A DoD Enterprise DevSecOps Reference Design

DevSecOps with DoD Cloud IaC, Microsoft Azure, and GitHub

Author: DOD Office of the CIO and DISA Hosting and Compute Center (HaCC) with Microsoft Federal

26 September 2022, v0.01, b385

Document Approvals

Approved by:

LAMB.GEORGE.WILLIE.IV.1506724321 Digitally signed by LAMB.GEORGE.WILLIE.IV.1506724321 Date: 2022.10.03 16:12:45 -04'00'

George W. Lamb

DoD CIO, Cloud and Software Modernization Directorate

WILLIAMS.JONATHAN.MICHAEL.1273151423 Digitally signed by WILLIAMS.JONATHAN.MICHAEL.1273151423 Date: 2022.10.04 15:45:54 -04'00'

Jonathan M. Williams

DISA, Hosting and Compute Center, Infrastructure as Code PM

Contents

List of Tables

List of Figures

1 Introduction

Modern information systems and weapons platforms are driven by software. As such, the DoD is modernizing software practices to deliver resilient software at the speed of relevance. DoD Enterprise DevSecOps Reference Designs provide guidance on how specific collections of technologies come together to form a secure and effective DevSecOps platform for building software.

This DoD Enterprise DevSecOps Reference Design uses Software-as-a-Service (SaaS) and Infrastructure as Code (IaC) to quickly establish a DevSecOps environment. This environment includes development tools to create and manage the software production process as well as separate cloud infrastructure for integration, test, production, and operations management. The reference design makes significant use of Cloud Service Provider (CSP) native services to secure and operationalize the environment into a DevSecOps platform. It also provides patterns for using CSP native services to build and deliver production applications.

This approach recognizes the advances CSPs have made in providing programmable infrastructure, on-demand tools, and a catalog of robust services that run in the cloud. It is possible to build complex applications and Application Programming Interfaces (APIs) for defense missions by configuring and deploying CSP services in the CSP marketplace. This design does not dictate the products or services to be acquired, but provides an implementation pattern leveraging these cloud resources. Note that CSP services are proprietary, and programs must be aware of the potential for vendor lock-in as they select and deploy services.

The DoD, in partnership with Microsoft, developed this reference design using Azure and GitHub technologies. Alternative technologies for the key aspects of this design are available through other CSP vendors. Working directly with Microsoft was appropriate to leverage their commercial best practices for using the Azure Government cloud. The DISA Hosting and Compute Center (HaCC)[1] developed the key DoD Cloud Infrastructure as Code (DoD Cloud IaC) used for this design along with templates to secure the Azure Cloud Services. This reference design leverages provisional authorized CSP services, which allow for the inheritance of controls when pursuing Authority to Operate (ATO). Systems using this reference design must still be assessed for a full ATO, but the provisional approval simplifies the process.

1.1 Development Pattern and Design Elements

Using CSP native tooling is most appropriate for programs that run on the cloud with commodity components. Within the cloud ecosystem, the CSP commercial clouds have the most complete offerings, while those tailored for Federal and DoD specific workloads at Impact Levels (IL)4-5+ are more limited in both tools and services. The development pattern used throughout this design takes advantage of the richness

[1]https://www.hacc.mil/Products/DoD-IaC

of commercial clouds for development then moving to more specific clouds for testing and operations. This approach is referred to as code low, deploy high.

Figure 1 illustrates the development pattern using Microsoft GitHub tools and services at IL2 to build software and pushing it up to IL4/5 for testing and operations. While this pattern is common across DoD software factories, it is specifically called out in this design reference due to the focus on using CSP native services.

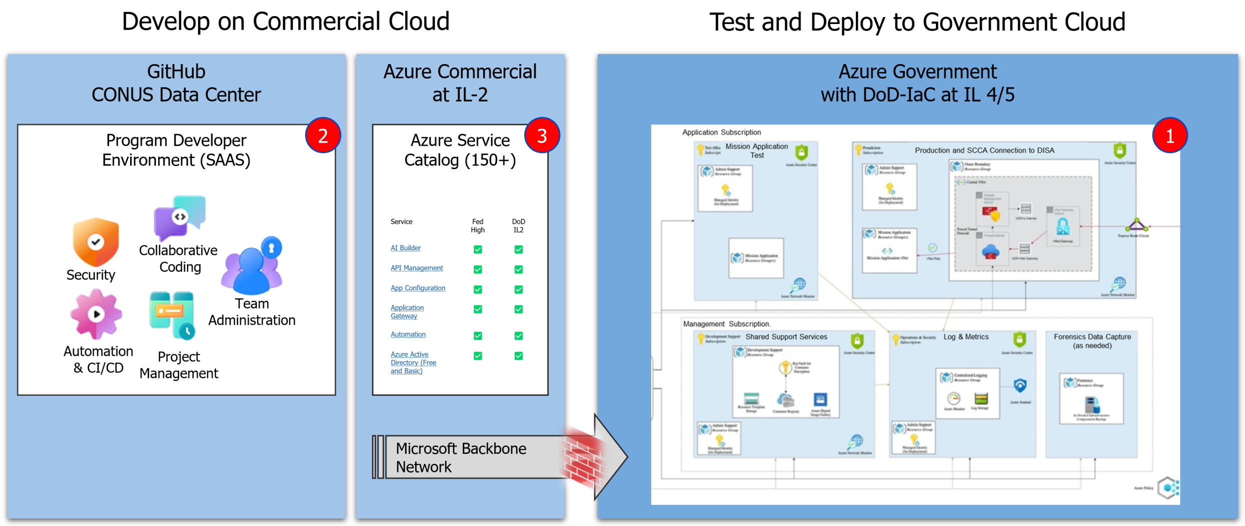


Figure 1: Azure Reference Design development pattern and design elements

The development pattern presents the basic flow of software through the cloud environment via a DevSec-Ops pipeline. There are three design elements necessary for implementing a complete software factory. These are described briefly, then expanded throughout the document.

1. **Use Infrastructure as Code to create the cloud infrastructure for the DevSecOps environment.** This element is labeled as #1 in *Figure 1*. It includes the subscriptions and virtual networks inside the dark blue box of the IL4/5 cloud. All the cloud infrastructure is generated programmatically including the integration, test, and production environments, as well as the security and centralized logging environment and cloud native access point (CNAP). This design reference uses the DoD Cloud IaC tools which are available on the Azure marketplace and include a DISA provisional ATO. The DISA HaCC office maintains the Azure DoD Cloud IaC baseline and regularly pushes updates.

   While the Azure DoD Cloud IaC is used for this reference there are other options for implementing the design. Microsoft also provides a baseline for the Azure DoD cloud, called **Mission Landing Zone (MLZ)**[2]. MLZ is being assessed for provisional ATO and Microsoft is developing a Cloud Access Point (CAP) as an option to the DISA CNAP to connect the Azure Cloud to the DoDIN.

[2] http://aka.ms/MissionLZ

2. **Leverage CSP managed development suite provided as a service.** This element labeled #2 in *Figure 1* simplifies the provisioning of DevSecOps tools and presents a familiar suite for new team members. CSP managed development services just require a license and account to start building software. There are many different development suites available across CSPs. This design reference uses Microsoft GitHub Enterprise suite. Guidance for configuring GitHub to create organizations, teams, repos, CI/CD pipelines, and limiting scope for the code low, deploy high development pattern is provided. While this guidance is GitHub specific, it illustrates configuration aspects for establishing a DoD DevSecOps software factory.

3. **Develop applications using CSP native services and policies to secure the services.** CSP services are powerful building blocks used to build internet applications. They are easy to use and can be assembled into production applications quickly. The service catalog for Azure services is notionally represented by #3 in *Figure 1*. CSP services are used throughout the DoD Cloud IaC baseline, notably the Azure logging and monitoring services. The DoD Cloud IaC baseline also provides wrappers to secure services for application use.

   Every CSP has their own service portfolio, and not all services are supported across clouds at different impact levels. The list of CSP's, CSP services, and IaC secured services continues to grow. It is the responsibility of the Program Management Office (PMO) or development organization to verify that services used for development are indeed available in the target production cloud environment. The final section of this document includes examples that use common services in typical DoD applications.

This design does not cover containers or Cloud Native Computing Foundation (CNCF) Kubernetes and service mesh for managing containers. By using CSP native services the runtime complexity is managed by the CSP. Throughout the examples in this reference design, Microsoft Azure as the CSP, controls the provisioning and implementation details, while developers focus on configuring and customizing services to build program specific services and applications. If necessary, Azure provides a fully managed Kubernetes service[3], that can be used to build more complex systems. Other CSP's provide similar support for native CSP services, Kubernetes, and service mesh.

The deployment pattern and design elements together form the basis for a fully functioning software factory. Software pipelines will be created by the development teams using the GitHub tools and secured by the proper service configurations automated by the DoD Cloud IaC baseline. This approach to using CSP services covers a large portion of DoD applications that can be quickly built and securely deployed into production.

[3]https://aka.ms/AzureKubernetesService

1.2 Scope

This reference design is intended for programs that will primarily leverage CSP native capabilities as part of a DevSecOps software factory. The DoD Cloud IaC created environment fully resides in the cloud. Transitioning to other production runtime environments is possible but outside the scope of this reference design. CSPs are developing edge capabilities that include managed services and extend the cloud to new operational environments. As mentioned above, containerized applications are covered by other DoD DevSecOps reference designs.

The deployment pattern presents development at IL2 and production to IL4/5. It is possible to use this reference design to push products to higher impact levels (IL6+) via cross-domain solution, but there are additional required controls which are not covered by this design.

The DoD Cloud IaC baseline has a DISA provisional ATO for building a fully functional cloud software factory environment. They also include configurations for many CSP services used to deploy applications. Programs will need to work with their security officer to transition from provisional ATO to full ATO.

1.3 Target Audience

The target audiences for this document include:

- DoD PMOs and project teams who use DevSecOps to develop, secure, deploy, and operate mission applications, services, and APIs.
- DoD organization DevSecOps teams who manage (instantiate and maintain) DevSecOps software factories and associated pipelines.
- DoD Enterprise DevSecOps capability providers who leverage CSP native functionality.
- Authorizing Officials (AOs) who need to understand the controls and tools to monitor cloud applications and services, especially applications developed with cloud native services.

1.4 Reference Documents

This reference design aligns with these documents:

- DoD Digital Modernization Strategy, 2019.
- DoD Software Modernization Strategy, 2022.
- Cloud Native Access Point Reference Design, 2021
- DISA Cloud Computing Security Requirements Guide.
- Presidential Executive Order on Strengthening the Cybersecurity of Federal Networks and Critical Infrastructure (Executive Order (EO) 1380).
- National Institute of Standards and Technology (NIST) Cybersecurity Framework.

1.4.1 DoD DevSecOps Document Relationships

This reference design is to be included with the DoD DevSecOps document set as shown in *Figure 2*. It is compatible with the DevSecOps Tools and Activities Playbook version 2.1.

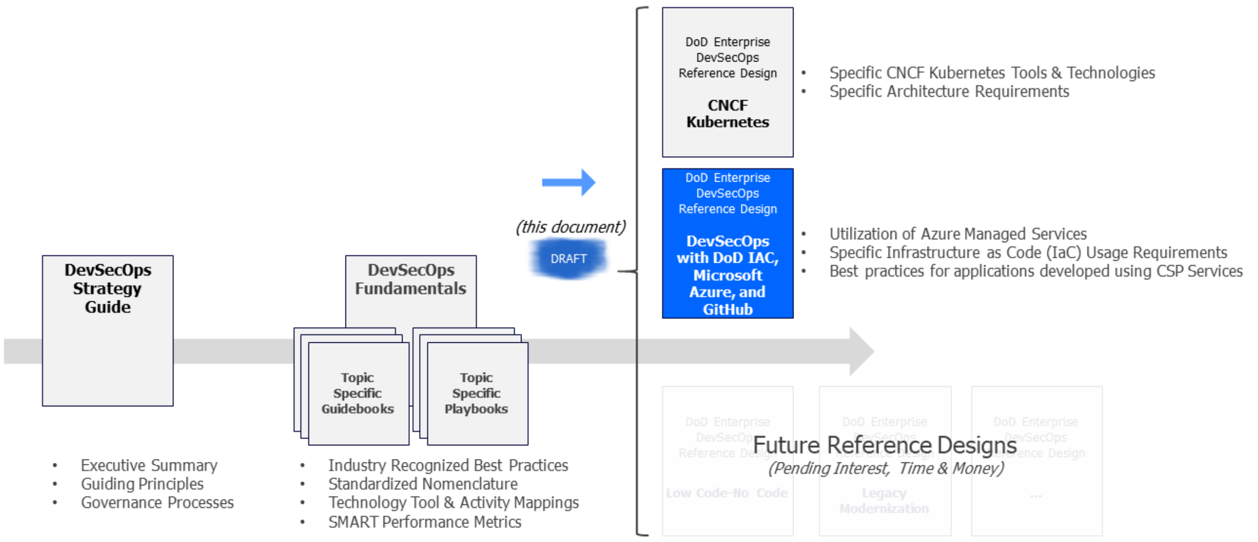

DoD Enterprise DevSecOps Reference Design
CNCF Kubernetes
Specific CNCF Kubernetes Tools & Technologies
Specific Architecture Requirements
(this document)
DRAFT
DoD Enterprise DevSecOps Reference Design
DevSecOps with DoD IAC, Microsoft Azure, and GitHub
Utilization of Azure Managed Services
Specific Infrastructure as Code (IaC) Usage Requirements
Best practices for applications developed using CSP Services
DevSecOps Strategy Guide
DevSecOps Fundamentals
Topic Specific Guidebooks
Topic Specific Playbooks
Executive Summary
Guiding Principles
Governance Processes
Industry Recognized Best Practices
Standardized Nomenclature
Technology Tool & Activity Mappings
SMART Performance Metrics
Future Reference Designs
(Pending Interest, Time & Money)
Low Code-No Code
Legacy Modernization
...

Figure 2: DevSecOps Document Relationships

1.5 Document Overview

This document is organized as follows:

- Section 1: describes the background, design elements and scope of this document.
- Section 2: provides a mapping of this design reference to the DoD DevSecOps software factory components.
- Section 3: describes the prerequisites for using the DoD Cloud IaC baseline, which includes the Azure accounts and roles necessary to get started.
- Section 4: covers the DoD Cloud IaC baseline which automatically builds the software factory and production enclaves along with the operations, security, and centralized logging environment and DoDIN cloud access point.
- Section 5: covers the GitHub build and development tools used as a service from the Azure GovCloud. These tools also orchestrate the DevSecOps pipeline across the DoD Cloud IaC environments.

- Section 6: covers the logging and monitoring environment used by the operations team to continuously monitor the DevSecOps software factory and production environments.
- Section 7: presents a mission application that was developed using the components of this reference design.
- The appendices provide a mapping from this reference design to the fundamental tools and activities necessary for specification as a DoD DevSecOps software factory.

2 Software Factory Components and Interconnects

The DevSecOps Fundamentals document describes a digital platform as a group of resources and capabilities that form a base upon which other capabilities or services are built and operated within the same technical framework. Use of a digital platform is encouraged to accelerate development, delivery, and cybersecurity accreditation. Each DoD DevSecOps reference design focuses on the digital platform as part of a complete software factory. *Figure 3* shows a notional software factory which includes applications, a digital platform with essential configurations for DevSecOps, and the underlying infrastructure. The digital platform configurations are known as Reference Design Interconnects. Well-defined interconnects enable tailoring of the digital platform, while ensuring that core capabilities of the software factory remain intact.

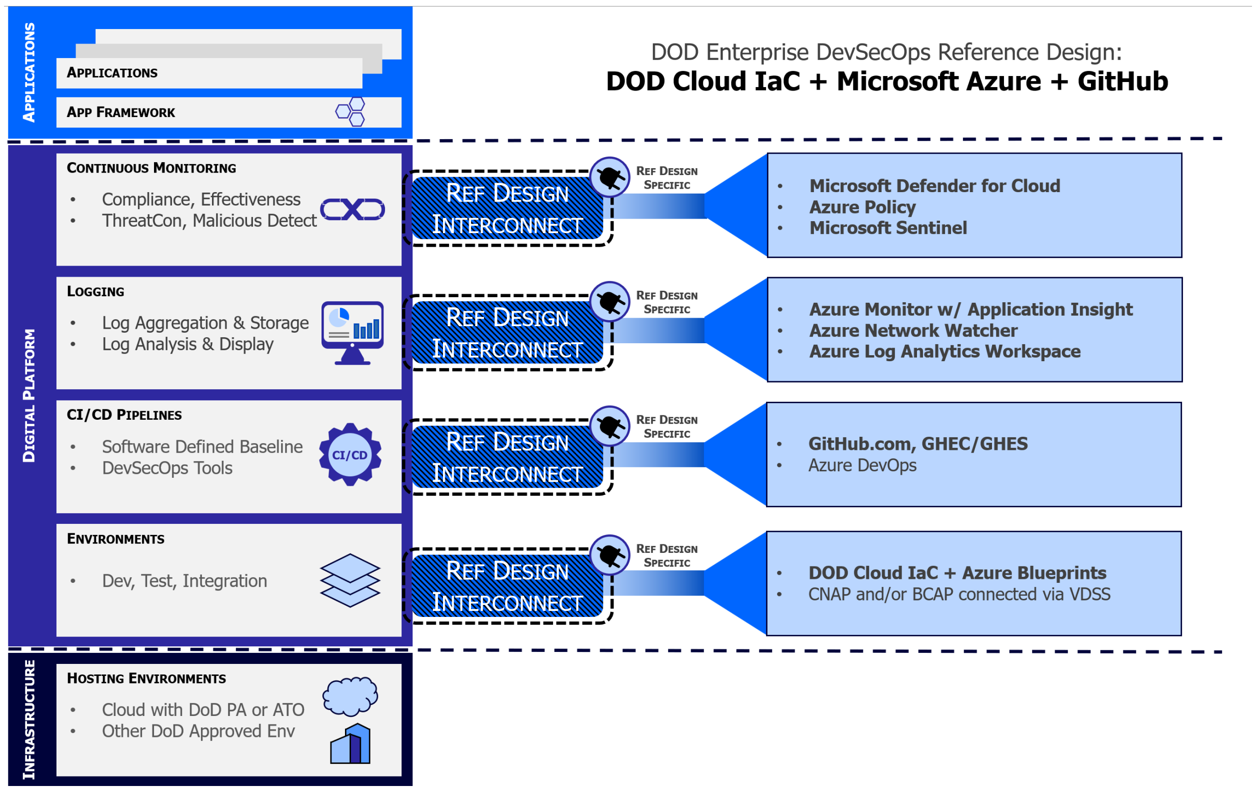


Figure 3: DoD Software Factory Interconnects

This design reference identifies interconnects as appropriate for the development patterns and design elements using CSP native services in a software factory. The cited capabilities are those provided by Microsoft Azure and DISA. Tailoring to leverage comparable technology from other CSP vendors may be appropriate subject to cyber certification.

2.1 Digital Platform Reference Design Interconnects

The reference design interconnects for a DevSecOps digital platform are presented in four areas: environment, CI/CD Pipelines, Logging, and Monitoring.

2.1.1 Environment

Azure Commercial includes a complete Software as a Service (SaaS) implementation of GitHub. GitHub is a key component used for developing and managing the software factory pipelines. Since it is provided as a service, there is no configuration necessary, just acquiring the appropriate accounts and using the tools. GitHub runners are used to pull code from the repository and insert into the integration, test, and production environments that are established by the DoD Cloud IaC.

The DoD Cloud IaC templates can be used to create a complete environment or a subset depending on program requirements. The complete environment includes the operations, security, and centralized logging enclave for monitoring across the various software factory enclaves. The templates can also deploy a CNAP to connect the Azure Government cloud to the DoDIN. By using the DoD Cloud IaC templates a significant portion of the security controls and policies are automatically configured to establish the basic DoD cybersecurity protections for the enclaves. Microsoft is also developing a set of Azure blueprints that can be used in place of or to augment the DoD Cloud IaC templates.

2.1.2 CI/CD Pipelines

DevSecOps pipelines are created and managed using GitHub. Chapter 5 includes guidance to create pipelines that are secured using zero-trust principles such as least privilege and application segmentation. Specific pipelines are created based on the software being delivered. Chapter 7 includes reference pipelines for typical applications such as web apps, data access, and API creation.

2.1.3 Logging

This design leverages the full suite of Azure logging services which includes Azure Monitoring, Azure Network Watcher, and Azure Log Analytics. Chapter 6 provides guidance for using the logging services for DoD programs.

2.1.4 Continuous Monitoring

Continuous Monitoring combined with logging provides real time insight into how the software factory is operating and identifies anomalies that may be operational or security risks. This design applies Azure

Policy, Microsoft Defender for Cloud, and uses Microsoft Sentinel as the Security Information and Event Manager (SEIM) to perform continuous monitoring. Chapter 6 provides guidance on using these Microsoft services.

