf/scorecard/blob/main/docs/checks.md#maintained
?	Packaging	no published package detected	github.com/ossf/scorecard/blob/main/docs/checks.md#packaging
8 / 10	Pinned-Dependencies	unpinned dependencies detected -- score normalized to 8	github.com/ossf/scorecard/blob/main/docs/checks.md#pinned-dependencies
0 / 10	SAST	no SAST tool detected	github.com/ossf/scorecard/blob/main/docs/checks.md#sast
0 / 10	Security-Policy	security policy file not detected	github.com/ossf/scorecard/blob/main/docs/checks.md#security-policy
?	Signed-Releases	no releases found	github.com/ossf/scorecard/blob/main/docs/checks.md#signed-releases
10 / 10	Token-Permissions	tokens are read-only in GitHub workflows	github.com/ossf/scorecard/blob/main/docs/checks.md#token-permissions
10 / 10	Vulnerabilities	no vulnerabilities detected	github.com/ossf/scorecard/blob/main/docs/checks.md#vulnerabilities

Figure 34 - Screenshot of GitHub Actions Analysis

Secure Enterprise DevOps in practice

As technology continuously evolves, a developer's work never remains the same, and failing to keep up can mean exposing the business to cyber threats or failing to meet deadlines. Some of these threats like social engineering, inability to drive consistent policies, difficulties raising issues, and siloed teams have consistently remained a thorn in the side of every enterprise.

To mitigate these risks, it is important to drive awareness on Secure Enterprise DevOps best practices in our teams. Start by providing continuous training, a self-service portal, blending teams, and embedding security in every aspect of team practices and stages in the application lifecycle.

Keep in mind that not every team works in the same way. In large enterprises with teams and systems built up over decades, teams create and adapt their own working styles and behaviors. Let's explore some of these working styles in a visualization of DevOps operating models in the diagram below.

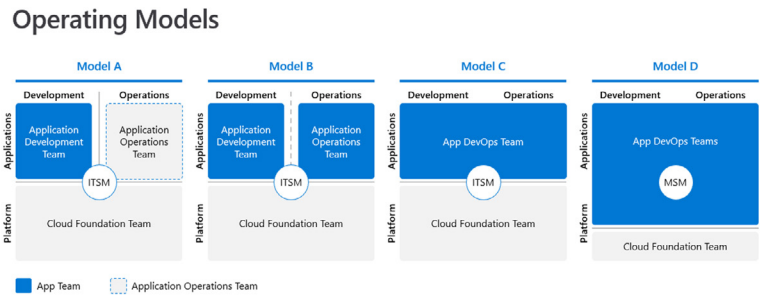


Figure 35 - Operating Models Overview

Operating Model A outlines a more traditional way of working where teams are beholden to multiple dependencies due to legacy components or specific restrictions for the system they've built and maintained over time. Contrast this with the DevOps team in Model D. With almost no dependencies, this team is much more independent. Count on better performance from these independent teams, especially when switching to a product-centric setup.

For security officers, different operating models can mean tending to requirements of disparate teams. Moreover, different operating states create issues for automation practices across teams as well. Ultimately, teams that operate without rigid boundaries share knowledge. If this practice is not followed throughout the company, knowledge gaps arise.

In this chapter, we'll review some ways for your teams to align different operating models within the organization to better support security teams and the automation capabilities DevOps and DevSecOps teams need to function, such as:

- Teach DevSecOps best practices with InnerSource and security champions
- Find ways to leverage self-service support
- Switch to product-centric DevOps teams

The real life (hack)

Social engineering, the use of deception to manipulate individuals into divulging confidential or personal information, tends to target the most high-value targets. And members of a DevOps teams are no exception. Although they're among the most tech-savvy users within an enterprise, hackers continue to develop new phishing mails and malicious websites tactics to circumvent developer security measures. In 2021, a hacker compromised a developer's account with a phishing mail from a Word macro attachment that ultimately led to the theft of $55M from a crypto company.

Phishing can compromise everyone in an organization, including the DevOps team.