2.2 Applications

The digital platform provides the base capability to develop applications and services for DoD missions. CSP service catalogs include many powerful building blocks that can be customized into mission capable applications. Examples of such applications include web-based applications, data access, and APIs to expose microservices. These are essential components for modern software systems. Chapter 7 provides examples using typical Azure services to implement modern cloud-based systems. The examples focus on building and deploying secure, scalable, production ready software across the global Azure cloud.

3 Prerequisites

This chapter describes prerequisites required to deploy a software factory using this reference design. A comprehensive instruction manual is provided with DoD Cloud IaC that provides greater detail on each of these requirements.

3.1 Azure Active Directory Tenant and Administrator Roles

To leverage DoD Cloud IaC full administrative rights to an empty Azure Active Directory (AAD) Tenant is required.

3.1.1 Create a DoD Cloud IaC provisioning account

Create a DoD Cloud IaC "provisioning account." This account requires:

- AAD Premium P2 license
- Roles: Global Administrator, User Access Administrator, Privileged Role Administrator
- Protected by Conditional Access (CA) Policies and Multi-factor Authentication (MFA)

3.2 Azure Enterprise Agreement and Subscriptions

DoD Cloud IaC requires five named subscriptions that are part of an Enterprise Agreement for Microsoft Azure. DoD Cloud IaC is incompatible with "Pay as You Go," or PAYGO, Azure Subscriptions.

3.2.1 Preparing the Azure Enterprise Agreement and Subscriptions

Obtain an Enterprise Agreement for Azure (if you do not already have one) by working with your Azure seller.

Create five empty subscriptions as part of your Enterprise Agreement:

- Operations
- Development
- Test
- Production
- Forensics

3.3 DoD Cloud IaC Deployment Virtual Machine

The DoD Cloud IaC baseline is deployed from a Virtual Machine (VM) running inside your environment. This VM contains all the code, scripts, and software required to stand up the software factory environments.

3.3.1 Deploy a DoD Cloud IaC provisioning VM

This VM can be deployed directly from the Azure Marketplace (see Section 4.6). Alternatively, the VM can be provisioned manually; this process is described at length in the DoD Cloud IaC Installation Guide. The VM contains the following items:

- DoD Cloud IaC code baseline
- DoD Cloud IaC configuration files
- Supported version of PowerShell, with required PowerShell modules

4 DoD Cloud Infrastructure as Code (DoD Cloud IaC)

This reference design specifies using Infrastructure as Code to create the cloud infrastructure for the DevSecOps environment. The DISA DoD Cloud IaC templates provide a DISA cyber-security approved baseline from which to build an environment that has the attributes for a DevSecOps software factory. This section introduces the DoD Cloud IaC baseline and describes the salient DevSecOps features that any baseline must have.

Figure 4 shows an example of the environment created by DoD Cloud IaC at IL5.

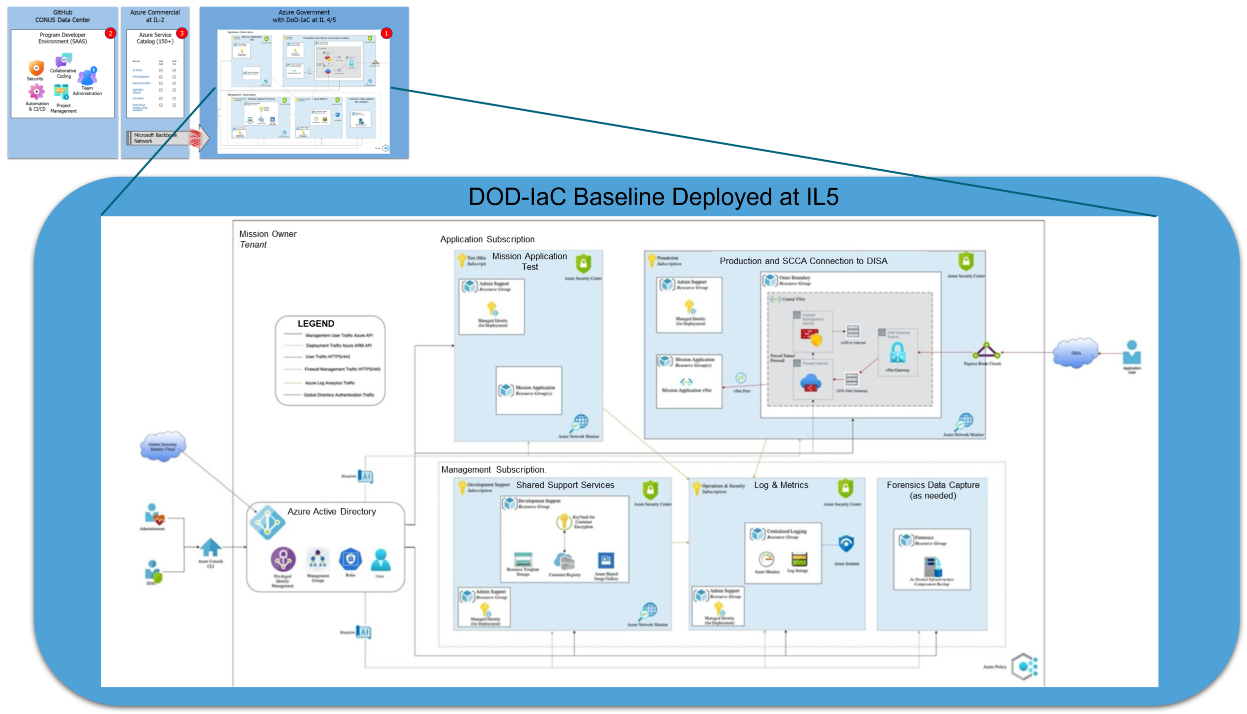


Figure 4: DoD Cloud IaC Baseline providing the IL5 DevSecOps Environment

Deploying code into the environment and orchestrating the DevSecOps pipeline is managed using tools available in GitHub provided via SaaS and described in the next chapter. Code will be programmatically delivered into the IL5 environment and the CI/DC pipelines will move it through test and into production. As shown in the diagram, the DoD Cloud IaC created environment includes the subscriptions that are necessary to operate the software factory.

4.1 DoD Cloud IaC Overview

The DoD Cloud IaC baselines generate preconfigured, preauthorized, Platform as a Service (PaaS)- environments, which exist in the form of IaC templates that organizations can deploy to establish their own decentralized cloud platform. These baselines reduce mission owner security responsibilities by leveraging inheritance from Cloud Service Provider (CSP) PaaS managed services, where host and middleware security, including hardening and patching, is the responsibility of the CSP. The PaaS services utilized by the DoD Cloud IaC baseline have achieved a DISA Provisional Authorization (PA). Whenever possible, DoD Cloud IaC leverages managed security services offered by CSPs over traditional data center tools for improved integration with cloud services. DoD Cloud IaC baselines can be built into DevSecOps pipelines to rapidly deploy the entire environment and mission applications.

While mission owners still must conduct an application-level Assessment and Authorization (A&A), the underlying environment deployed by the DoD Cloud IaC has an authorization from DISA Risk Management Executive (RME) and inheritable controls in eMASS. The inheritance from the PaaS and underlying Infrastructure as a Service (IaaS) accounts for approximately 40% of all controls, significantly expediting the authorization process for mission owners.

4.2 Environment

The DoD Cloud IaC for Azure baseline builds out a secure, multi-subscription environment in Azure, consisting of Test, Production, Shared Services, Centralized Logging, and Forensics subscriptions. Traffic is routed in a hub and spoke model from the Outer-boundary VNET hub with Azure Firewall to the spoke subscriptions. Services utilized in each subscription are properly configured for security and centralized logging.

4.3 DoD Cloud IaC Templates

A complete list of IaC templates is available at the DoD Cloud IaC website[4]. The baseline encompasses IaC for Application and Database Hosting, Container Hosting, Artificial Intelligence/Machine Learning, Internet of Things, Hybrid Cloud computing and other cloud services.

4.4 Cloud-Native Access Point (CNAP)

The DoD Cloud IaC employs Azure tooling as a Cloud Native Access Point. This includes Azure Active Directory as an Identity Provider (IDP) as well as Azure Firewall, Azure Application Gateway including Azure WAF or Azure Front Door.

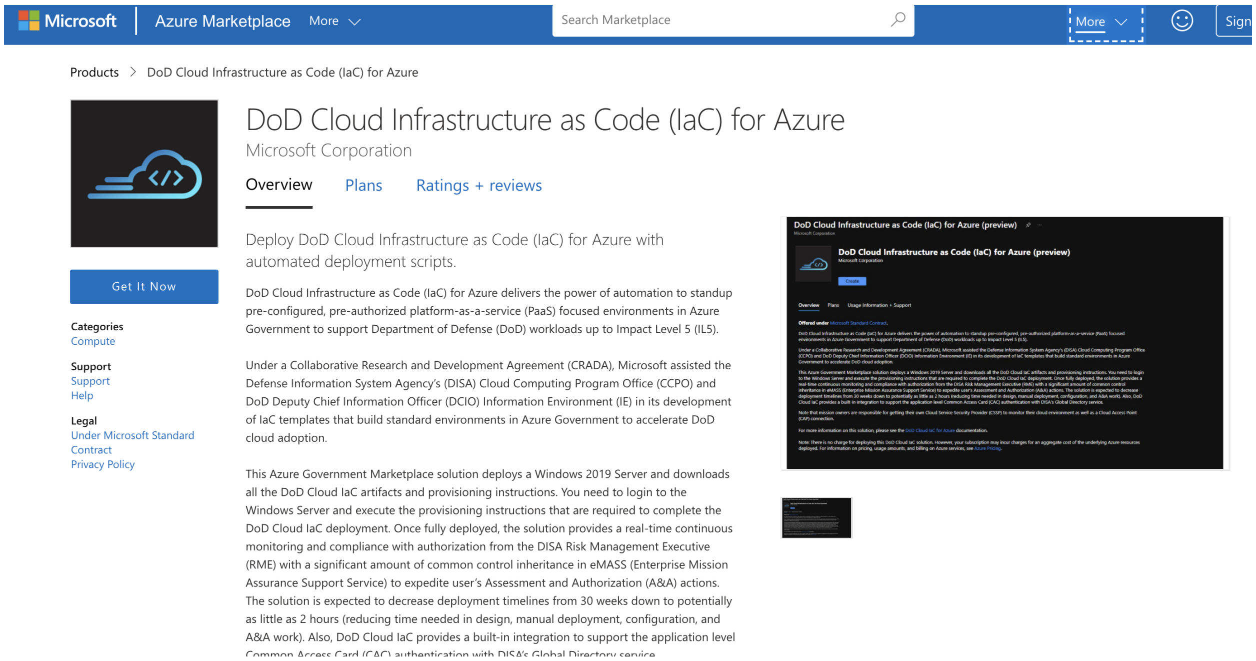
[4]https://www.hacc.mil/Products/DoD-IaC/

4.5 Baseline Updates

DoD Cloud IaC baselines are updated monthly.

4.6 Accessing the DoD Cloud IaC Baselines & Supporting Documentation

The DoD Cloud IaC baselines are Distribution C and limited to the US Government and their contractors. The baseline is available in the Azure Marketplace (*Figure 5*) and as a DISA HaCC repo.



DoD Cloud Infrastructure as Code (IaC) for Azure

Microsoft Corporation

Deploy DoD Cloud Infrastructure as Code (IaC) for Azure with automated deployment scripts.

DoD Cloud Infrastructure as Code (IaC) for Azure delivers the power of automation to standup pre-configured, pre-authorized platform-as-a-service (PaaS) focused environments in Azure Government to support Department of Defense (DoD) workloads up to Impact Level 5 (IL5).

Under a Collaborative Research and Development Agreement (CRADA), Microsoft assisted the Defense Information System Agency's (DISA) Cloud Computing Program Office (CCPO) and DoD Deputy Chief Information Officer (DCIO) Information Environment (IE) in its development of IaC templates that build standard environments in Azure Government to accelerate DoD cloud adoption.

This Azure Government Marketplace solution deploys a Windows 2019 Server and downloads all the DoD Cloud IaC artifacts and provisioning instructions. You need to login to the Windows Server and execute the provisioning instructions that are required to complete the DoD Cloud IaC deployment. Once fully deployed, the solution provides a real-time continuous monitoring and compliance with authorization from the DISA Risk Management Executive (RME) with a significant amount of common control inheritance in eMASS (Enterprise Mission Assurance Support Service) to expedite user's Assessment and Authorization (A&A) actions. The solution is expected to decrease deployment timelines from 30 weeks down to potentially as little as 2 hours (reducing time needed in design, manual deployment, configuration, and A&A work). Also, DoD Cloud IaC provides a built-in integration to support the application level Common Access Card (CAC) authentication with DISA's Global Directory service

DoD Cloud Infrastructure as Code (IaC) for Azure (preview)

Microsoft Corporation

DoD Cloud Infrastructure as Code (IaC) for Azure (preview)

Microsoft Corporation

Offered under Microsoft Standard Contract.

DoD Cloud Infrastructure as Code (IaC) for Azure delivers the power of automation to standup pre-configured, pre-authorized platform-as-a-service (PaaS) focused environments in Azure Government to support Department of Defense (DoD) workloads up to Impact Level 5 (IL5).

Under a Collaborative Research and Development Agreement (CRADA), Microsoft assisted the Defense Information System Agency's (DISA) Cloud Computing Program Office (CCPO) and DoD Deputy Chief Information Officer (DCIO) Information Environment (IE) in its development of IaC templates that build standard environments in Azure Government to accelerate DoD cloud adoption.

This Azure Government Marketplace solution deploys a Windows 2019 Server and downloads all the DoD Cloud IaC artifacts and provisioning instructions. You need to login to the Windows Server and execute the provisioning instructions that are required to complete the DoD Cloud IaC deployment. Once fully deployed, the solution provides a real-time continuous monitoring and compliance with authorization from the DISA Risk Management Executive (RME) with a significant amount of common control inheritance in eMASS (Enterprise Mission Assurance Support Service) to expedite user's Assessment and Authorization (A&A) actions. The solution is expected to decrease deployment timelines from 30 weeks down to potentially as little as 2 hours (reducing time needed in design, manual deployment, configuration, and A&A work). Also, DoD Cloud IaC provides a built-in integration to support the application level Common Access Card (CAC) authentication with DISA's Global Directory service.

Note that mission owners are responsible for getting their own Cloud Service Security Provider (CSSP) to monitor their cloud environment as well as a Cloud Access Point (CAP) connection.

For more information on this solution, please see the DoD Cloud IaC for Azure documentation.

Note: There is no charge for deploying this DoD Cloud IaC solution. However, your subscription may incur charges for an aggregate cost of the underlying Azure resources deployed. For information on pricing, usage amounts, and billing on Azure services, see Azure Pricing.

Figure 5: Screenshot of DoD Cloud IaC in the Azure Marketplace

To access the private preview, go to the Azure Marketplace[5]. Select "Get it Now". Customers need to verify that they have a .mil email address to receive access.

[5] https://aka.ms/DoD-IaC-Azure-Marketplace

5 Azure Software Factory Reference Design

This reference design specifies using GitHub to create the program development environment. GitHub provides DISA cyber-security approved managed capabilities from which to build, test, and deliver software securely within a DevSecOps software factory. This section introduces GitHub and describes the design and configuration patterns required to meet DevSecOps guidelines.

Figure 6 shows an example of the program development environment deployed at FedRAMP Moderate / IL2.

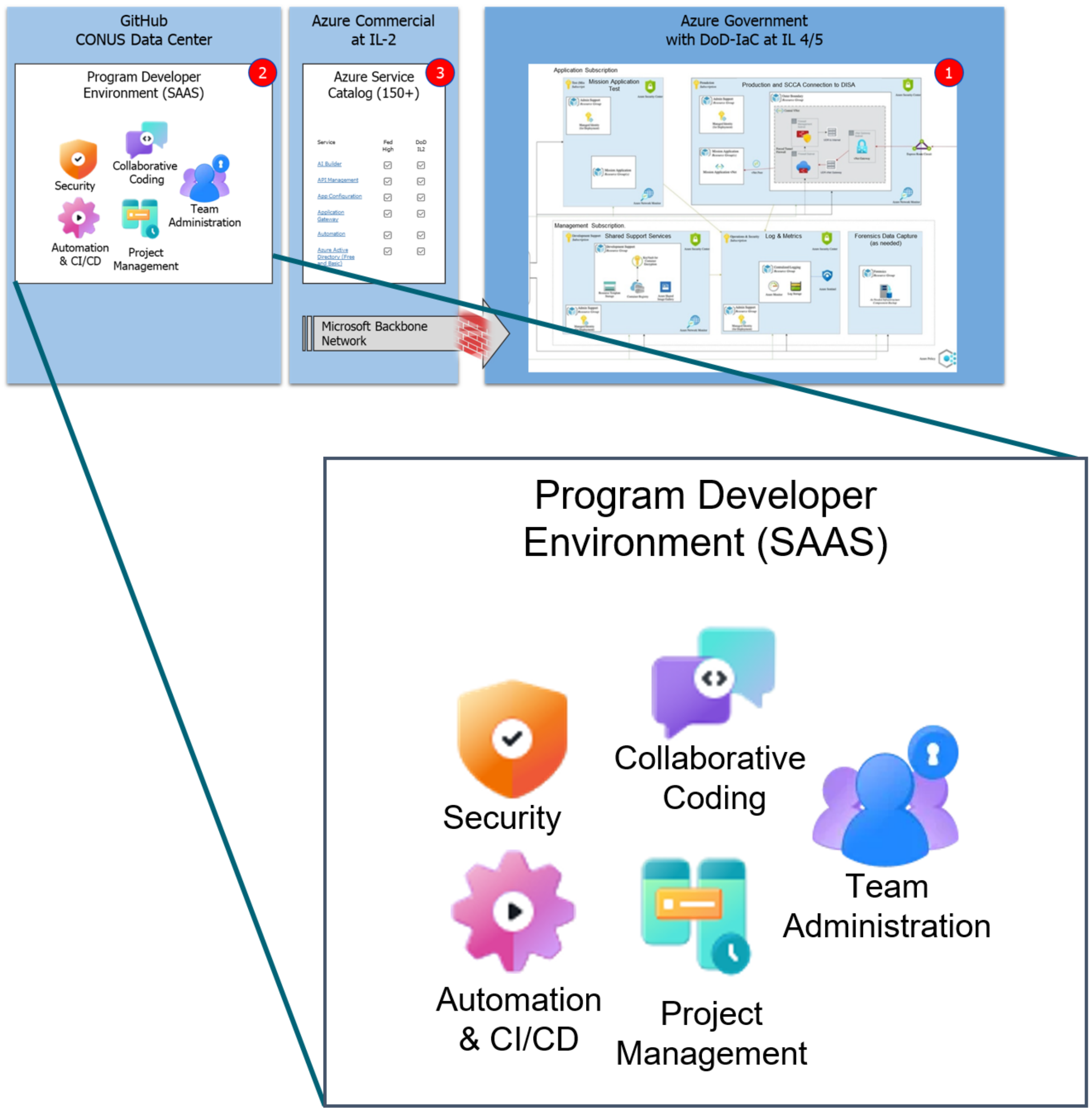

GitHub
CONUS Data Center
Program Developer Environment (SAAS)
2
Azure Commercial at IL-2
Azure Service Catalog (150+)
3
Microsoft Backbone Network
Azure Government with DoD-IaC at IL 4/5
1
Program Developer Environment (SAAS)
Security
Collaborative Coding
Team Administration
Automation & CI/CD
Project Management

Figure 6: GitHub.com providing the Program Developer Environment

Microsoft provides two options for comprehensive DevSecOps tooling with Azure-native integration: **GitHub** and **Azure DevOps**. While both are viable implementation options for a secure Software Factory, this reference design *recommends the use of GitHub.*

GitHub is the appropriate choice for most organizations as it is home to the largest global developer community, it is the long-term successor to Azure DevOps, and it provides a rich, integrated DevSecOps experience and a large ecosystem of relevant capabilities.

For a brief discussion on Azure DevOps, see *Azure DevOps* in the *Additional Azure-Native DevSecOps Offerings* appendix.

5.1 GitHub

GitHub is a software development platform, purchased by Microsoft in 2018. Its hosted version, GitHub.com, is home to much of the world's software and most of the Open-Source community. It provides many essential DevSecOps capabilities, including source code repositories, continuous integration/deployment automation, issue tracking, and more.

Microsoft offers GitHub as several distinct products. These include free and enterprise versions which offer the same general suite of capabilities but vary in the number of supported team members and in terms of security features. **This reference design leveraged GitHub.com and recommends this option wherever possible** given its free nature and versatility. A team can be up and running on GitHub.com with a free account within minutes. Start at https://github.com/team.