Phishing can compromise everyone in an organization, including the DevOps team. What's worse, when a developer is breached, the impact can be catastrophic. How do we confront the constant reality of social engineering? The answer is to focus on expanding awareness and continuous training for all employees to remain vigilant on cyber security. Most importantly, team members need a complete understanding of not only the developer environment, but also the configuration and capabilities of the DevOps tools and the application environments. Moreover, it's crucial to create a reporting mechanism like a self-service portal, so that employees get support as quickly as possible. With awareness, training, and tools, your DevOps teams will be much more prepared to outpace social engineering.

Teach DevSecOps best practices with InnerSource and security champions

Often, DevOps teams stray away from the required security, quality, and compliance policies expected of them on a daily basis. It's likely not intentional. Too frequently, these issues stem from a lack of awareness and understanding of the expectations asked of them. **Making teams aware of these policies is an important first step in closing potential security issues.**

To truly adopt and maintain Secure Enterprise DevOps, usage and setup policies for development processes must remain front and center.

To truly adopt and maintain Secure Enterprise DevOps, usage and setup policies for development processes must remain front and center. These policies need to be supported by written best practices, automated guiderails, self-service opportunities, and clear documentation. Make sure to leverage your combined team's knowledge and draft a security strategy with the joint DevSecOps team. This strategy and set of policies should then undergo a thorough review by security, legal, data protection, and software development specialists. Drawing from a combined set of expertise makes a policy understandable and objective. Below are some examples of these policies, with even more examples of shared practices to be found here.

Example security policies	Example legal policies	Example data protection policies
Branch protection policies	Usage of copyleft licenses in private/internal repositories	Storing personal data in GIT
Usage of verified GitHub Actions	Usage of incompatible OSS licenses	Keeping client assets/ data secure
Environment and Secret management		

Generally, teams focus on the system they develop and maintain, but they also build best practices on how to be compliant with their security and DevOps policies. Issues arise when trying to share this knowledge across the enterprises and with other DevOps teams in the organization. It is not that teams don't want to share the knowledge; they are simply focused on their system and workload. So how do you ensure that employees across teams become aware and retain knowledge of these different practices?

One option is to expand the use of InnerSource practices to immediately increase security ownership and shared responsibility across teams. InnerSourcing is where teams compile best practices across domains and share them in a central repository to be viewed and leveraged in a repeatable way. It is important to note that InnerSource (and open-source) methods also require policies setting and awareness across teams. Anytime an individual makes a secure contribution, they must be guided by clear and simple policies.

One option is to expand the use of InnerSource practices to immediately increase security ownership and shared responsibility across teams.

Successful enterprises leverage the champion model to improve security policy awareness, where one team member is the security representative in each respective team for the organization. These leaders are responsible for disseminating security knowledge to their teams. They also develop security practices within their teams that are communicated back to the entire organization. When both functions of the security champion model are implemented, information is shared from the ground up and from a central source.

Security champions and InnerSourcing work best when used in tandem. Together they offer the quickest path to policy and best practice adoption, providing teams both a leader and a repository for examples.

Find ways to leverage self-service support

With so many ways to secure DevOps environments, building one that truly works for your organization may feel daunting. However, sometimes the optimal path is the easiest. When building secure DevOps environments, feel empowered to leverage all the best practices that GitHub provides including ways to secure your GIT repository, the code it holds, and access policies for contribution. Once you have created the environments leverage templates and APIs to develop a self-service API portal. The foundation of the templates and support from the self-service portal when used together provide a secure backbone for any enterprise.

The foundation of the templates and support from the self-service portal when used together provide a secure backbone for any enterprise.

Use templates, automation, and insights to help your DevOps team configure environments the right way from the start. One example configuration captures how to create scratch branch protection rules, enable Dependabot, assign maintainers, and configure GitHub Advanced Security. By providing the DevOps team with insights, governance is then automatically provisioned for each project. With insights, DevOps teams also better understand the behavior of their contributors. For example, when someone works with GitHub for the first time, it's possible to automate a message for their first contribution with guidance on how to secure secrets with GitHub Actions. Leveraging automation to provide specific training and guidance helps define the learning expectations for each role on the team.

Leveraging automation to provide specific training and guidance helps define the learning expectations for each role on the team.