For more information on GitHub's other offerings, see the *GitHub Offerings* in the *Additional Azure-Native DevSecOps Offerings* appendix.

The bulk of this chapter is focused on specifically using GitHub as the *CI/CD Orchestrator* in a Software Factory. GitHub, additionally, provides capabilities that span many phases of the DevSecOps lifecycle, as defined in the *DevSecOps Fundamentals Guidebook*[6]. A mapping of these capabilities to DevSecOps lifecycle phases is presented in the Appendix.

When using any GitHub product as the CI/CD Orchestrator in a DevSecOps Software Factory, the following are best practice recommendations. Each recommendation will be explained in greater detail in the subsequent sub-sections.

- Use **Organizations, Teams, and Private Repositories** to create a collaborative, secure and flexible structure that enhances developer productivity and secures access to artifacts.
- Use **GitHub Actions** to orchestrate secure CI/CD pipelines.
- Leverage **GitHub-hosted** runners, which are hosted in Azure Commercial[7] for workloads up to IL5.
- Optionally, deploy **Self-hosted Runners** inside your Azure environment to securely perform CI/CD actions for workloads up to IL5.

[6] https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOps%20Fundamentals%20Guidebook-DevSecOps%20Tools%20and%20Activities_DoD-CIO_20211019.pdf

[7] https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners#cloud-hosts-for-github-hosted-runners

- Use **OpenID Connect (OIDC) in Azure**[8] to access Azure resources securely and limit the scope of access via Azure Active Directory RBAC.

Note: All traffic between Azure Commercial and Azure Government stays entirely on the Microsoft backbone; Azure Government directly peers to the commercial Microsoft Azure network[9]. This is a significant security feature for Azure and GitHub customers.

5.2 Organizations, Teams, and Private Repositories

> *Use **Organizations, Teams, and Private Repositories** to create a collaborative, secure and flexible structure that enhances developer productivity and secures access to artifacts.*

GitHub provides users the ability to create Organizations[10] that allows groups of users to collaborate across one or more projects (repos). You can invite people to become members of an organization and assemble them into Teams that reflect the organization's structure with cascading access permissions. Members in an organization can be granted Roles such as *Owners*, *Members*, *Billing Managers* and *Security Managers*, that each have specific capabilities.

GitHub Private Repositories[11] allow organizations to set the visibility of a repository to *Public* or *Private*. Public repositories are accessible to everyone on the internet; Private repositories are only accessible to you, people you explicitly share access with, and, for organization repositories, certain organization members.

The relationship between GitHub Organizations, Teams, and Repositories is shown in *Figure 7*.

[8] https://docs.github.com/en/actions/deployment/security-hardening-your-deployments/configuring-openid-connect-in-azure

[9] https://docs.microsoft.com/en-us/azure/azure-government/documentation-government-plan-security#environment-isolation

[10] https://docs.github.com/en/organizations

[11] https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/managing-repository-settings/setting-repository-visibility

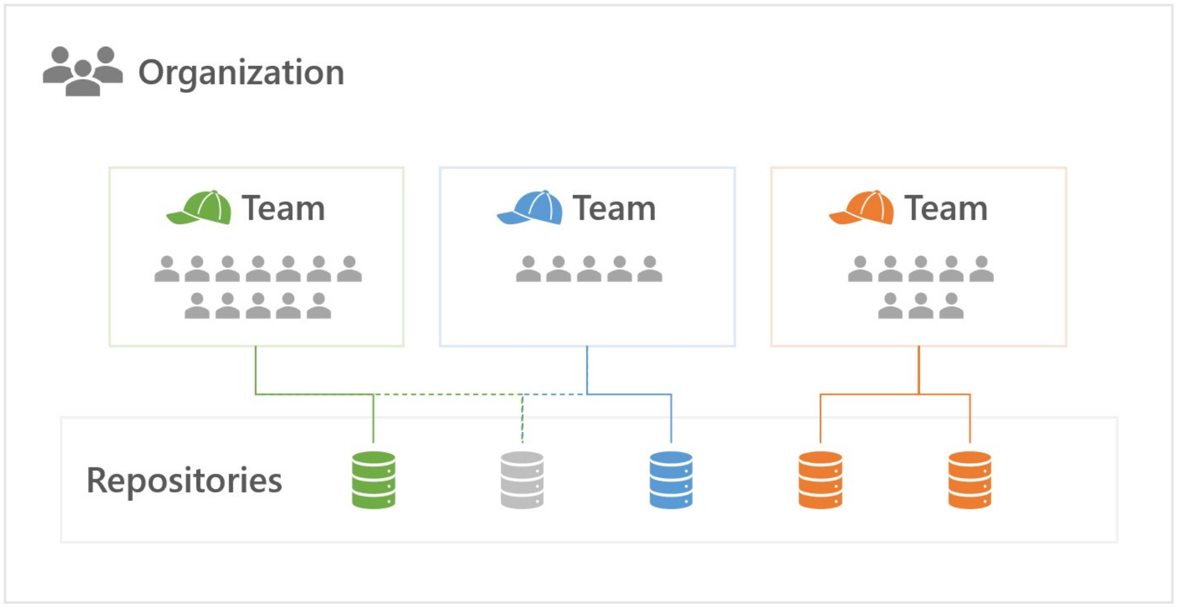


Figure 7: GitHub Organizations, Teams, and Repositories

5.2.1 How to use GitHub Organizations, Teams, and Repositories

Create or join a GitHub Organization if you are going to be servicing multiple Teams. Create Teams to help manage permissions and access to individual repositories. Create a repository for any product, service, or capability being developed by your Teams. All these organizational entities can be configured to Public or Private based on your individual needs.

5.2.2 Guidance for GitHub Organizations, Teams, and Repositories

- Organizations should use Private Repositories for custom or configured software systems in IL4 and higher environments.
- Organizations may choose to make some repositories Public and Open-Source, following any applicable organization policies.

5.3 GitHub Actions

> *Use GitHub Actions to orchestrate secure CI/CD pipelines.*

The software factory should use GitHub Actions to automate the CI/CD pipeline processes defined in the DevSecOps lifecycle plan phase which are Plan, Develop, Build, Test, Release & Deliver, Deploy, Operate, and Monitor. While there is no "one size fits all" for CI/CD processes, it is important that each software team embrace the DevSecOps culture and define processes that suit its software system architectural choices. The CI/CD process is illustrated in *Figure 8*.

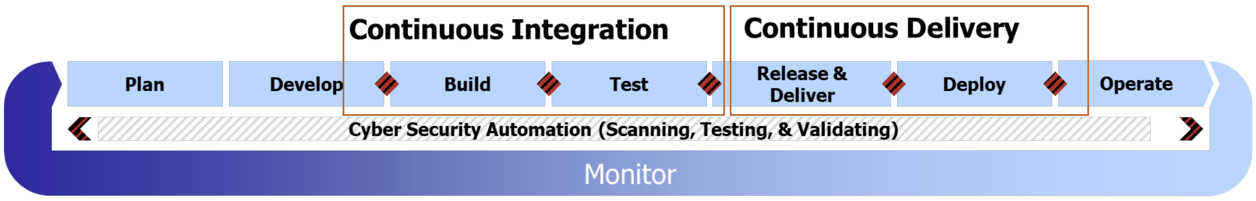


Figure 8: CI/CD Overview

GitHub Actions[12] allow you to continuously build and test code to verify that commits do not introduce errors. More frequent commits reduce the total volume of code change that developers must debug between commits. This is a key DevSecOps practice.

5.3.1 Implementing GitHub Action workflows

To construct a workflow, teams can pull in Actions from a rich marketplace. Table 1 maps the CI/CD process from *Figure 8* to important pipeline activities and the GitHub Actions that this Reference Design recommends to implement them.

Table 1: Recommended GitHub Actions for implementing key CI/CD pipeline activities

Phase	Activity and Purpose	Action Name
Build	Compile and/or link	Framework specific (*e.g.*, Java[13], .NET[14], Python[15])
Build	Dependency vulnerability checking; scans your pull requests for dependency changes and will raise an error if any new dependencies have existing vulnerabilities.	**Dependency Review[16]**

[12]https://docs.github.com/en/actions

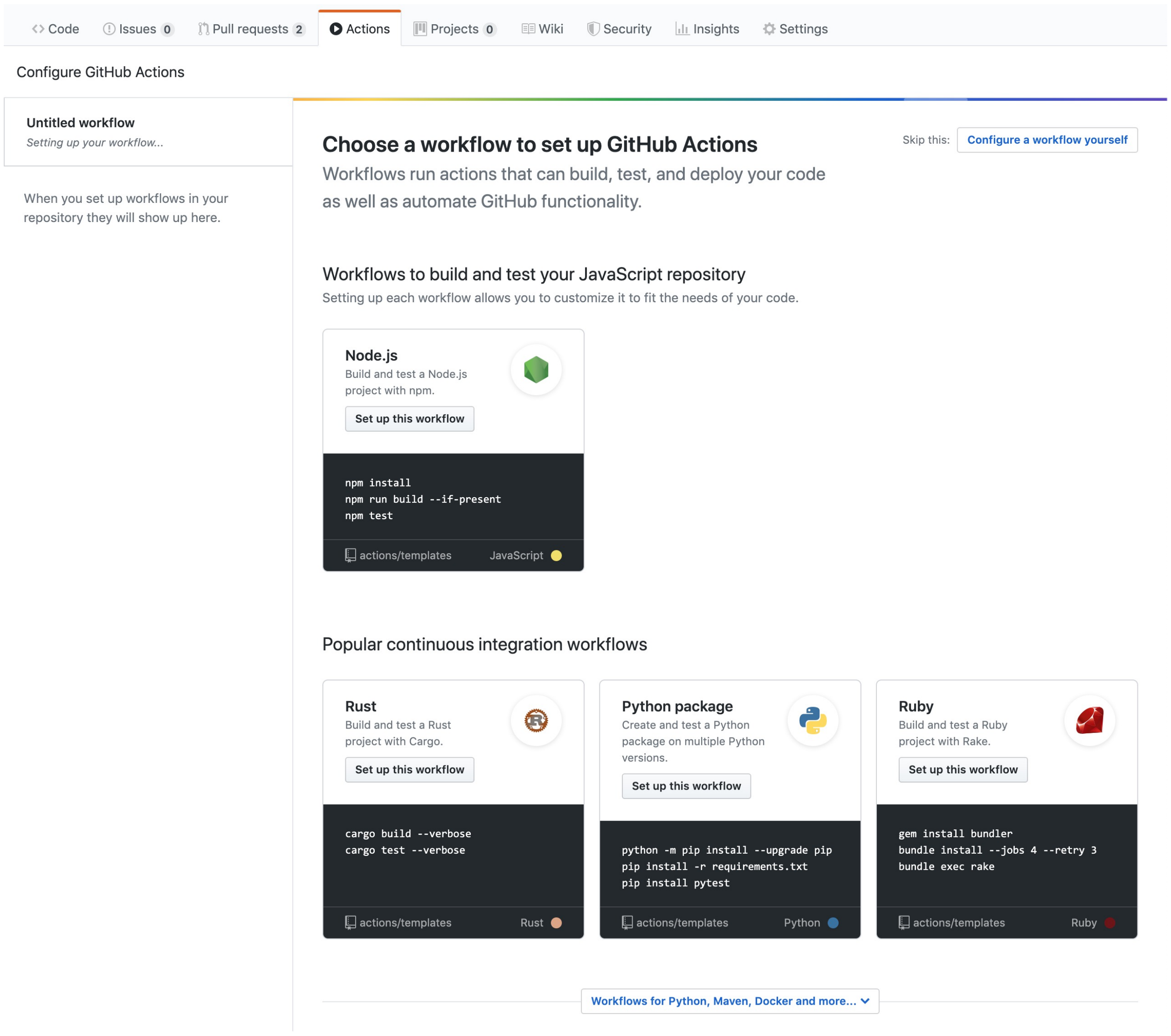
Phase	Activity and Purpose	Action Name
Test	Code Quality; generate and analyze Code Coverage reports with CodeCov.	**CodeCov**[17]
Test	Software Bill of Materials; generate an SBOM using Syft[18] by scanning the workspace directory and uploading an artifact SBOM in SPDX format.	**Anchore SBOM**[19]
Test	OWASP ZAP; runs the ZAP spider against the specified target (by default with no time limit) followed by an optional ajax spider scan and then a full active scan before reporting the results. The alerts will be maintained as a GitHub issue in the corresponding repository.	**OWASP ZAP Full Scan**[20]
Test	Automated Tests; runs automated tests included in your code using framework-specific tools like pytest, dotnet test, etc.	Framework specific (*e.g.*, Java, .NET, Python)

Phase	Activity and Purpose	Action Name
Test	Vulnerability Scanning; runs GitHub's industry-leading semantic code analysis engine, CodeQL, against a repository's source code to find security vulnerabilities. It then automatically uploads the results to GitHub so they can be displayed in the repository's security tab. CodeQL runs an extensible set of queries, which have been developed by the community and the GitHub Security Lab to find common vulnerabilities in your code.	**GitHub CodeQL**[21]
Deploy	Deployment; verified Azure actions to deploy on a variety of PaaS resources	**Official Azure Deployment Actions**[22]
Continuous	Package and dependency check; monitor vulnerabilities in dependencies used in your project and keep your dependencies up-to-date.	**GitHub Dependabot**[23]

GitHub Actions workflows are written as YAML files and stored in the repository under the `.github/worklfows/` directory. If creating a new workflow file, you can either start from scratch or

[13] https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-maven
[14] https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-net
[15] https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python
[16] https://github.com/marketplace/actions/dependency-review
[17] https://github.com/marketplace/actions/codecov
[18] https://github.com/anchore/syft
[19] https://github.com/marketplace/actions/anchore-sbom-action
[20] https://github.com/marketplace/actions/owasp-zap-full-scan
[21] https://github.com/github/codeql-action
[22] https://aka.ms/azure-gh-actions
[23] https://docs.github.com/en/code-security/dependabot

start from one of the available templates. This is shown in *Figure 9*.



When you set up workflows in your repository they will show up here.

Choose a workflow to set up GitHub Actions

Workflows run actions that can build, test, and deploy your code as well as automate GitHub functionality.

Workflows to build and test your JavaScript repository

Setting up each workflow allows you to customize it to fit the needs of your code.

Node.js

Build and test a Node.js project with npm.

```
npm install
npm run build --if-present
npm test
```

Popular continuous integration workflows

Rust

Build and test a Rust project with Cargo.

```
cargo build --verbose
cargo test --verbose
```

Python package

Create and test a Python package on multiple Python versions.

```
python -m pip install --upgrade pip
pip install -r requirements.txt
pip install pytest
```

Ruby

Build and test a Ruby project with Rake.

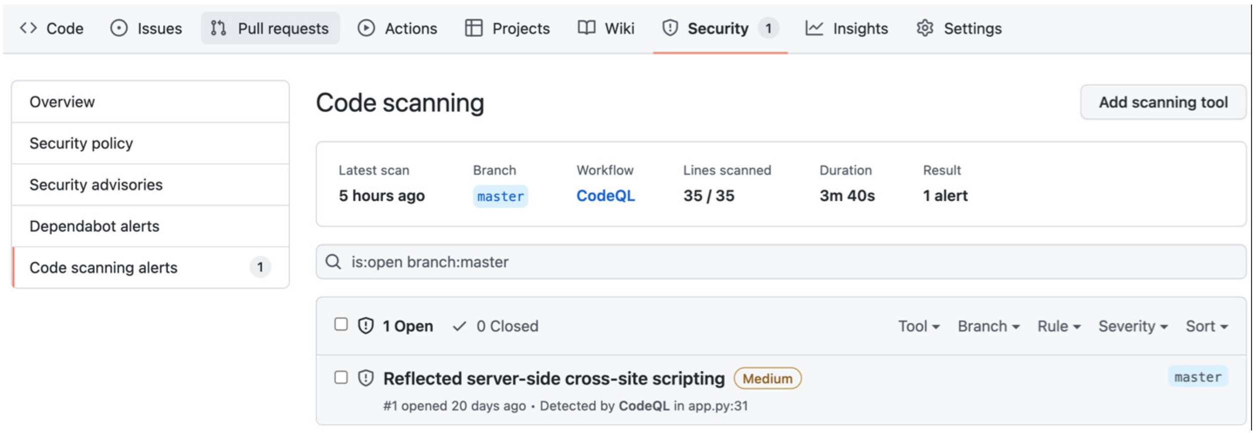
```
gem install bundler
bundle install --jobs 4 --retry 3
bundle exec rake
```

Figure 9: Creating a new workflow to set up GitHub Actions

5.3.2 Guidance for using GitHub Actions

- Organizations should have an *automated* periodic build (weekly or daily) to ensure that code quality, vulnerabilities, and dependencies are scanned on a regular basis.
- Organizations should leverage GitHub's marketplace for GitHub Actions, including Microsoft-developed Actions to deploy to Azure resources.

- Organizations should use GitHub's best practices in using Actions, that include Security Hardening[24] best practices.
- Teams should include CI steps in their GitHub Action workflows for, at a minimum, linters (style and formatting), security checks, code coverage, and functional checks.
- Teams should include DevSecOps stages in their GitHub Action workflows for, at a minimum, Build (if applicable), Release & Deliver, Deploy and (Smoke Tests?).
- Teams should enable Security and Analysis settings, including Dependency Graph and Dependabot alerts. Security alerts from CodeQL display in the Security scanning alerts section under the Security tab[25].



Code scanning

Latest scan	Branch	Workflow	Lines scanned	Duration	Result
5 hours ago	master	CodeQL	35 / 35	3m 40s	1 alert

is:open branch:master

1 Open ✓ 0 Closed

Reflected server-side cross-site scripting Medium master
#1 opened 20 days ago • Detected by CodeQL in app.py:31

Figure 10: Security scanning alerts

5.4 Action Runners - GitHub-hosted and Self-hosted

> *Deploy self-hosted runners inside your Azure environment to securely perform CI/CD actions.*

GitHub Actions are executed by software called "runners." Runners are available in the following two options:

GitHub-hosted runners[26]. Execution is handled entirely by GitHub without any administration needed on behalf of the user. GitHub-hosted runners are provisioned by GitHub in Azure's Commercial Regions. Traffic between Azure Commercial and Government regions traverses Microsoft's Azure backbone, and never over the open internet.

Self-hosted runners[27]. Installed, configured, administered by the user, but can run virtually anywhere.

[24]https://docs.github.com/en/actions/security-guides/security-hardening-for-github-actions
[25]https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/enabling-features-for-your-repository/managing-security-and-analysis-settings-for-your-repository
[26]https://docs.github.com/en/actions/using-github-hosted-runners
[27]https://docs.github.com/en/actions/hosting-your-own-runners

Runner updates occur automatically either when a new job is assigned to them or within a week of release. An organization may choose to deploy self-hosted runners inside their cloud environment to provide an added layer of security and an added layer of control.

5.4.1 Action Runners in architectural context

Organizations can use GitHub-hosted, self-hosted runners, or a combination of both. *Figure 10* shows an architecture with both GitHub-hosted runners (running in Azure Commercial) and self-hosted runners (running inside Azure Government).