Enterprises with multiple teams often face the same legal and security issues when securing their DevOps environments. Legal and security experts cannot control each individual GIT repository and environment but need a way to ensure the security and legality of all the GIT repositories in their organization. The best way to free up DevOps teams while providing control to legal/security experts is through self-service support. This support configures GitHub conform enterprise requirements, so a controlled and secured DevOps platform is available for teams. With all stakeholders the support team defines the policies with which DevOps teams need to be compliant. Automation of GitHub using GitHub Apps, Code Scanning, Dependabot Alerts, and WebHooks can validate the defined policies and support the teams to apply to the policies. Automated report findings of non-compliance will cycle back to the DevOps teams or assigned security team.

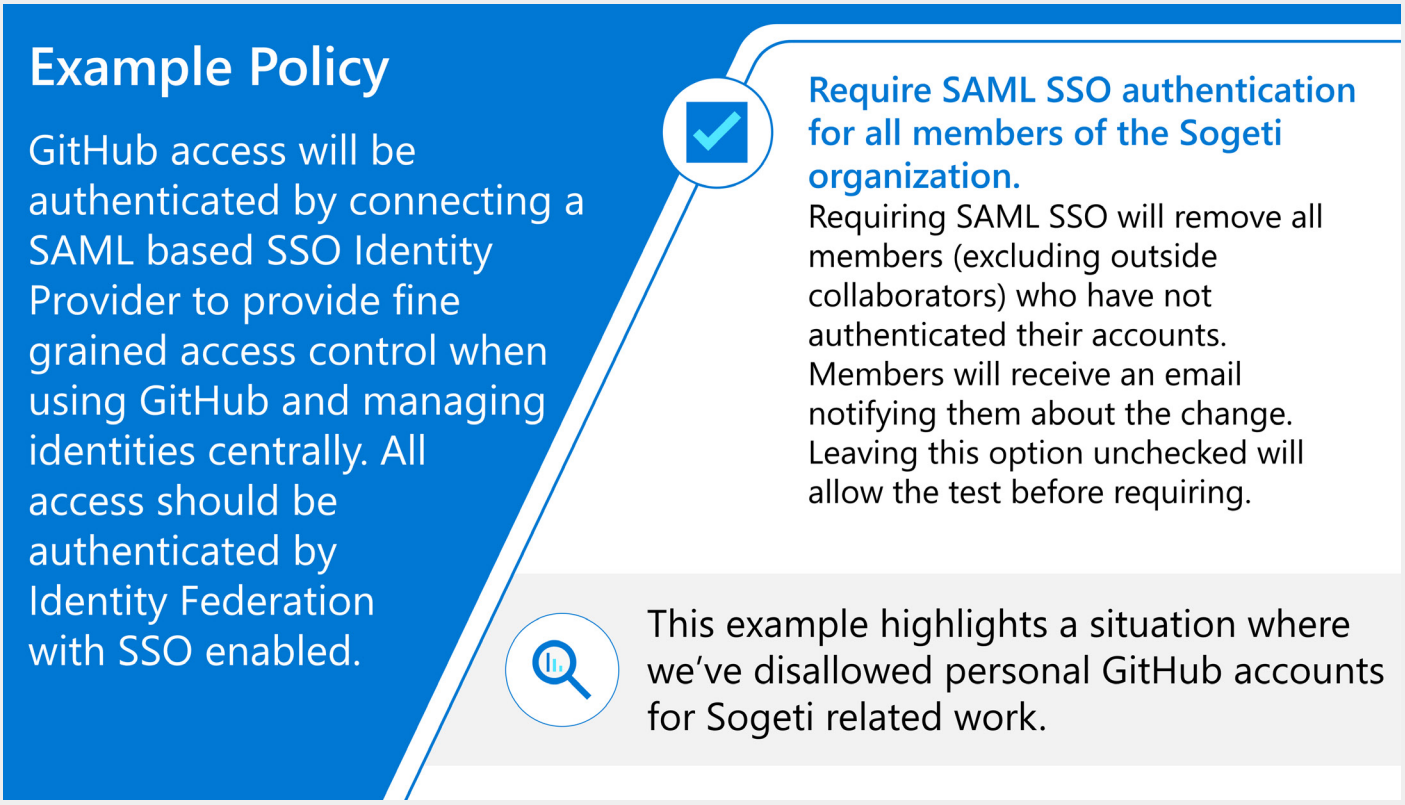


Figure 36 - SSO Policy Example

With automation of policy validation, the DevOps team environment can be secured at scale. Whenever policy infringements are detected, a direct response is then initiated. By knowing the responsible project lead and Security Officer we can escalate when infringements aren't resolved.