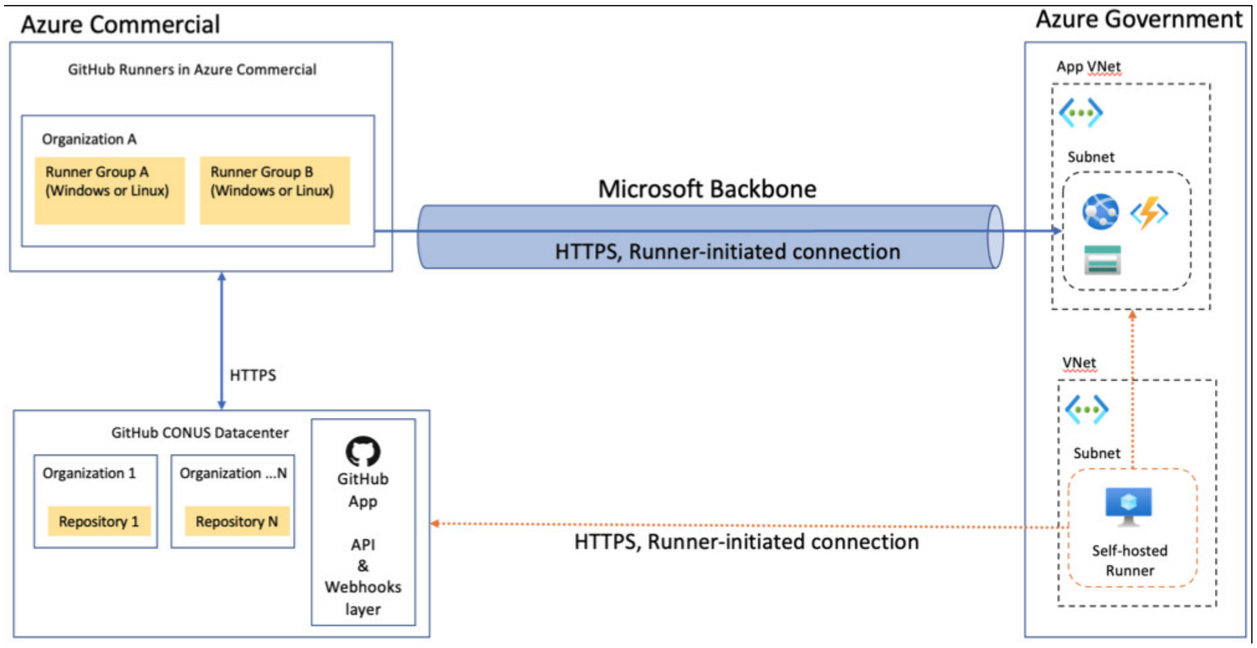


Figure 11: Architecture - GitHub Action Runners in Context

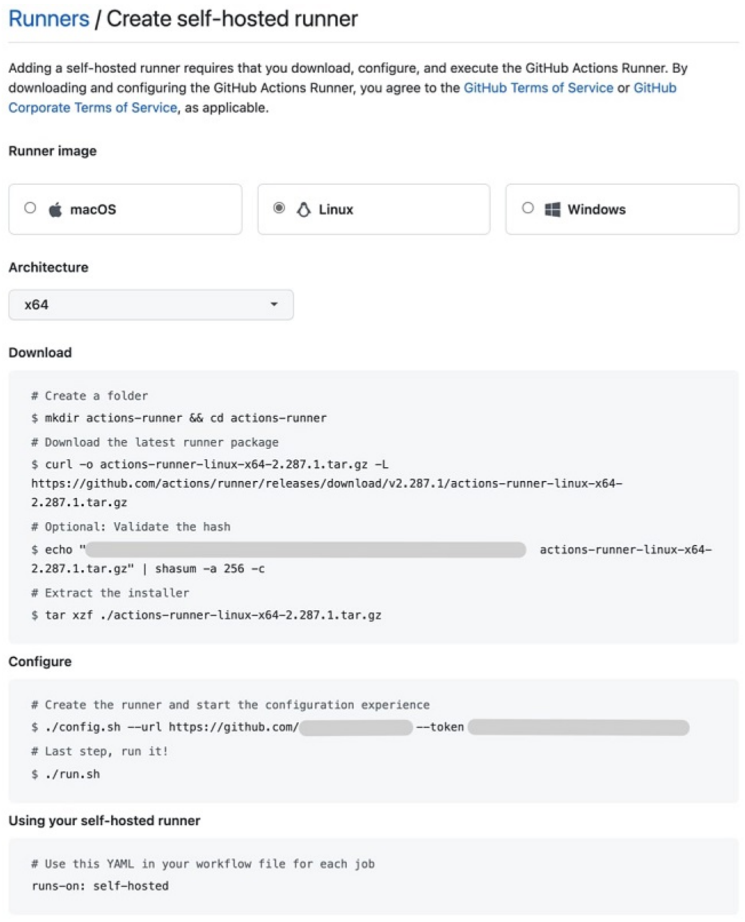
The three top-level components to this architecture include the CI/CD orchestrator (GitHub, lower left), GitHub-hosted runners (Azure Commercial, upper left), and the application hosting environment (Azure Government, right).

1. Teams work in a managed GitHub private repository. This repository is hosted in a CONUS GitHub Datacenter. All runners communicate with GitHub over HTTPS, *runner-initiated* connections.
2. **GitHub-hosted runners** are managed entirely by GitHub and execute inside Azure Commercial Virtual Machines. These runners perform CI/CD automation actions to build, test, scan, and deploy software to the DoD Cloud IaC hosting environment in Azure Government across the Microsoft Backbone. This ensures no communications travel across the open internet.

3. **Self-hosted runners** execute inside the secured Azure Government environment that was provisioned by DoD Cloud IaC. Self-hosted runners can be provisioned across an array of Azure compute options, from Virtual Machines to serverless Azure Functions. Runners communicate with GitHub via OAuth authenticated HTTPS *long poll* connections. This one-way poll watches for queued jobs and pulls code into your environment for action. Artifacts created by the self-hosted runners are **deployed to their intended destination** (*e.g.,* Azure App Service) without ever leaving the secured Azure Government environment.

GitHub-hosted runner implementation. GitHub-hosted runners are provided without additional configuration.

Self-hosted runner implementation. The instructions to create a self-hosted runner are provided when you configure a runner in GitHub.com. *Figure 12* shows the process for creating a self-hosted runner. Additional configuration options can be found online in the linked GitHub self-hosted documentation.



Runners / Create self-hosted runner

Adding a self-hosted runner requires that you download, configure, and execute the GitHub Actions Runner. By downloading and configuring the GitHub Actions Runner, you agree to the GitHub Terms of Service or GitHub Corporate Terms of Service, as applicable.

Runner image

macOS [✓ Linux] Windows

Architecture

x64

Download

```
# Create a folder
$ mkdir actions-runner && cd actions-runner
# Download the latest runner package
$ curl -o actions-runner-linux-x64-2.287.1.tar.gz -L
https://github.com/actions/runner/releases/download/v2.287.1/actions-runner-linux-x64-
2.287.1.tar.gz
# Optional: Validate the hash
$ echo "[redacted]  actions-runner-linux-x64-
2.287.1.tar.gz" | shasum -a 256 -c
# Extract the installer
$ tar xzf ./actions-runner-linux-x64-2.287.1.tar.gz
```

Configure

```
# Create the runner and start the configuration experience
$ ./config.sh --url https://github.com/[redacted] --token [redacted]
# Last step, run it!
$ ./run.sh
```

Using your self-hosted runner

```
# Use this YAML in your workflow file for each job
runs-on: self-hosted
```

Figure 12: Self-hosted runner instructions

5.4.2 Guidance for using Action Runners

- Organizations workloads up to IL5 should use GitHub.com with GitHub-hosted runners. All communication between the Runners and Azure Government is on the Microsoft backbone and not the Open Internet.
- Organizations with IL5 workloads can optionally use self-hosted runners.

5.5 Using GitHub Codespaces (Dev Containers)

GitHub Codespaces[28], or dev containers, are Docker containers that are specifically configured to provide a full-featured development environment with Visual Studio Code[29], running in a Chrome/Edge browser window. *Figure 13* shows that whenever you work in a Codespace, you are using a dev container on a virtual machine running in an Azure region; the actual region is dependent on your geographic location. Codepsaces allow developers to use the full capabilities of Visual Studio Code (VSCode) with its associated Source Code Repository.

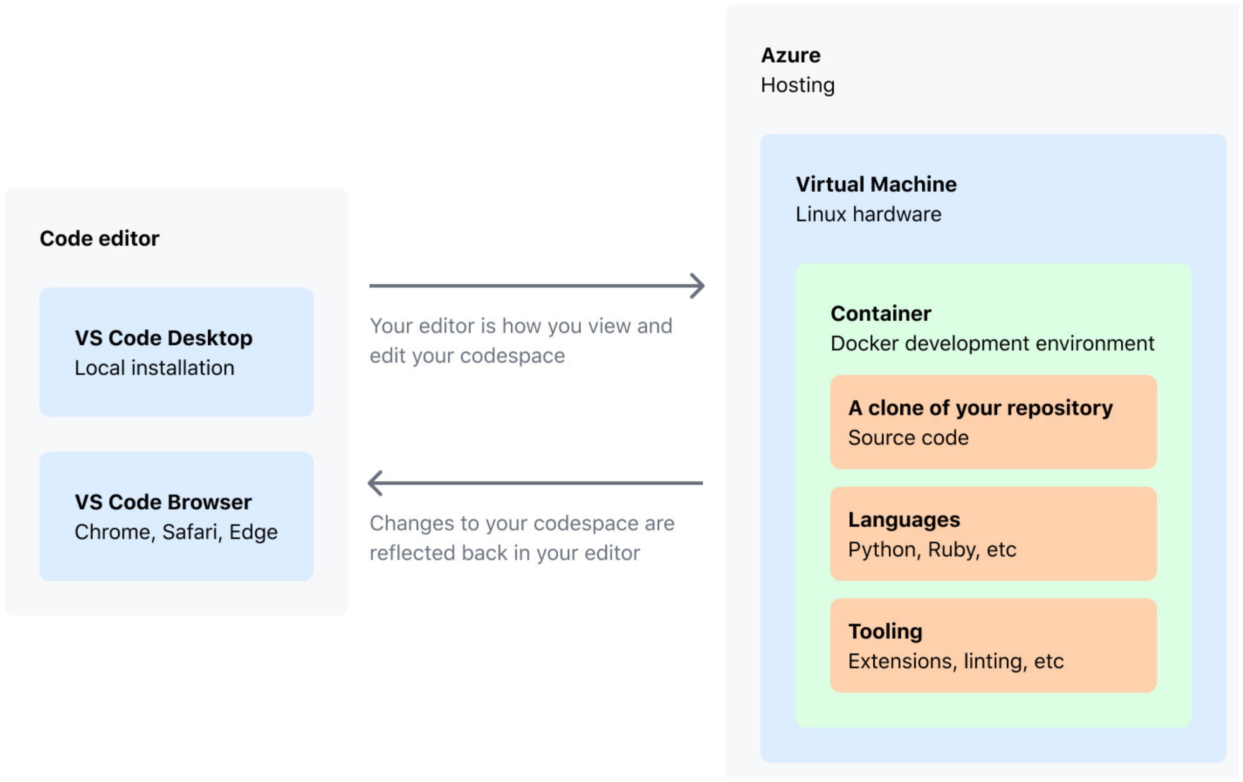


Figure 13: Codespace hosting

[28] https://github.com/features/codespaces
[29] https://code.visualstudio.com/

5.5.1 Standardized dev environments

Codespaces allows teams to provision standardized development environments that are built on customized Docker containers and versioned configuration files, preventing environment drift. Teams can ramp up rapidly using any device that has Chrome or Edge, although users may also can also choose use run Codespaces inside VSCode. Users will continue to author code and use Git version control as they would from any other device, except that it is now available via a browser.

5.5.2 Guidance for using Codespaces

Although using Codespaces is not mandatory, Organizations can use Codespaces to rapidly become productive on controlled environments available on any device that supports Chrome or Edge. Using Codespaces incurs a fee[30].

5.6 Secure Access to Azure

> *Use **OpenID Connect (OIDC) in Azure** to access Azure resources securely and limit the scope of access via Azure Active Directory RBAC.*

The Azure Login Action[31] supports two different methods for authentication with Azure.

OpenID Connect (OIDC) with an Azure service principal using a Federated Identity Credential[32] is the recommended method. It enables developers to secure their deployments to Azure through OpenID Connect's integration between Azure AD and GitHub Actions. This alleviates the need for managing long-lived cloud credentials in the GitHub Actions secret store—meaning that teams will not have to manage credentials in both Azure and GitHub. These capabilities also minimize the chances of service downtime due to expired credentials.

Service principal with secrets[33] uses an Azure Active Directory Service Principal. The resulting JSON is stored as a *Secret* in GitHub and used in an Azure Login Action.

5.6.1 Establishing the secure connection from Action Runner to Azure

See the referenced link in Microsoft's documentation for detailed implementation steps for OIDC. The high-level steps are as follows:

[30] https://docs.github.com/en/billing/managing-billing-for-github-codespaces/about-billing-for-codespaces
[31] https://github.com/marketplace/actions/azure-login
[32] https://docs.github.com/en/actions/deployment/security-hardening-your-deployments/configuring-openid-connect-in-azure
[33] https://docs.microsoft.com/en-us/azure/developer/github/connect-from-azure?tabs=azure-portal%2Clinux#use-the-azure-login-action-with-a-service-principal-secret

1. Create an Azure Active Directory (AAD) application and service principal.
2. Add GitHub's Federated Credentials.
3. Create secrets inside GitHub to support the login action.
4. Add the Azure Login action (`azure/login`) to the GitHub Action workflow and configure accordingly.
5. Verify that the Action can run successfully and login to Azure.

5.6.2 Guidance for connections to Azure

- Organizations should use OIDC within workflows to deploy to Azure whenever possible.

6 Logging and Continuous Monitoring

"Inspect and log all traffic before acting" is a key principle in the NSA's zero trust security model. In the cloud it is possible to enable comprehensive logging to support this principle. By using DoD Cloud IaC per design pattern 2–Use Infrastructure as Code to create the cloud infrastructure for the DevSecOps environment–the essential logging and monitoring services are enabled by default.

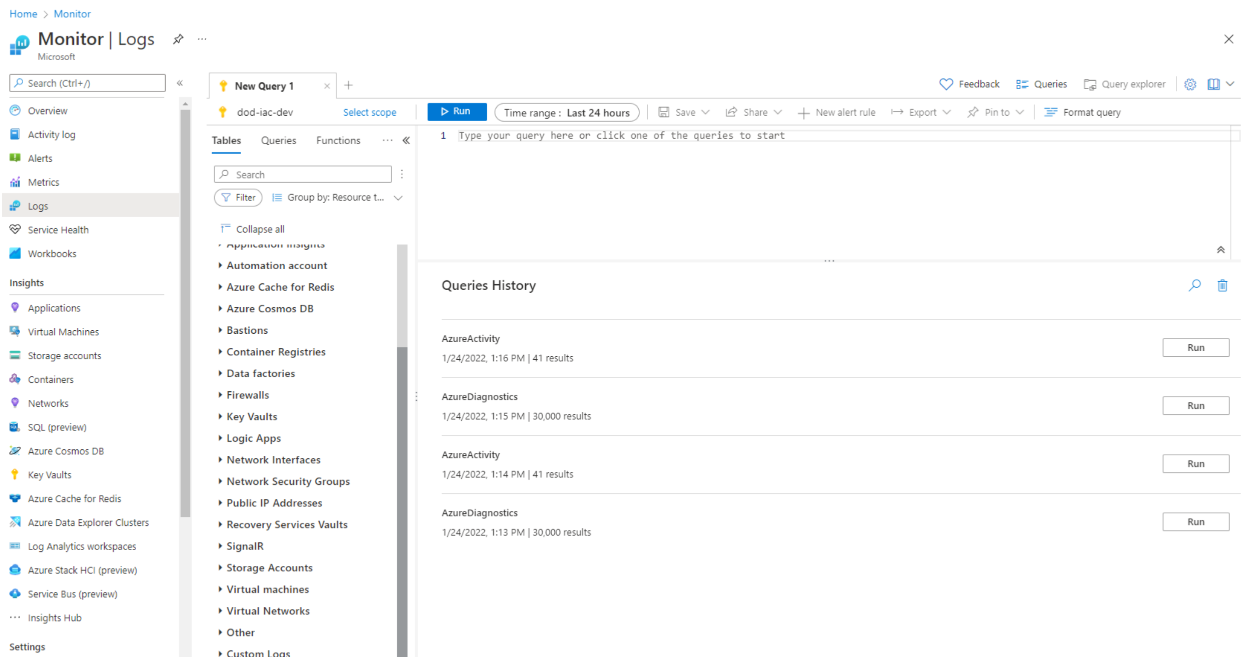
These services are:

- Azure Monitor
- Azure Application Insights
- Azure Network Watcher
- Microsoft Defender for Cloud
- Azure Policies
- Microsoft Sentinel

Those deploying applications in Azure should use each of them. Their purpose along with configuration guidance is provided below.

6.1 Azure Monitor

Monitor is the native logging system in Azure. Azure Active Directory (AD), Azure cloud services, and applications can send logs to it. Azure AD and Azure cloud services come pre-integrated with Monitor; logging need only be enabled. Applications can be configured to send logs to Monitor avoiding the need for external logging frameworks and providing a central place for all logs. Infrastructure-as-a-Service (IaaS) components such as virtual machines and containers can also send logs to Monitor, however, this guidance is focused on platform-as-a-service (PaaS) based deployments.



```
Type your query here or click one of the queries to start
```

AzureActivity
1/24/2022, 1:16 PM | 41 results

AzureDiagnostics
1/24/2022, 1:15 PM | 30,000 results

AzureActivity
1/24/2022, 1:14 PM | 41 results

AzureDiagnostics
1/24/2022, 1:13 PM | 30,000 results

Figure 14: Monitor Screenshot

6.1.1 Implementing Azure Monitor

Azure Monitor is available by default in all Azure environments. It has a built-in dashboard that allows you to audit your cloud resources (*Figure 14*). To achieve comprehensive logging with Monitor you must deploy the cloud services that will hold the logs: a Log Analytics Workspace and one or more Storage Accounts. The Log Analytics Workspace is primary because it provides the ability to search the logs via Azure's SQL-like query language. The Storage Accounts are used for long-term log storage and for non-repudiation (logs can be locked to prevent deletion for specified periods of time). Cloud services send their logs to the Log Analytics Workspace and the Storage Account in parallel via the service Diagnostics feature. This is depicted in *Figure 15*.

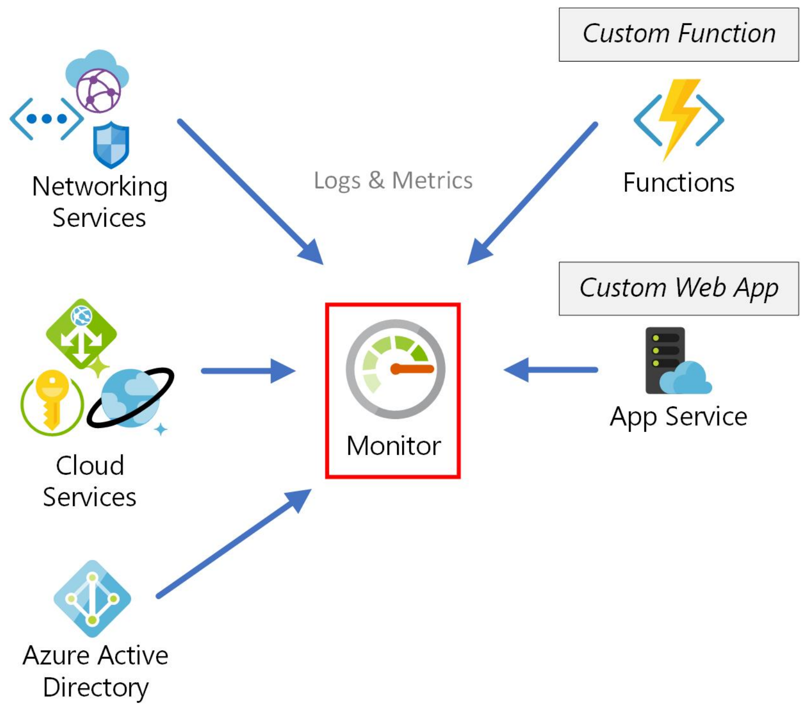


Figure 15: Centralized Logging Design

6.1.2 Guidance for Azure Monitor

- Create a single Log Analytics Workspace for the Azure tenant. Configure the services' diagnostics to send logs to it. This provides centralized logging.
- Diagnostic should be enabled for all cloud services in the tenant and for Azure Active Directory.
- In the service diagnostics enable all logging categories by default. Metrics can be stored as well if needed, however, storing metrics can increase the cost of logging significantly.
- Place this Log Analytics Workspace in a Resource Group and Subscription where permissions can be limited to tenant administrators. Developers and others will be able to see logs for their cloud resources but not for others. It is recommended that this workspace be stored in your primary region.
- One log archive Storage Account is required per region because, unlike Log Analytics Workspaces, the cloud services can only archive logs in a Storage Account located in the same region as they are.
- In custom application code, make use of Azure Monitor libraries so the code can send custom logs to Monitor. This avoids the need for external logging frameworks and puts all the logs in one place.
- To prevent the logs in the Storage Account from tampering, enable the "legal hold" feature on all the blob containers holding the logs. This sets a period, for example, 1 year, where the logs being stored cannot be deleted or changed. Keep in mind that once you set a legal hold on a Storage Account

container you will not be able to remove it for the specified period no matter what your permissions are.

6.2 Azure Application Insights

Application Insights provides detailed telemetry for the cloud services that host or support applications, for example Azure Functions and Azure App Service. It shows end-to-end call tracing across cloud services, performance metrics, failures, and other detailed metrics (See *Figure 16*). Application Insights is a critical tool for monitoring applications in operation.

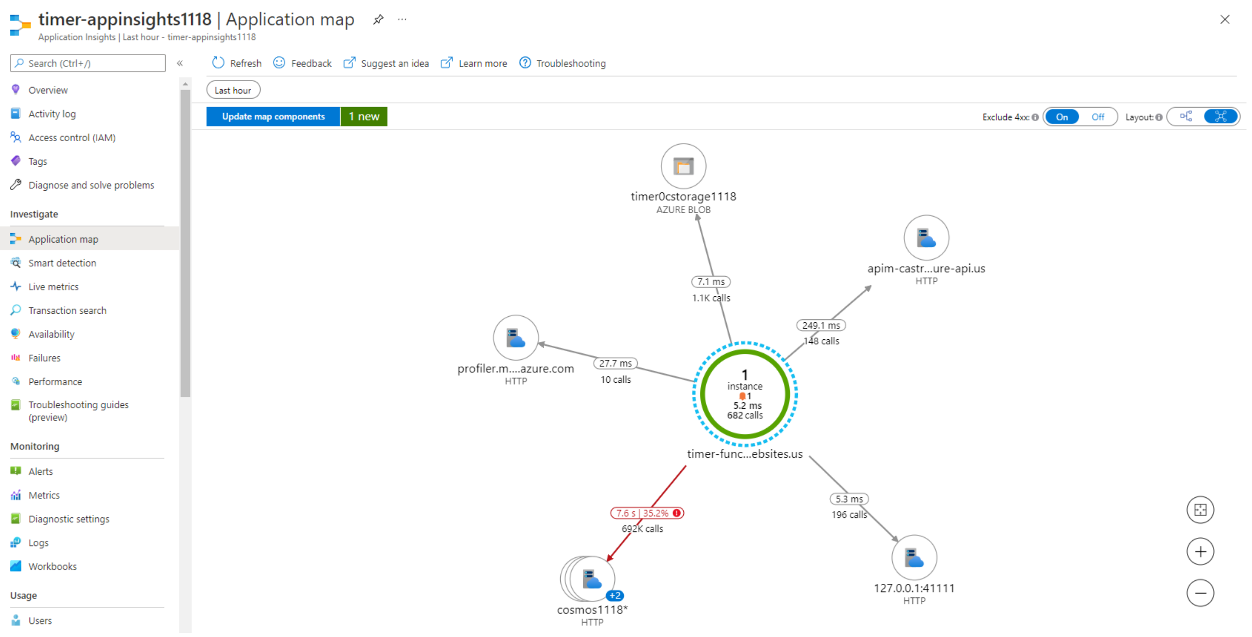


Figure 16: Application Insights Screenshot

6.2.1 Implementing Azure Application Insights

Associate an Application Insights resource with every cloud service that supports it. This is depicted in *Figure 17*.

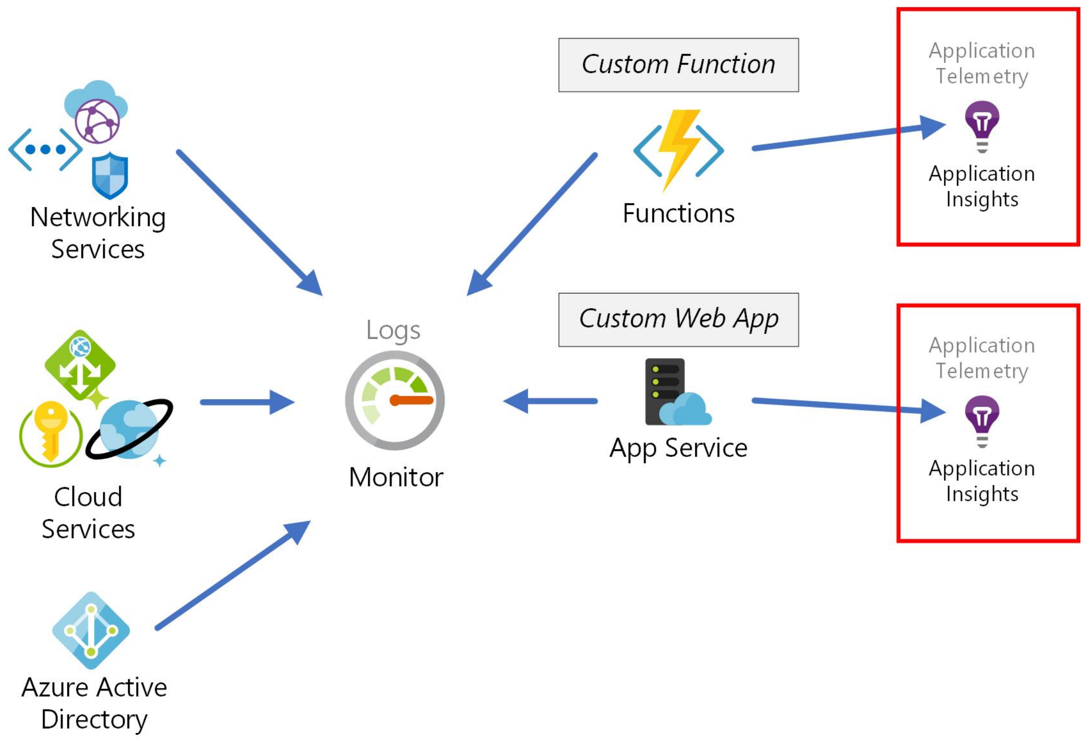


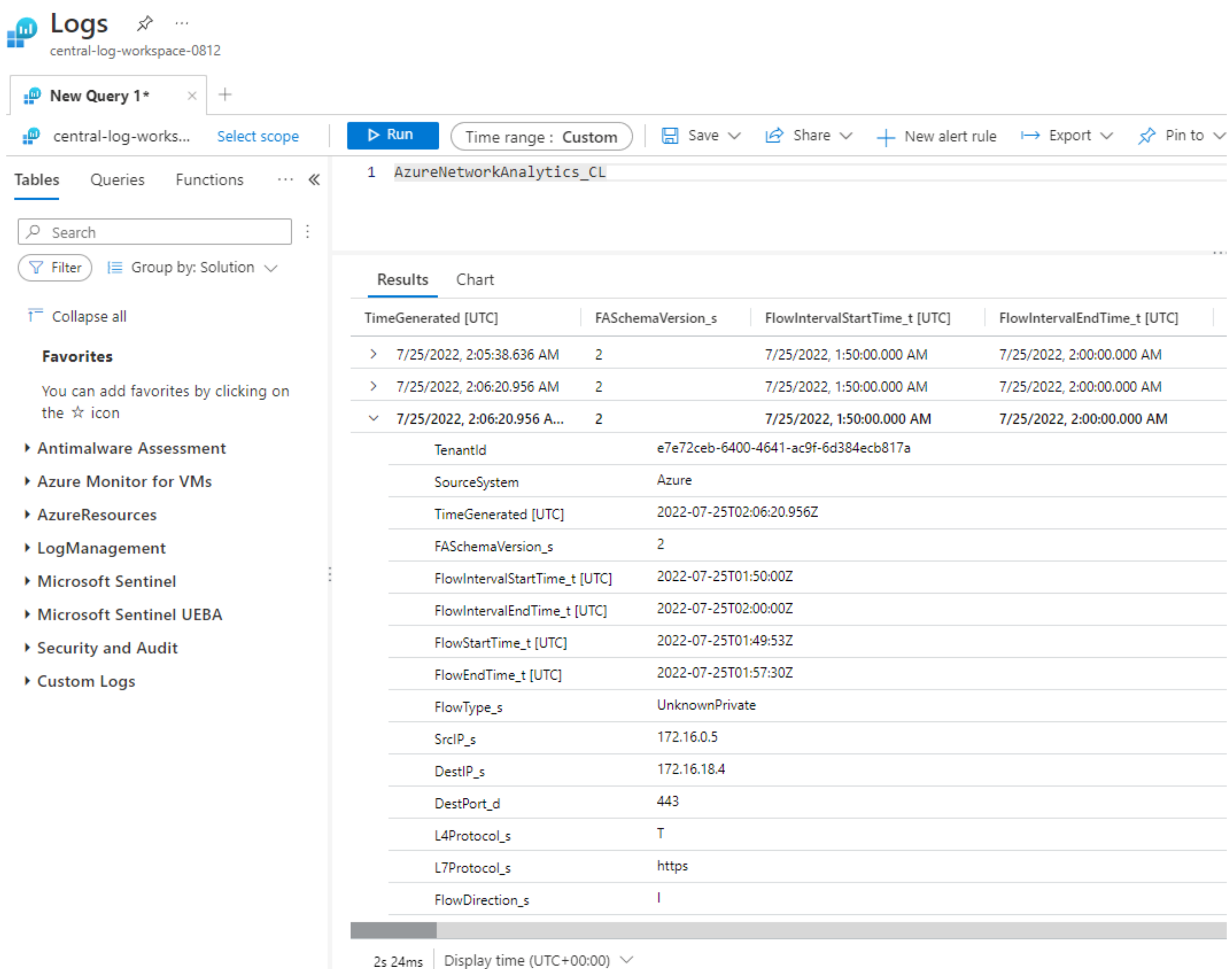
Figure 17: Application Insights Design

6.2.2 Guidance for Azure Application Insights

- Dedicate one Application Insights resource per cloud service. If for example, you have two Azure Function services then create two Application Insights resources, one for each.

6.3 Azure Network Watcher

Network Watcher provides low-level network monitoring. In the PaaS and SaaS contexts, Network Watcher helps monitor TCP/IP traffic from Network Security Groups (NSGs). Its network traffic analytics feature gives administrators an overview of traffic going in and out of the environment at the packet level. The packet logs can be found in Azure Monitor in the `AzureNetworkAnalytics_CL` table as shown in *Figure 18*. With Network Watcher enabled the operational team can monitor traffic passing through your NSGs.



```
AzureNetworkAnalytics_CL
```

TimeGenerated [UTC]	FASchemaVersion_s	FlowIntervalStartTime_t [UTC]	FlowIntervalEndTime_t [UTC]
7/25/2022, 2:05:38.636 AM	2	7/25/2022, 1:50:00.000 AM	7/25/2022, 2:00:00.000 AM
7/25/2022, 2:06:20.956 AM	2	7/25/2022, 1:50:00.000 AM	7/25/2022, 2:00:00.000 AM
7/25/2022, 2:06:20.956 A...	2	7/25/2022, 1:50:00.000 AM	7/25/2022, 2:00:00.000 AM

TenantId	e7e72ceb-6400-4641-ac9f-6d384ecb817a
SourceSystem	Azure
TimeGenerated [UTC]	2022-07-25T02:06:20.956Z
FASchemaVersion_s	2
FlowIntervalStartTime_t [UTC]	2022-07-25T01:50:00Z
FlowIntervalEndTime_t [UTC]	2022-07-25T02:00:00Z
FlowStartTime_t [UTC]	2022-07-25T01:49:53Z
FlowEndTime_t [UTC]	2022-07-25T01:57:30Z
FlowType_s	UnknownPrivate
SrcIP_s	172.16.0.5
DestIP_s	172.16.18.4
DestPort_d	443
L4Protocol_s	T
L7Protocol_s	https
FlowDirection_s	I

Figure 18: Network Watcher Packet Logs

6.3.1 Implementing Azure Network Watcher

Create a Resource Group called "NetworkWatcherRG" in each Subscription that has or could have an NSG. Create a Network Watcher resource in each of these Resource Groups. One Network Watcher is needed for each region. This is depicted in *Figure 19*.

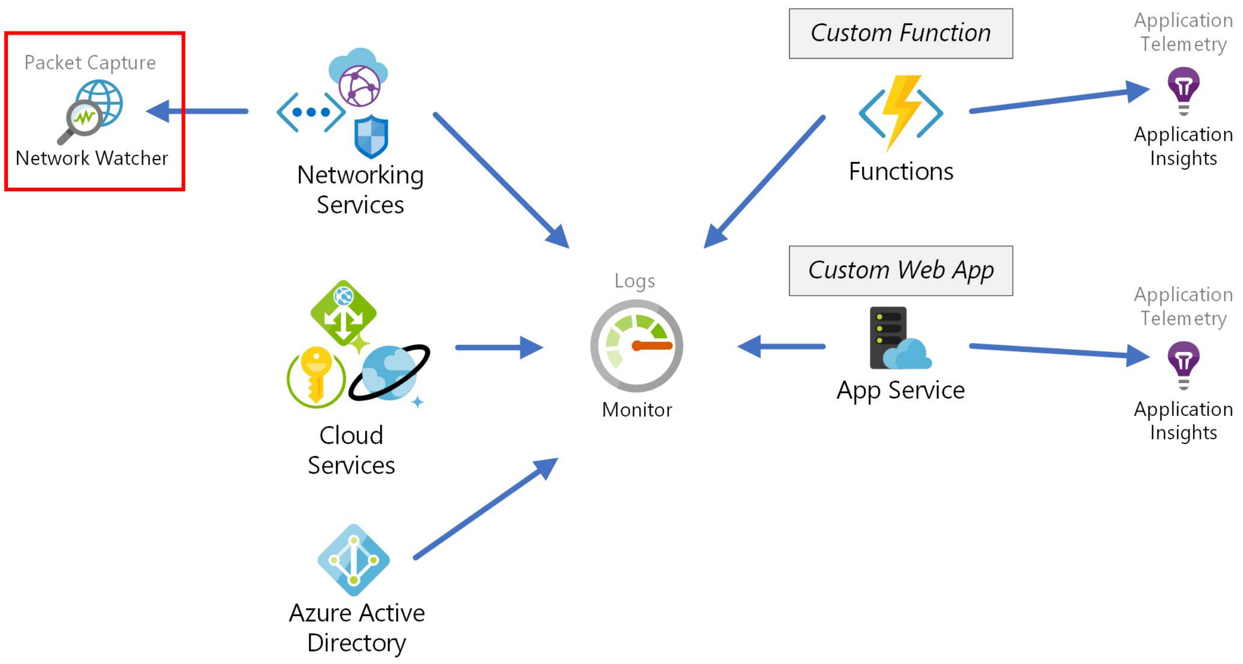


Figure 19: Network Watcher Design

6.3.2 Guidance for Azure Network Watcher

- Enable "Flow Logs" for every NSG. Associate the NSG Flow Log with the appropriate Network Watcher resource based on its region and Subscription. The flow logs should send their data to the centralized logging capability described above.

6.4 Microsoft Defender for Cloud

Microsoft Defender for Cloud (formerly "Azure Security Center") is a built-in feature of Azure that provides continuous security monitoring of the cloud services. It comes with a large set of default rules to catch service misconfigurations. The operations team can view its finding using the Microsoft Defender for Cloud dashboard.

6.4.1 Implementing Microsoft Defender for Cloud

As depicted in *Figure 20*, Microsoft Defender for Cloud monitors all cloud services. By default, Microsoft Defender for Cloud is enabled, therefore no action is required to take advantage of this Microsoft capability.

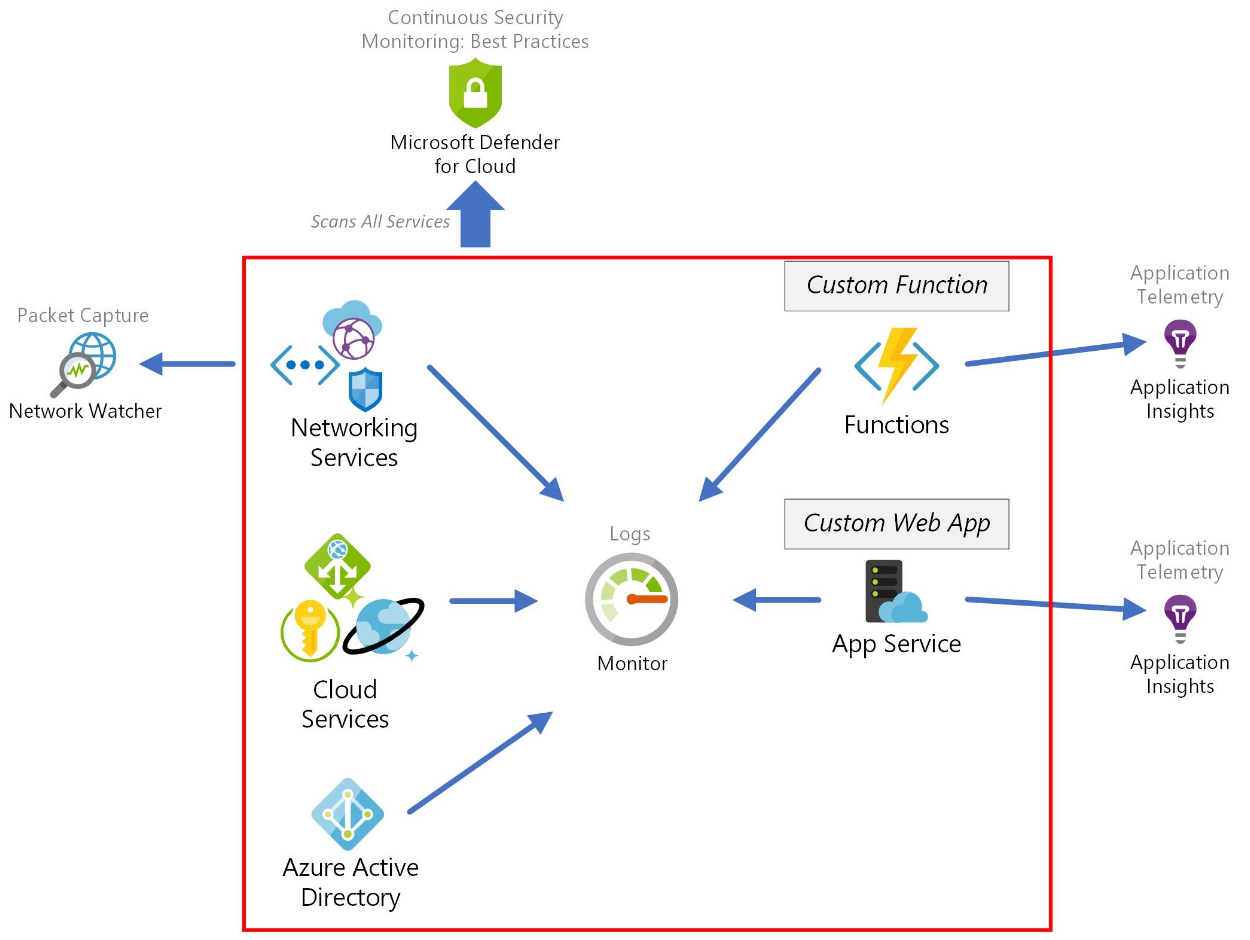


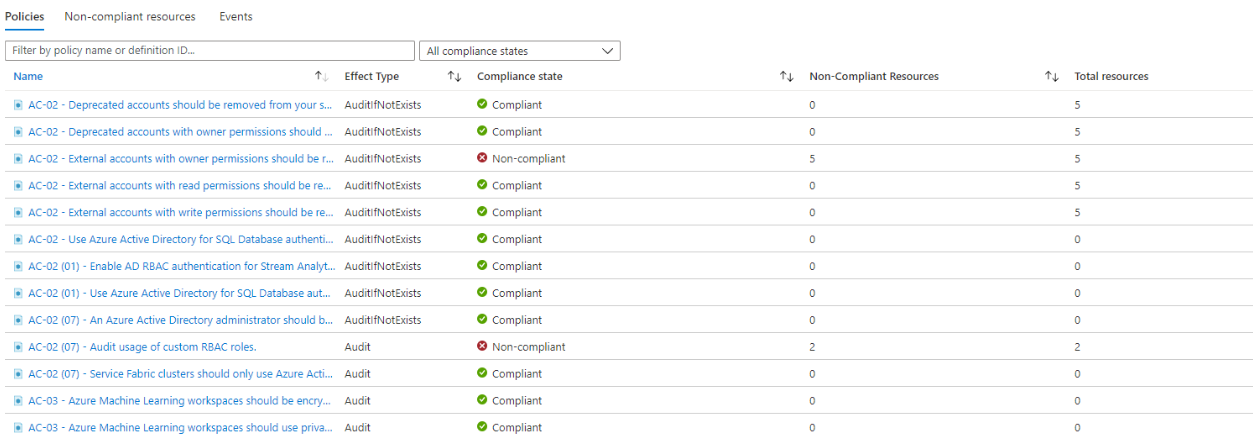
Figure 20: Microsoft Defender for Cloud Design

6.4.2 Guidance for Microsoft Defender for Cloud

- For comprehensive threat detection, "Enable all Microsoft Defender for Cloud plans" should be turned on for each Subscription. There is a small additional cost for the extended coverage.

6.5 Azure Policies

Policies allow you to create custom configuration checking rules. These rules run continuously therefore preventing or detecting cloud service configurations that expose your environment to attack. For example, Policies can check if a Storage Account has public access instead of private. Policies have a built-in dashboard that shows the status of each rule. This is depicted in *Figure 21*.