A self-service portal for DevOps teams is the forcing function to activate these automatic validations, security checks, environment provisioning, and governance reviews. Each repository created via this self-service portal can easily be controlled on all aspects of project governance including security. For example, it is relatively easy to set up a self-service portal that validates the security and analysis features enabled on your environments. If these services have not been enabled, the self-service APIs send notifications on how to enable these features to the team. At the same time, this API can be configured to send a training resource that details the policy, the value of the setting, and more documentation to the team.

Self-service portals also support the use of GitHub project administration systems and can immediately escalate high priority incidents to the project lead even when he isn't a member of the GitHub project. Or, when a high severity issue occurs on multiple projects directly, the portal can automate a report out to the cybersecurity department with full coverage of all affected projects.

Switch to product-centric DevOps teams

For greater agility, organizations are moving from centralized "project-centric" models to de-centralized "product-focused" delivery models, where DevOps product teams take full ownership of the end-to-end cycle of a product or service. This product-centric approach provides the decentralization and adaptive structure that teams and enterprises need to succeed in their digital transformation journey. Instead of sourcing from a technology siloed approach, teams are built solely to help product-focused ideas and goals.

Time and tools are the key ingredients for product-centric team success. Time will help these DevOps teams mature their collaboration, fine-tune practices, and automate where needed. But they'll also need the tools to define, build, test, and release in an efficient and secure way powered by automation, innovation, and best practices.

Time and tools are the key ingredients for product-centric team success.

DevOps product teams can start by establishing a training program so that each new team member learns various platforms, standards, tools, best practices and earns recommended certifications before they become part of a team. Once a new team is constructed, build up their work experience via DevOps dojo's, InnerSource, and hackathons. Direct the product team support unit to provide training, team building capability, and staffing requests via demand forecasting.

As a result, business leaders can request a cross-functional, autonomous team that's used to working together to solve a business problem instead of requesting a specific number of analytics specialists, infrastructure professionals, developers, security practitioners. The maturity of their collaboration and simplicity of team structure allows for a reduction in project startup time, tightened security practices, and a higher likelihood for project success.

The maturity of their collaboration and simplicity of team structure allows for a reduction in project startup time, tightened security practices, and a higher likelihood for project success.

Closing thoughts

It's clear, one of the pitfalls of enterprise security is focusing solely on building and maintaining secure applications. While a critical undertaking, applications are not the hackers' only playing field and many attacks target developer environments. Enterprises need to defend developer boxes, release pipelines, and production data within test environments. These hacks are increasingly well-known, including the Codecov and SolarWinds breaches, where hackers gained access to organizations' intellectual property via either pipeline tools or maintenance tools.

While a critical undertaking, applications are not the hackers' only playing field and many attacks target developer environments.

For regulated industries, security now requires an upheaval of compliance and security practices. The new mandate for compliance is for organizations to know which software is up to date, who initiated it, and for this historical data to be recorded in audit trails. Second, from a security perspective, it is vital to know what code is deployed to production and which software can interact with production data. Further, enterprise DevOps cybersecurity methodologies must incorporate a secure and governed approach to dictate who edits release pipelines, provisions environments, sets configurations, and makes code changes. This should govern not only their own environments and code, but also the dependencies and integration across teams.

The pressures of this new threat landscape are not only being felt at corporations. President Biden's Executive Order on cybersecurity and The European Union Agency for Cybersecurity (ENISA) 2021 publication, 'ENISA Threat Landscape for Supply Chain Attacks', have made it clear that securing enterprise DevOps environments is top of mind for governments as well. In Biden's executive order, he calls for the formation of security standards to improve supply chain security. Whereas the 'ENISA Threat Landscape for Supply Chain Attacks' publication details new cybersecurity methodologies to educate organizations on supply chain attacks.