Policies Non-compliant resources Events

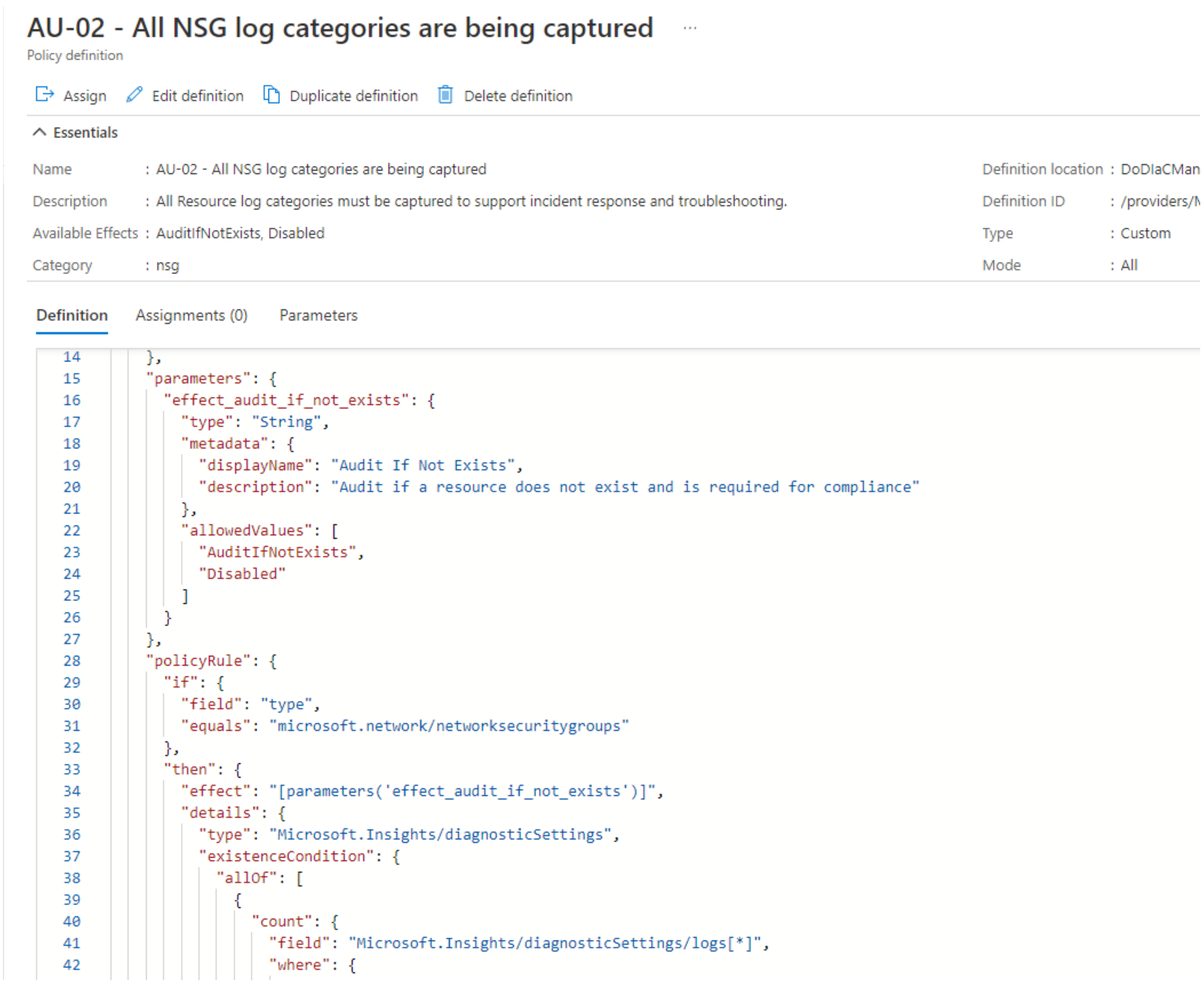
Filter by policy name or definition ID... | All compliance states

Name	Effect Type	Compliance state	Non-Compliant Resources	Total resources
AC-02 - Deprecated accounts should be removed from your s...	AuditIfNotExists	✓ Compliant	0	5
AC-02 - Deprecated accounts with owner permissions should ...	AuditIfNotExists	✓ Compliant	0	5
AC-02 - External accounts with owner permissions should be r...	AuditIfNotExists	✗ Non-compliant	5	5
AC-02 - External accounts with read permissions should be re...	AuditIfNotExists	✓ Compliant	0	5
AC-02 - External accounts with write permissions should be re...	AuditIfNotExists	✓ Compliant	0	5
AC-02 - Use Azure Active Directory for SQL Database authenti...	AuditIfNotExists	✓ Compliant	0	0
AC-02 (01) - Enable AD RBAC authentication for Stream Analyt...	AuditIfNotExists	✓ Compliant	0	0
AC-02 (01) - Use Azure Active Directory for SQL Database aut...	AuditIfNotExists	✓ Compliant	0	0
AC-02 (07) - An Azure Active Directory administrator should b...	AuditIfNotExists	✓ Compliant	0	0
AC-02 (07) - Audit usage of custom RBAC roles.	Audit	✗ Non-compliant	2	2
AC-02 (07) - Service Fabric clusters should only use Azure Acti...	Audit	✓ Compliant	0	0
AC-03 - Azure Machine Learning workspaces should be encry...	Audit	✓ Compliant	0	0
AC-03 - Azure Machine Learning workspaces should use priva...	Audit	✓ Compliant	0	0

Figure 21: DoD Cloud IaC Custom Policies

6.5.1 Implementing Azure Policies

Policies support *security in operations* since they automate what would otherwise be a manually intensive effort of verifying the secure configuration of hundreds of cloud services and resources. They should be written to fully cover the security-sensitive configurations of every cloud service in your environment. The Policy rules require custom development as part of your automated security scanning and monitoring approach. *Figure 22* shows a snippet of custom Policy code as an example.


AU-02 - All NSG log categories are being captured

Policy definition

Name	: AU-02 - All NSG log categories are being captured	Definition location	: DoDIaCMan
Description	: All Resource log categories must be captured to support incident response and troubleshooting.	Definition ID	: /providers/M
Available Effects	: AuditIfNotExists, Disabled	Type	: Custom
Category	: nsg	Mode	: All

```
},
"parameters": {
  "effect_audit_if_not_exists": {
    "type": "String",
    "metadata": {
      "displayName": "Audit If Not Exists",
      "description": "Audit if a resource does not exist and is required for compliance"
    },
    "allowedValues": [
      "AuditIfNotExists",
      "Disabled"
    ]
  }
},
"policyRule": {
  "if": {
    "field": "type",
    "equals": "microsoft.network/networksecuritygroups"
  },
  "then": {
    "effect": "[parameters('effect_audit_if_not_exists')]",
    "details": {
      "type": "Microsoft.Insights/diagnosticSettings",
      "existenceCondition": {
        "allOf": [
          {
            "count": {
              "field": "Microsoft.Insights/diagnosticSettings/logs[*]",
              "where": {
```

Figure 22: Azure Policy Code

6.5.2 Guidance for Azure Policies

- Create custom Policies to monitor security-critical configurations of every service in your tenant.
- Favor deny Policies over detection and remediation Policies. It is better to prevent a vulnerability than to correct one. For example, prevent a Storage Account from being created with public access.
- Deploy Policies at the highest level possible, usually the Management Group level. The higher the level, depending on your Subscription and Management structure, the more services you can protect.
- All environments (development, test, production, etc.) must be covered by Policies in addition to your production environment. Data is often exfiltrated from a misconfigured development environment.

- In the Azure Policy dashboard, Policies are listed by the names. To assist with auditing establish a naming convention. DoD Cloud IaC, for instance, puts the NIST-800-53 control ID at the start of the Policy name.

6.6 Microsoft Sentinel

Sentinel is a cloud native incident management and Security Information and Event Management (SIEM) service. It can identify potentially malicious actions in your tenant automatically and generate incidents and alerts. It can be used for threat hunting and automated incident response. Sentinel gets its data from a Log Analytics Workspace. Having a single Workspace, per the guidance above, facilitates Sentinel's ability to detect threats and simplifies the manual threat hunting process because all the logs are in one place.

6.6.1 Implementing Microsoft Sentinel

Sentinel is a built-in capability of Azure but must be enabled. *Figure 23* shows how Sentinel fits into the overall design.

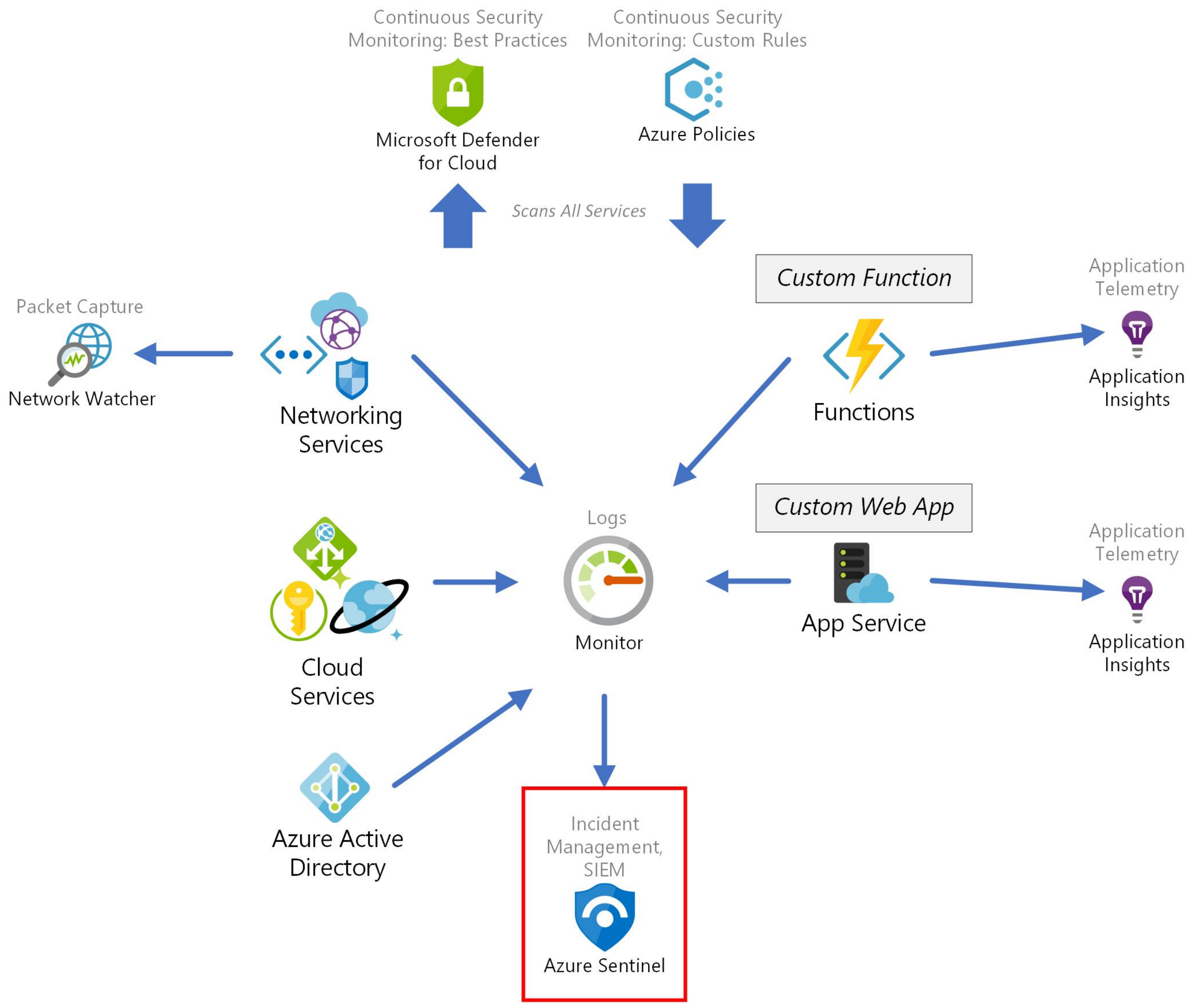


Figure 23: Sentinel Design

6.6.2 Guidance for Microsoft Sentinel

- Enable Sentinel on the centralized Log Analytics Workspace.
- Create tailored and customized Sentinel Workbooks to standardize and simplify incident detection.
- Advanced users should create automated responses to expected threats. Compromise can take place in minutes so using automation to react is the most effective way to mitigate it.
- Enable SMS and email notifications for Sentinel alerts.

6.7 Logging and Continuous Monitoring in DoD Cloud IaC

The design described above is supported by DoD Cloud IaC out-of-the-box. After DoD Cloud IaC is deployed in a new Azure Government environment the following will be enabled:

- A centralized logging capability is created in an operations Subscription.
- All cloud services have Diagnostic enabled and send their logs to centralized logging.
- Azure Sentinel is enabled for the centralized logging workspace.
- Azure Active Directory logs are sent to centralized logging.
- Microsoft Defender for Cloud–full-service coverage–is enabled for all Subscriptions.
- A large set of Azure Policies mapped to NIST 800-53 is deployed.
- Network Watcher is enabled for each Region used.

Starting with this baseline, developers can connect their applications to the logging and monitoring architecture by using the Azure ARM Templates that are also provided as part of DoD Cloud IaC. These Templates provide secure configurations of about 50 Azure services. Connecting applications to the logging architecture would involve:

- Enabling Network Security Group Flow Logs and linking them with Network Watcher.
- Point application-level cloud service Diagnostics to central logging.
- Create an Application Insights Resource for each application-level service (e.g., Functions).

7 Application Development with Azure-Managed Services

This section provides recommendations for implementing design pattern 3–Develop applications using CSP native services and policies to secure the services. Specifically, it focuses on maximizing the use of CSP native services through a serverless approach. Developing applications using the serverless model allows you to take full advantage of the security, scalability, and resiliency features of the Azure cloud. Serverless applications are built by wiring together various cloud services, each of which provides a needed function. To do this you have to know what features the Azure services provide and what use case for which it is most suitable.

This chapter will illustrate the serverless application design process though a common use case: a web application backed by an independently accessible REST API. As shown in *Figure 24*, the REST API is used by external clients and by the web application. The guidance for this use case can be re-used and adapted to different needs. A heavy emphasis will be placed on security, scalability, and resiliency. The security guidance follows the Azure Security Benchmark[34].

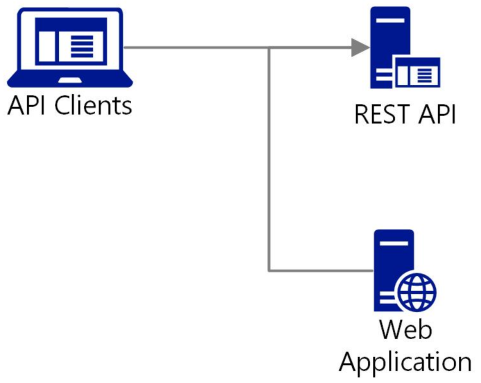


Figure 24: Example Application Overview

7.1 Automated Cloud Service Deployment

Automating the deployment and configuration of the Azure services is done through Azure Resource Manager (ARM) Templates with Azure PowerShell scripting as orchestration code. Deployment is done using pipeline runners (See Section 5.5.1 for related information.) The basic process is:

1. Create ARM templates and associated parameter files.
2. Create a PowerShell (or Azure CLI) script to deploy them.
3. Add these to your version control system.
4. Call the orchestration script within your CI/CD pipeline using the runners.

[34] https://docs.microsoft.com/en-us/security/benchmark/azure/

7.1.1 Implementing Cloud Service Automation

> *All cloud service configuration and deployment should be automated.*

Manual deployment of cloud services is highly discouraged because such deployments are not repeatable and are prone to errors. One exception is exploratory manual deployments. When working with an unfamiliar Azure service, for example, it is helpful to manually deploy it and explore its configuration options. That configuration can be exported as an ARM Template and serve as a baseline for subsequent automation.

7.1.2 Guidance for Cloud Service Automation

- Create small, single-purpose ARM templates rather than large, multi-purpose ones.
- Focus each template on one service and its supporting services. For example, one for Application Gateway that includes a KeyVault that it needs to store certificates.
- Create a template parameter file for each deployment environment, for example, one for development and one for production.
- Orchestrate the deployment of each template using a script (*e.g.*, PowerShell or the Azure CLI).

7.2 Resource Groups

The first design decision in Azure is identifying how many Resource Groups an application will need and what services will go in them. For this use case two Resource Groups are recommended: one for the web application and one for the API. Capabilities that are distinct, even if related, are better placed in their own Resource group and managed separately. This allows them to be developed and maintained separately and, per the zero-trust model, to be secured separately. Each capability is accessed via its public interface.

The Resource Group division is independent of the Azure Subscription. If, for example, you have development, test, and production Subscriptions then each will have the same two Resource Groups. *Figure 25* shows a simplified overview of the Resource Group pattern. The Subscriptions are shown to emphasize the point that development and test environments should mirror production.

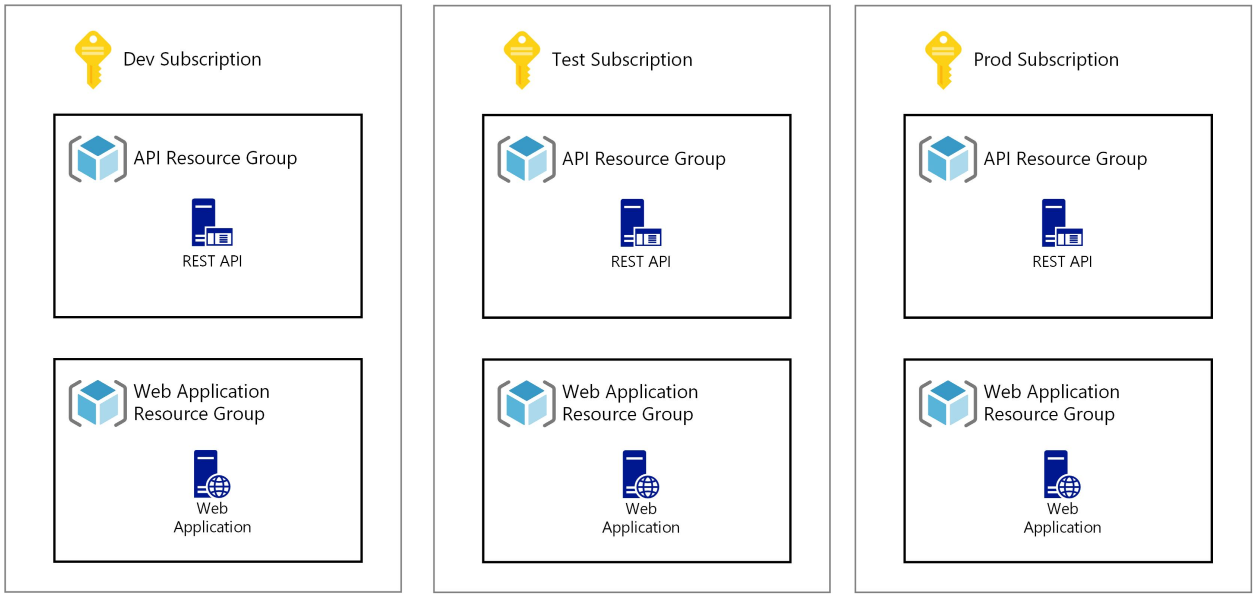


Figure 25: Basic Resource Group Design

7.2.1 Implementing Application Resource Groups

> *Segregate applications by Resource Group.*

The reasons for segregating applications by Resource Groups are that a) they provide an access-control boundary, and b) this re-enforces the micro-segmentation pattern. The access-control boundary means that only those who are working on the application will have access to it, i.e. least-privilege. In the cloud one of the ways micro-segmentation is implemented is by creating dedicated services and resources rather than sharing them. For example, creating a Storage Account for each Azure Function rather than one Storage Account that is shared by multiple Functions. When applications are deployed into separate Resource Groups, sharing services that should not be shared is discouraged.

7.2.2 Guidance for Application Resource Groups

- Create one Resource Group for the web application and one for the API per Subscription environment.
- For each Resource Group add a lock (Microsoft.Authorization/locks) with the level *CanNotDelete*.

7.3 Common Services

APIs and user-facing web applications have common design patterns and require nearly identical supporting cloud services. The three most important common design patterns are:

1. Virtual network design
2. Load-Balancing and application defense
3. Regional fail-over and content caching

7.3.1 Virtual Network Design

Few Azure PaaS services require a virtual network (VNet) to function. However, using Private Endpoints as recommended by the Azure Security Benchmark require VNets, so they play a prominent role in this guidance.

7.3.1.1 Virtual Network Implementation

> *Every service that supports Private Endpoint should have them enabled.*

Private Endpoints improve security in several ways. To explain, we must first understand endpoints in Azure. *Azure endpoints are completely different from physical endpoints in on-premises networking.* Azure services, in general, have multiple endpoints. We can categorize these endpoints as: inbound versus outbound, public (i.e., Internet facing) versus private, and data plane versus management plane. By default, for example with a Storage Account, a service will have: an endpoint that is public, inbound and data plane, 2) multiple public, outbound, data planes endpoints.

The goal of Private Endpoints is to restrict inbound, data plane traffic. A Private Endpoint creates an inbound, data plane endpoint within a VNet. When the public inbound endpoint is blocked, then all inbound data traffic can be limited to known sources. We want to limit inbound traffic like this for two reasons: 1) in most cases the inbound traffic to a cloud service is intra-service traffic only (for example, an Azure Function communicating with Cosmos DB), and 2) this supports hybrid models where the service user resides in an on-premises network.

Outbound, data plane endpoints are more challenging to restrict since we have little control over them. To restrict them we must add an additional construct: a hub VNet with Azure Firewall Premium. Azure Firewall Premium can filter outbound traffic by IP address and by domain name. We can force traffic from the application (spoke) VNet to the hub VNet by peering them and then adding a RouteTable resource that sets Azure Firewall Premium as the default gateway. With this construct in place, then you can add Firewall rules to restrict outbound traffic as appropriate.

Management plane traffic is likewise difficult to fully restrict. For this type of endpoint, we rely on Azure's built-in management back-plane and access control features, especially RBAC.

Private Endpoints reside in a subnet. Subnets that hold Private Endpoints have special constraints, for example, they do not allow NSGs. Therefore, it is recommended that you create a subnet specifically for Private Endpoints. Usually, one subnet dedicated to Private Endpoints is sufficient to provide the security benefits. You can, of course, create more than one but the marginal security gains are quickly outweighed by increased complexity and administrative overhead.

When adding Private Endpoints, it is highly recommended that you also create matching Private DNS Zones. When you add the Private Endpoint to a Private DNS Zone it creates an alias for the service FQDN in local (i.e., in your tenant) DNS that overrides the public IP with the private one.

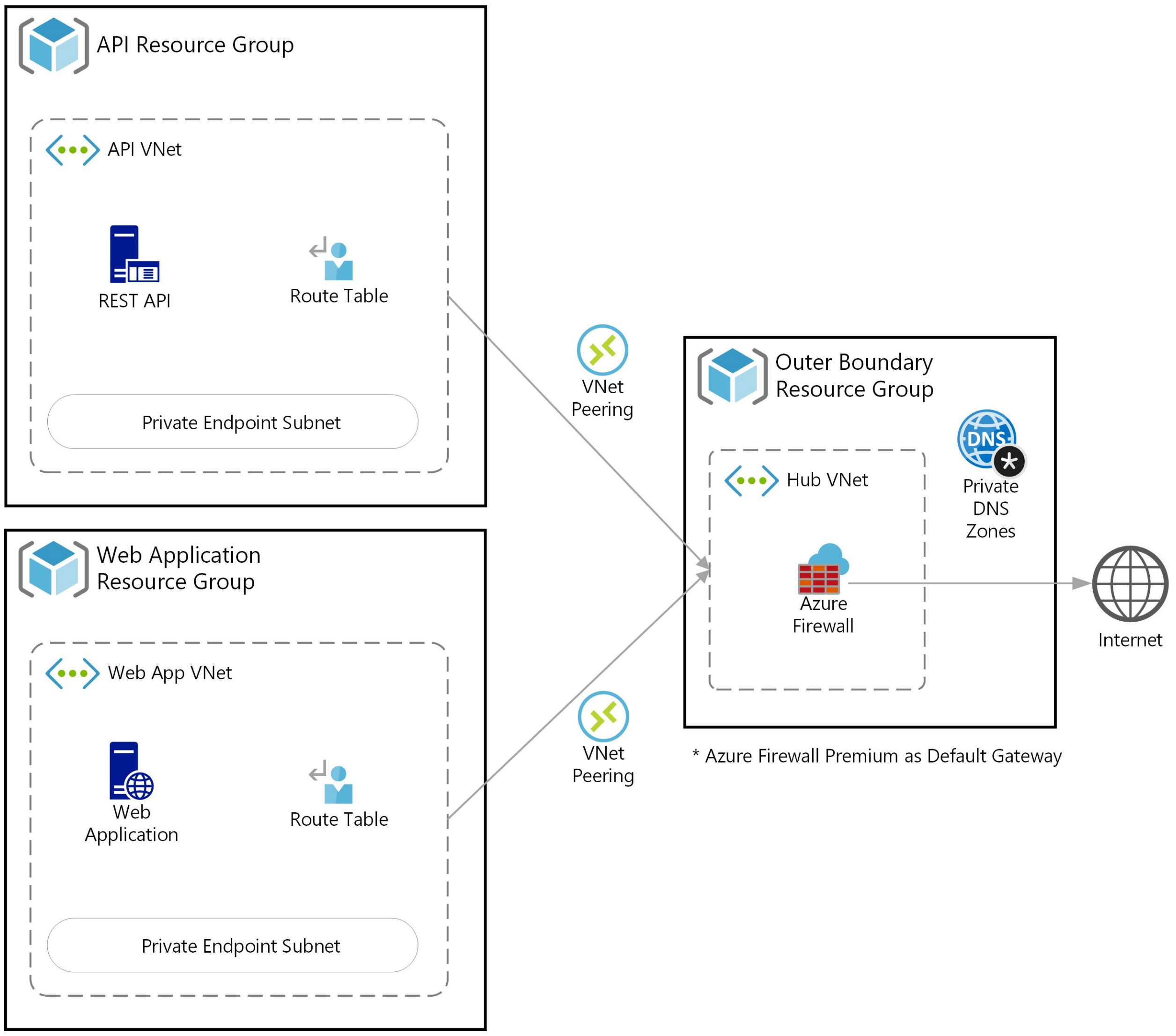


Figure 26: Virtual Network Design

Figure 26 shows an overview of the VNet design. The API and web application are abstracted and will be replaced with Azure services in the next few sections. Each VNet has a subnet dedicated to Private Endpoints. A RouteTable resource enables Azure Firewall Premium as the default gateway. Azure Firewall is a shared service so both the API and the web application use it. Network traffic is routed to Azure Firewall through VNet Peering. Azure Firewall can filter inbound and outbound traffic depending on your use case. In this example it filters outbound traffic. Inbound traffic will be handled by other services.

Private Endpoints should be registered in tenant level DNS. This is a strictly local DNS that maps the private IP to the cloud service's hostname. Azure Private DNS Zones provide this functionality. These private zones should be centralized in the Resource Group called "Outer Boundary" by DoD Cloud IaC. One zone is needed for each type of service. Storage Accounts need a zone for each child service as well.

7.3.1.2 Virtual Network Guidance

- Include a VNet with your application.
- Use Private Endpoint for every service that supports them.
- Create a subnet dedicated to Private Endpoints.
- Use the service level firewalls to block the inbound, public endpoints.
- Peer the application VNet with the Azure Firewall VNet.
- Add a RouteTable to all the VNet subnets that sets Azure Firewall as the default gateway.
- Create outbound filtering rules in Azure Firewall Premium (both network and application).
- Use non-overlapping private IP space in your environment. Peering will not work if the private IP space overlaps.
- Configure the VNet to proxy DNS requests through Azure Firewall.

7.3.2 Load-balancing and Application Defense

Applications should always be protected and never exposed directly to a client no matter where that client resides–even if it is the same tenant. Azure Application Gateway is the cloud service intended to fill this role. It provides load-balancing and a Web Application Firewall (WAF). The API and web application should be protected by their own Application Gateway.

7.3.2.1 Application Gateway Implementation

> *Every application with a web-based public interface should be behind its own Application Gateway with WAF enabled and with WAF rules specific to that application.*

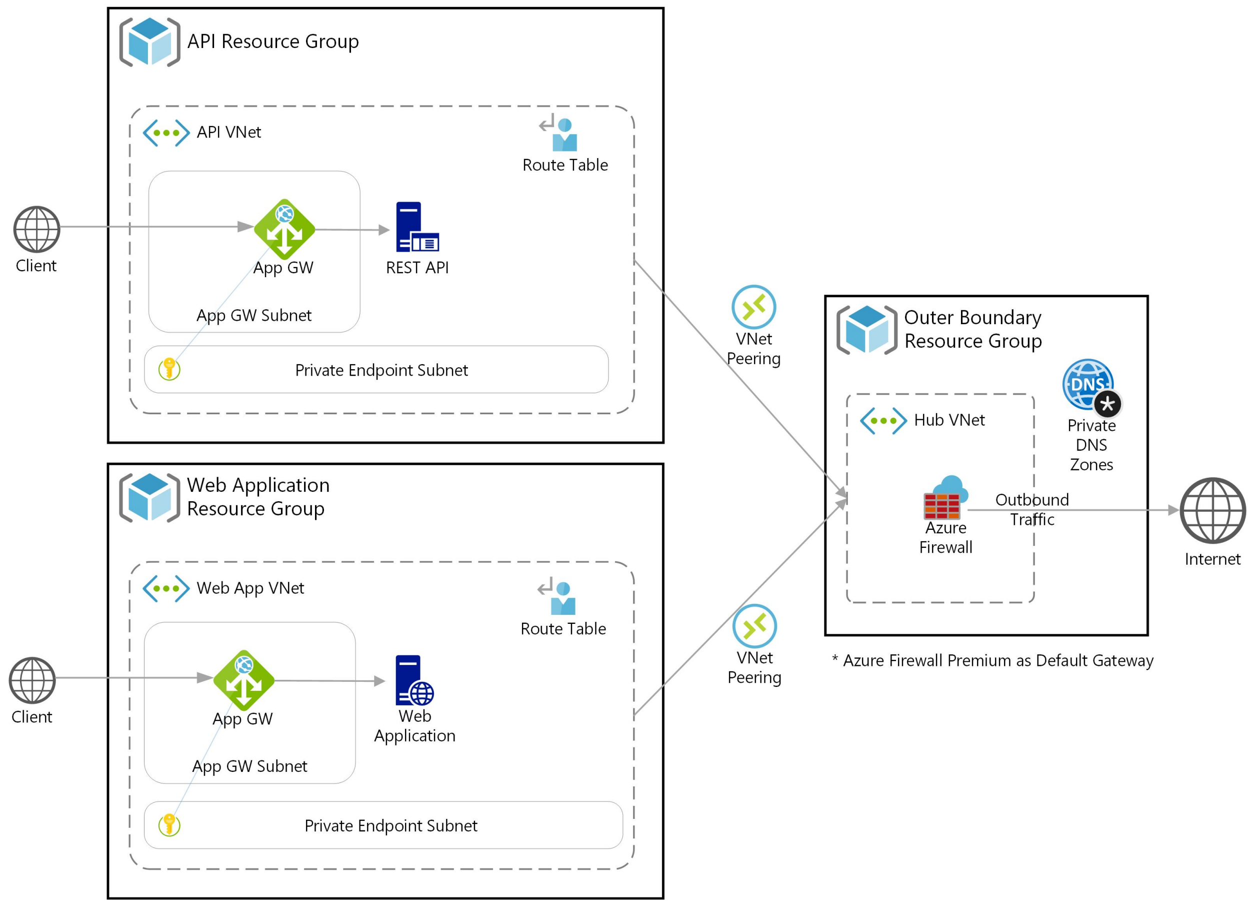


Figure 27: Application Gateway Design

In *Figure 27*, the API and web application have been protected by adding Application Gateway and WAF. Each has its own Application Gateway. They do not share one. In the zero-trust model, a model that assumes a breach, we isolate and segment as much as possible even at the extra cost of additional Application Gateways. Application Gateways provide the public endpoint. All inbound traffic for the API and web application goes through them. The network path is constrained by the other services such as the NSGs to only allow traffic from Application Gateway. No other inbound path is allowed.

The inbound traffic destined for Application Gateway is filtered in two ways. First, an NSG (not shown in the diagram) restricts it at the ports and protocols level by allowing only HTTPS (TCP/IP port 443) traffic. WAF provides the second filter. WAF supports the Open Web Application Security Project (OWASP) Core Rule Set natively. It need only be enabled. The OWASP rules block known attack signatures such as well-known vulnerabilities. WAF also supports custom rules.

Every web application and API should be protected by custom rules. These are rules that are customized specifically for the calls the web application or API expects. For example, if your API only supports a GET request of the form `/api/cars/1234` then the custom rules should only allow GET traffic matching that signature–no POST or DELETE actions, or any other URI pattern. With APIs the OpenAPI specification can be used as a guide to develop the custom rules. Web application custom rules should follow the same approach but will often take some experimentation to fully identify them. Custom WAF rules implement a zero-trust “default deny” design.

Figure 27 also shows a KeyVault in the private endpoint subnet. This illustrates placing a Private Endpoint for the KeyVault in the subnet dedicated to this purpose and then having the Application Gateway communicate with it. Technically, only the Private Endpoint attached to the KeyVault resides in the subnet, so this is a visual simplification. This pattern–an Azure service communicates with a supporting service through its Private Endpoint–is common and will be repeated throughout this guide.

When the API and web application are separated as shown in the diagram, the web application can only make calls to the API through the API’s Application Gateway. There is no back-channel that allows the web application to by-pass the first layer of security. The API does not “trust” the web application (per the zero-trust model) but treats it in the same way as it would any external client.

7.3.2.2 Application Gateway Guidance

- Create one Application Gateway per API (or related set of APIs) and per web application.
- Enable WAF on each Application Gateway.
- Enable the OWASP Core Rule Set in each WAF.
- Create custom rules *specifically tailored for your API or web application* to implement default deny and whitelisting.
- Enable Private Endpoint for supporting services such as KeyVault.
- Associate an NSG with the application gateway subnet and configure it to only allow inbound TCP/443 traffic.
- Restrict the inbound network path such that all traffic must go through Application Gateway to get to your backend.
- Add outbound Azure Firewall rules to limit traffic to from the Application Gateway to known Azure addresses.

7.3.3 Regional fail-over and content caching

Applications that require high availability must be tolerant of regional failures. If there is a regional outage, the application must be able to continue. To support this use case Azure provides a global service called Front Door. Front Door is unique because it does not reside in a single availability zone or region. Because it is global it can route traffic between regions and can tolerate failure in one of them.

7.3.3.1 Azure Front Door Implementation

> *Mission critical applications should support regional fail-over using Azure Front Door.*

Mission critical applications should be designed from the ground-up to be fault tolerant. The fault that this section is focused on is a regional service outage. Applications have different requirements and often complex workflows. Many of the Azure cloud services, such as Event Hubs and Cosmos DB, support multi-region replication. No single pattern will fit all workflows. The general advice is simply to plan for a service in each region going down and ensure that your design can handle this case.

The one pattern that is common, however, is to have a fault-tolerant public endpoint for your APIs and web applications. Azure Front Door will do this. It also aids performance by providing a global content delivery network (CDN). The CDN caches and serves content as close to the use as possible to reduce response times.

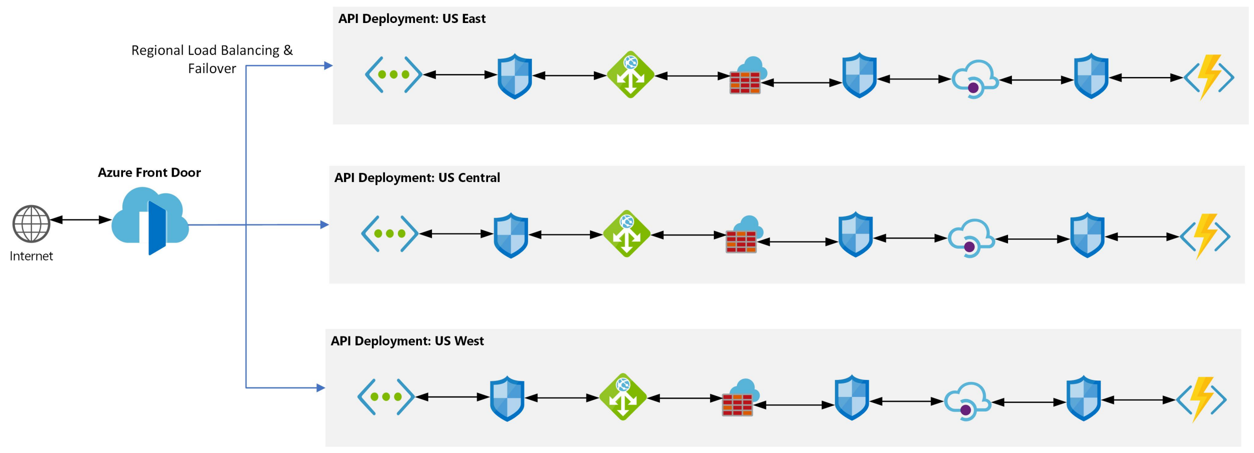


Figure 28: Front Door Design

Figure 28 shows a simple overview of the Front Door design pattern. It routes traffic from a single public endpoint to multiple regions. It sits in front of the Application Gateway which is still needed. Front Door plus Application Gateway provides fault-tolerance from the regional to the availability zone level. A separate Front Door resource is required for the API and the web application.

7.3.3.2 Azure Front Door Guidance

- Place Front Door in front of Application Gateway to provide regional fail-over.
- Design your API and web application to make effective use of CDN via the appropriate HTTP headers.

7.4 Data Storage

Most applications require a datastore. The most used PaaS datastores are: CosmosDB, Azure SQL Database, and Azure Cache for Redis. Redis is an in-memory cache that is used by API Management (APIM) and will be discussed in more detail in the APIM section.

7.4.1 Data Storage Implementation

> *Data stores should be protected via a combination of network and access security.*

As with the Azure cloud services previously discussed, the data services should be protected using Private Endpoints and have their Internet accessible, public endpoints blocked. *Figure 29* shows an example with Cosmos DB. The services that will store data in Cosmos DB or query it, connect to it via the Private Endpoint. Cosmos DB is show residing in the private endpoint subnet to illustrate the concept.

Access control should be done with Azure AD RBAC instead of keys or passwords. This enables fine-grained access control for data objects below the services level. For example, an Azure Function can have a Managed Identity with read-only permission to a Cosmos DB Container using RBAC.

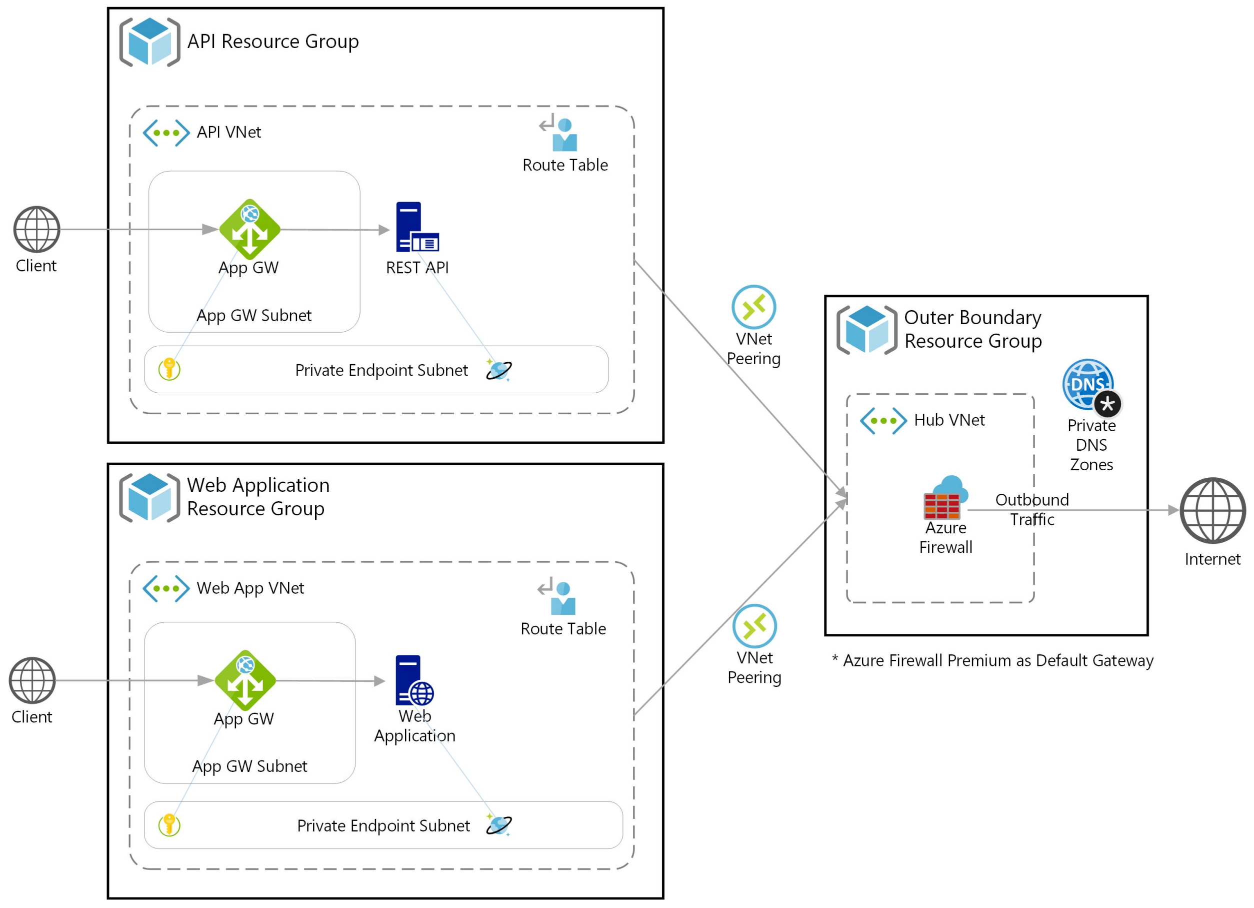


Figure 29: Data Store Design

7.4.2 Data Storage Guidance

- Add a Private Endpoint for each data service.
- Use RBAC to access the data services. Grant the RBAC role the fewest privileges it needs to perform its work, for example do not grant write access if it is not needed.
- Avoid using shared keys and passwords.
- For intra-service communication create Managed Identities that will assume the RBAC role.
- Enable Advanced Threat Protection on the data services.