Stay one step ahead of shift-left hackers using the practices in this Ebook. While there is no best-fit option that'll work in every scenario, the practices outlined here can be easily adapted to work for any enterprise. Remember, that without awareness and training, your teams won't truly accept and adopt the changes you've implemented. The journey, to securing enterprise DevOps environments is a continuous one. Take every opportunity to stay on top of new trainings, best practices, and the latest standards to best guard against risks.

Stay one step ahead of shift-left hackers using the practices in this Ebook. While there is no best-fit option that'll work in every scenario, the practices outlined here can be easily adapted to work for any enterprise.

How Microsoft & Sogeti can help

Secure DevOps or DevSecOps for enterprises requires secure DevOps environments powered by cross-team collaboration, a focus on developer velocity, and cutting-edge tooling. Microsoft offers learning resources, products, and services to position all DevSecOps teams for innovation, regardless of language, framework, or cloud.

Azure expert managed service provider, Sogeti, continuously improves business-focused digital delivery for DevSecOps teams utilizing cloud and DevOps Centers around the globe. Sogeti uses its DevSecOps Adoption Framework and CloudBoost library to drive continuous improvement and InnerSource adoption within DevSecOps teams—leveraging the full platform capabilities of Azure for governance, security, and compliance as a cloud-native foundation.

Inspire other technical leaders to integrate security into their DevOps practices by sharing this whitepaper on social media or email.

Resources

Microsoft DevSecOps solution: Integrate security into every aspect of the software delivery lifecycle. Learn how Microsoft offers a complete solution to enable DevSecOps, or secure DevOps, for apps on the cloud (and anywhere) with Azure and GitHub.

GitHub: Secure at every step. GitHub helps enterprises stay ahead of security issues, leverage the security community's expertise, and use open source securely.

Microsoft Azure: Strengthen your security posture with Azure. Gain unique security advantages derived from global security intelligence, sophisticated customer-facing controls, and a secure hardened infrastructure.

Documentation: Learn how Azure security helps to protect your applications and data, support your compliance efforts, and provide cost-effective security for organizations of all sizes.

Talk to our sales team: Talk to our specialists to see how Microsoft can help your DevSecOps team.

Sogeti works with Microsoft and GitHub to ensure its clients create value from their DevOps platform, approaches, and cloud capabilities.

Sogeti DevSecOps Adoption Framework: Release faster with a DevSecOps Enterprise Reference Architecture, product descriptions, and DevSecOps Blueprints.

Sogeti CloudBoost Library: Fully automated Cloud Landing zones, a Cloud foundation for Enterprise DevOps teams to land applications on a secure base.

Sogeti OneNative Services: Support for the cloud native journey by providing Sogeti Product and Feature teams which create business value with experienced DevOps teams on a secure and compliant foundation.

Authors

Samit Jhaveri is the Director of Product Marketing with Microsoft Azure focused on cloud application development, DevSecOps, and DevOps with GitHub. He serves as the business leader working across product management, sales leadership & finance with responsibility for defining and executing the e2e go-to-market strategy including pricing & offers and execution plans such as campaigns and field & partner motions for growing the business. Prior to his current role, Samit led an engineering team at Microsoft's Server and Tool Division and was responsible for shipping several B2B solutions for different vertical industries. Samit earned his MBA from the University of Washington and a Masters in Management Information Systems from the University of Arizona.

Clemens Reijnen is Sogeti's Global CTO Cloud Services and DevOps leader. He has been awarded the Microsoft Most Professional Award for 10 years in a row and is a SogetiLabs Technical Fellow. He co-authored the books '6 tips to integrate security into your DevOps practices' and 'Enterprise DevOps Report 2020-2021' with Microsoft and writes frequently on cloud and DevOps on Sogeti.com. As a global DevOps leader, he works closely with Sogeti's large enterprise customers to ensure their cloud adoption and Enterprise DevOps transformation programs create value for the business.

Sogeti co-authors:
Peter Rombouts, Matt Braafhart, Rahul Sharma, Sandra Parlant, Martijn Mulder, Andre Andersen, & Yogesh Patil.

© 2022 Microsoft Corporation. All rights reserved. This document is provided "as is." Information and views expressed in this document, including URL and other internet website references, may change without notice. You bear the risk of using it. This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.