7.5 API Management

This section applies to the API. API Management (APIM) should be used to wrap one or more API according to the facade pattern. It provides API clients with a single, managed interface even when there are multiple

API implementations in the backend. APIs can be bundled in and offered to clients as separate Products. This gives the API developer control over which APIs are exposed and to what clients.

APIM supports API versioning to maintain backwards compatibility. For example, you might start with API V1 and then add breaking changes to API V2 while keeping version 1 as-is.

APIM has built-in support for API call caching. A GET request, for instance, can be cached for a designated time to improve performance. Cached data calls are usually several times faster than calls to the backend. We recommend using Azure Cache for Redis for this purpose rather than APIM's native caching. Redis provides more flexibility and can cache more data.

APIM provides advanced capabilities through custom logic described in the form of APIM Policies. These Policies, not to be confused with Azure Policies, are written a template language as an XML document. They are needed to implement caching, API throttling, and various security checks. They can also be used to rewrite HTTP headers and the API call itself.

7.5.1 APIM Implementation

APIs should always be placed behind APIM. This gives the API developer full control over what APIs are exposed and to whom. It separates client from the backend implementation, helps perfromance, and adds many security controls.

As shown in *Figure 30*, APIM should be deployed into its own subnet. APIM is one of the services that can be deployed in VNet integration mode where it occupies its own subnet. When deployed into its own subnet it does not use a Private Endpoint, the VNet integration gives APIM a private IP address by default. Redis, however, should be deployed with a Private Endpoint. Note that Azure Firewall is omitted from this diagram for simplicity but is still part of the design.

The inbound traffic first goes to Application Gateway, then to APIM, and then to the backend. Not shown but also possible, the traffic can go through Azure Firewall before Application Gateway or go through Azure Firewall in between Application Gateway and APIM. It is highly recommended that Azure Firewall be in the inbound path and always in the outbound path.

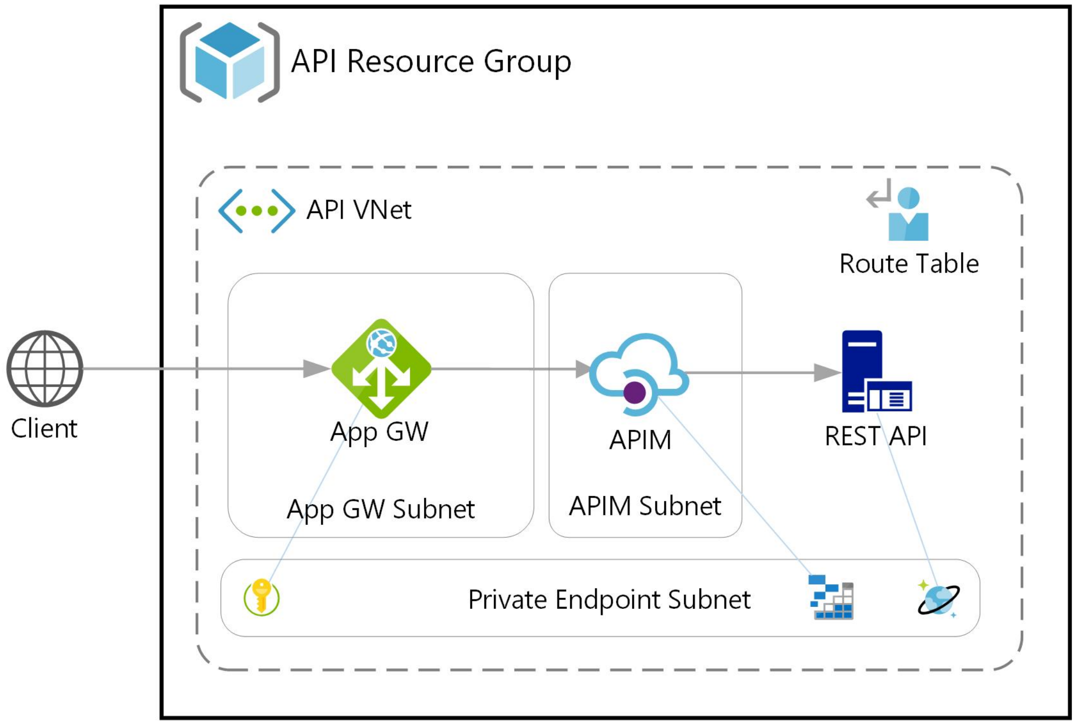


Figure 30: APIM Design

7.5.2 APIM Guidance

- Always use APIM in front of APIs.
- Deploy APIM in “Internal” mode in its own subnet.
- Deploy an NSG for the APIM subnet that restricts inbound traffic.
- Use Azure Cache for Redis for API caching.
- Deploy Redis with a Private Endpoint.
- Configure the Redis firewall to only allow traffic from the APIM subnet.
- Use APIM versioning to manage the evolution of your APIs.
- Bundle your APIs into Products according to the needs of your user base.
- Enable API throttling via APIM Policies.
- Consider also using APIM Policies to validate API calls.
- Use OpenAPI specifications to define your REST APIs. APIM can ingest these natively.
- Create an Application Insights resource specifically for APIM.

7.6 Application Hosting

Azure provides a variety of PaaS services to host application code. The ones you will most often use are:

- Azure App Service: Hosts web applications. Removes the need to deploy, configure, secure, and maintain a web server. This is the recommended service for hosting dynamic web applications.
- Azure Functions: Azure's "serverless" service. This is the recommended services for hosting stateless APIs. Functions are also recommended for building complex, highly parallel workflows. Many Azure cloud services can be wired together using Functions because, in addition to being triggered by HTTP requests, they can be triggered by messages, database change, or on a timer.
- Azure Logic Apps: Provide a low-code/no-code serverless platform with hundreds of pre-built components. Logic Apps are a potential host of simple APIs. The considerable number of pre-built components makes them a strong choice for integration with external applications and SaaS capabilities.
- Azure Service Fabric: Provides a low-level serverless hosting environment. It requires more configuration but is a good option if you have code that communicates to a system over a protocol other than HTTP.

This list is not exhaustive but should be considered as the primary options for building applications that are purely PaaS-based and avoid the need for VMs and containers.

7.6.1 Application Hosting Implementation

In this example uses case the web application will be hosted in Azure App Service and the API in Azure Functions. *Figure 31* shows the abstract icons replaced by the actual Azure cloud services. Functions and App Service are deployed in VNet Integration mode (*i.e.*, in their own subnet) and are also associated with a Private Endpoint. The purpose of this design choice is to keep the traffic with the VNet and to filter inbound and outbound traffic. The Private Endpoint is for inbound network traffic, the VNet Integration is for outbound traffic.

As covered in the section on Application Insights above, the Function and App Service should each have its own Application Insights instance. In addition to application monitoring Application Insights can also capture custom application logs.

The Functions service can host many distinct APIs with different implementations, each with different signatures. APIM can route the request to the appropriate Function code. For example, one function might implement `/car/{id}` and another implement `/garage/{id}`. APIM can present a single API to clients that supports both cars and garages. Or an API for one client with only car lookups and an API for another client with both.

Deployment of code to Functions and App Service should always be done using "slots". Slots represent deployed code and are identified by a number. A slot is activated so that that deployment becomes live. To illustrate, assume the initial version of the code is deployed into slot 1. When the code is updated, it is deployed into slot 2, and then slot 2 is made the live slot. If there is a problem with the code, the original slot 1 can be re-activated with minimal down time. Slots, therefore, provide a quick roll-back feature.

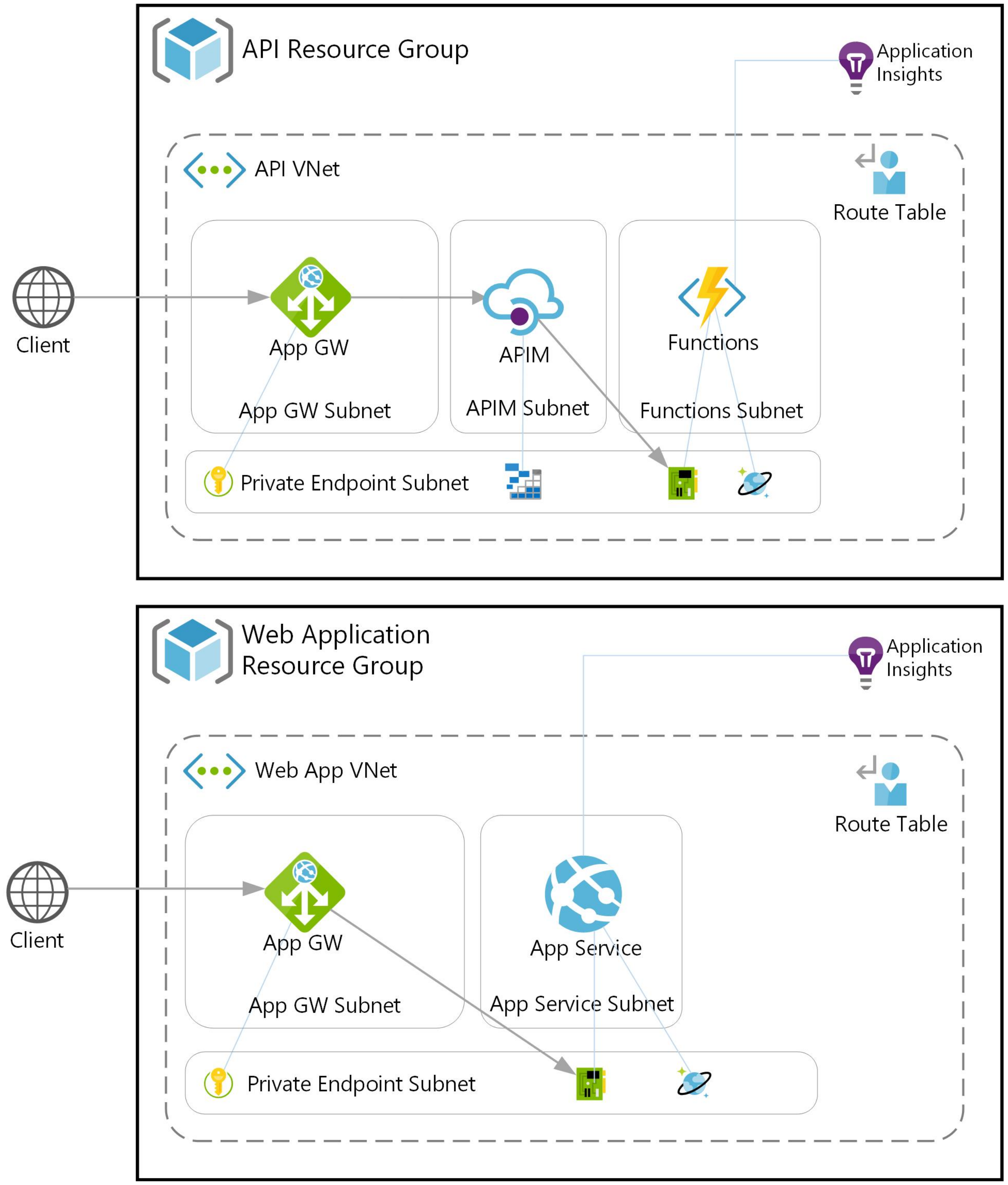


Figure 31: Application Hosting Design

7.6.2 Application Guidance

- Use Azure Functions as the primary host for REST APIs.
- Deploy more than one function in the Functions service.
- Use App Service as the primary host for dynamic web applications.
- Deploy Functions and App Service in VNet Integration mode.
- Create Private Endpoints for the Functions and App Services.
- Restrict all inbound and outbound network traffic to known paths.
- Create an Application Insights resource for each Functions and App Service.
- Use APIM to present API client with a facade. Clients should not access Functions directly nor have credentials for it.
- Always deploy to Functions or App Service slots.

8 APPENDIX: DevSecOps Lifecycle Tables

The following tables align elements of this reference design with the Lifecycle phases defined in the *DevSecOps Fundamentals Guidebook: DevSecOps Tools & Activities.* Please note that additional table-mappings will be provided in a separate file.

Table 2: Additional (non-CI/CD) GitHub capabilities aligned to DevSecOps lifecycle phases

Phase	Tool	Baseline	Capability
Plan	Issue tracking system	REQUIRED	GitHub Issues
Plan	Project management system	REQUIRED	GitHub Projects
Plan	Configuration Management Tool	REQUIRED	GitHub Repos
Develop	Integrated development environment (IDE)	REQUIRED	GitHub Codespaces
Develop	Source code repository	REQUIRED	GitHub Repos
Develop	Source code repository security plugin	PREFERRED	GitHub Advanced Security, GitHub Dependabot
Build	Artifact repository	REQUIRED	GitHub Packages
Build	Static Application Security Test (SAST) tool	REQUIRED	GitHub Advanced Security
Build	Dependency checking / Bill of Materials checking tool	PREFFERED	GitHub Dependabot

Table 3 shows the features, benefits, and inputs and outputs of the CI/CD orchestrator aligned with using GitHub Actions. GitHub Actions provide the ability to deploy code to Azure, and indeed to *any platform and cloud*.

Table 3: CI/CD Orchestrator features, benefits, inputs, and outputs, aligned with GitHub Actions.

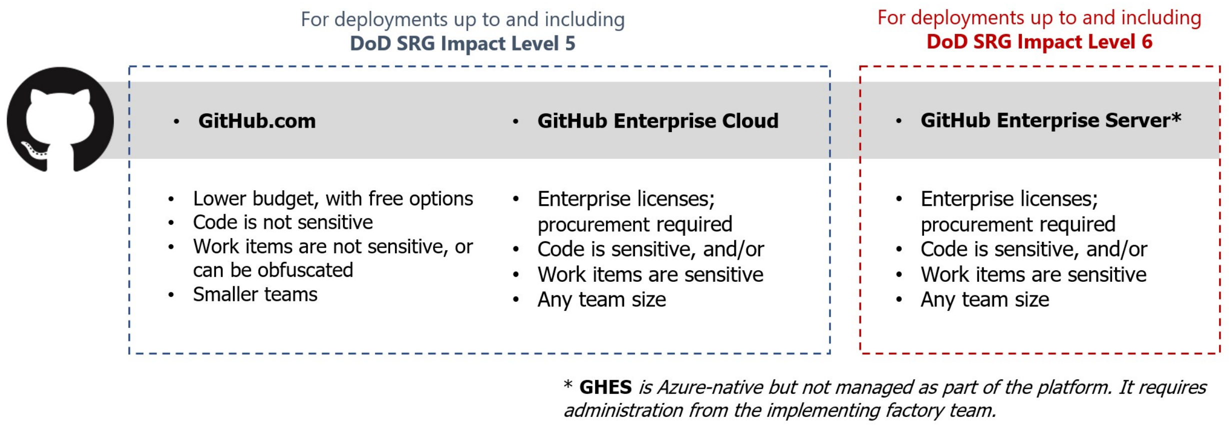
Features	Benefits	Inputs	Outputs	Baseline	GitHub
Create pipeline workflow	Customizable pipeline solution	Human input about: a set of stages; a set of event triggers; each stage entrance and exit control gates; activities in each stage	Pipeline workflow configuration	REQUIRED	**GitHub Actions**
Orchestrate pipeline workflow execution by coordinating other plugin tools or scripts.	Automate the CI/CD tasks; Auditable trail of activities	Event triggers (such as code commit, test results, human input, etc.); Artifacts from the artifact repository	Pipeline workflow execution results (such as control gate validation, stage transition, activity execution, etc.); Event and activity audit logs	REQUIRED	**GitHub Actions**

9 APPENDIX: Additional Azure-Native DevSecOps Offerings

9.1 GitHub Offerings

> *Use the simplest GitHub product that will meet your needs.*

Figure 32 illustrates the spectrum of GitHub options available with native Azure integration. For simplicity, it is shown as three broad categories, but know that it truly is a spectrum that will need to be considered given your specific requirements. Each offering is described in more detail below.



For deployments up to and including **DoD SRG Impact Level 5**

- **GitHub.com**
  - Lower budget, with free options
  - Code is not sensitive
  - Work items are not sensitive, or can be obfuscated
  - Smaller teams
- **GitHub Enterprise Cloud**
  - Enterprise licenses; procurement required
  - Code is sensitive, and/or
  - Work items are sensitive
  - Any team size

For deployments up to and including **DoD SRG Impact Level 6**

- **GitHub Enterprise Server***
  - Enterprise licenses; procurement required
  - Code is sensitive, and/or
  - Work items are sensitive
  - Any team size

* **GHES** *is Azure-native but not managed as part of the platform. It requires administration from the implementing factory team.*

Figure 32: Using guidance in this reference design, GitHub.com can be used for deployments up to and including DoD Security Requirements Guide (SRG) Impact Level 5

GitHub.com. Offers a generous free tier that will suit many teams. Hosted in the cloud, it is ideal for situations where work items and code are not sensitive in nature, although the CI/CD actions can be done in a customer-owned virtual network to achieve up to *DoD SRG Impact Level 5*. **This reference design leveraged GitHub.com and recommends this option wherever possible** given its free nature and versatility. A team can be up and running on GitHub.com with a free account within minutes. Start at https://github.com/team.

GitHub Enterprise Cloud (GHEC). For larger teams or enterprises, this managed offering provides many of the same capabilities as GitHub.com, with added support, scale, and security options (e.g., SSO). GHEC is also simple to start using, starting with a trial license. Sign up for a trial license at https://github.com/enterprise and contact sales (as the site says) in parallel. An enterprise license will need to be purchased to continue using beyond the trial window.

GitHub Enterprise Server (GHES) – *self-hosted option*. For cases where customer-hosted tooling is required or where cloud-hosted solutions are not available. GHES is functionally identical to the GHEC managed offering and provides native integration with the Azure platform. However, GHES requires administration from the factory team. Like GHEC, start with GHES by obtaining a trial license at https://github.com/enterprise. To deploy GHES, search for GitHub Enterprise Server in the Azure Portal. It is deployed as a pre-configured virtual machine image.

9.1.1 GitHub Offering Selection Guidance

- Organizations workloads up to IL5 should use GitHub.com with GitHub-hosted runners.
- Organizations with IL5 workloads can optionally use self-hosted runners.
- Organizations with workloads in AirGap clouds (IL6 and higher), should use GitHub Enterprise Server (GHES).

9.2 Azure DevOps

Organizations that have already made investments in Azure DevOps can leverage the continued ability of the tool, while planning for a migration to GitHub in the mid-term as that is the long-tem successor to Azure DevOps. GitHub Actions and Azure Pipelines have similar syntax and conceptual archtiecture, which can make migrating to GitHub Actions straightforward. GitHub provides a Migration Guide[35] for organizations embarking on this transition.

Azure DevOps supports both Azure-hosted and Self-hosted Agents[36] (runners) that can run within an Organization's VNet in Azure. Like GitHub, the communication is always initiated by the Agent, which queries Azure DevOps to determine the jobs that are required to be run and to report logs and status.

Azure DevOps uses a Service Connection to allow Pipelines to communicate with Azure. Service Connections utilize an Azure service principal to communicate with Azure, which can be scoped with the appropriate permissions to resources. The Azure DevOps guide for deploying to Azure Government[37] details the steps to create a service principal and configure a Pipeline to use it.

9.2.1 Guidance on when to use Azure DevOps

- Organizations with investments in Azure DevOps may continue to use it, although investments from Microsoft will continue to favor the long-term strategic direction of using GitHub.
- Organizations with few or no investments in DevOps are encouraged to start with GitHub.

[35] https://docs.github.com/en/actions/migrating-to-github-actions/migrating-from-azure-pipelines-to-github-actions
[36] https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents
[37] https://docs.microsoft.com/en-us/azure/azure-government/connect-with-azure-pipelines