

## CHAPTER 7

# Workload Protection – Containers

In this chapter, we discuss protecting container workloads in Azure. After reading this chapter, you will be able to select and implement appropriate controls for securing container workloads as part of your organization's security policy framework.

Containers can be hosted in multiple services in Azure, collectively referred to as container-as-a-service services. For the purposes of this book, I consider App Service for containers a platform-as-a-service service, as it includes further and opinionated web services such as scaling, load balancing, and authentication. Likewise, I consider virtual machines purely as infrastructure as a service, despite their support for nested virtualization and, thus, capability to act as a container host.

## Container Security

Container security is a broad and nuanced topic. To set the context for this book, I will briefly explain the core security concepts. If you are familiar with container security, feel free to skip ahead to the sections covering specific Azure container-as-a-service security.

## Build Security

Comprehensive container security starts with securing the container image. When a container image is built, it often uses a base image instead of defining the container image from scratch.

A base image is a parent image, typically containing at least the operating system and often purpose-built for specific workloads. For example, instead of a full-blown Ubuntu image, the more lightweight Alpine Linux could be used as a base image, reducing the attack surface. Standardizing and controlling the base image used for your container

applications is as important as standardizing and controlling so-called golden images for virtualized workloads. Furthermore, container images can build upon multiple layers of base images. This means that you can introduce opinionated base images on top of the generic Linux distributions, such as the enterprise standard base image for a Python application, which includes the necessary middleware in addition to the operating system.

After building your container image, you should scan it for vulnerabilities and remediate them before making your image available, such as pushing the image to your container registry.

Once you are satisfied with your newly built container image, your next task is to distribute it securely to your container workloads. In the cloud context, assurance of the integrity of the images is crucial. A common solution for this is to sign the images before distributing them. This allows you to limit your container hosts to run only signed images.

## Registry Security

In addition to controlling the base images and the resulting application container images, you will need to control the usage of container registries within your organization. You can either rely on a single, centrally managed, private container registry, a distributed approach where teams manage their container registries according to your cloud security framework, or combine these in a mixed model. By consolidating your container registries, you can perform and act on vulnerability scans centrally.

Containers can be hosted in multiple services in Azure, each with their own sets of security controls. Often the application teams host their container applications in multiple or even all these services. Therefore, in large enterprises, one of the trickiest operational challenges is to gather an up-to-date inventory of container images that are running in your environment. You need to solve this to be able to mitigate any security issues, such as finding of new vulnerabilities in the images.

Once you are assured of the origin and the security state of the images in your container registries, you are ready to move on to container runtime security.

## Runtime Security

The runtime security considerations can be split into the host hardening and application security.

Your host hardening responsibilities vary depending on the Azure service you are using to run your container images in. If you are hosting the container in infrastructure as a service, you are responsible for the host and should secure it using checklists such as the CIS Docker Benchmark.<sup>1</sup>

For container-as-a-service services, you need to understand the specifics of the individual services you choose to use. Azure Kubernetes Service provides a significant number of additional controls to you, but also leaves you responsible for configuring many of them. Azure Container Instance, on the other hand, takes care for patching and hardening the underlying container host, but limits the controls you have available for protecting the workload.

Your choice of container hosting service also dictates which controls are available to you for securing your container workloads. If you need to run multiple containerized applications in the same environment, Azure Kubernetes Service is a likely fit for you. By choosing Azure Kubernetes Service, however, you are responsible for a larger set of controls, such as network segmentation and enforcing the containers are executed using least privileges.

Finally, there are some application-level controls you need to apply in any service you are using to host your container workloads. These include access control to the registry, secret management, monitoring, and data-plane network control.

## Azure Container Registry

Azure Container Registry is a service for storing, distributing, scanning, and managing container images. Azure Container Registry is based on the open source Open Container Initiative (OCI) Distribution Specification and supports Docker images, Helm charts, OCI images, and OCI artifacts. Azure Container Registry also supports automating build tasks using the ACR tasks feature.

---

<sup>1</sup>[www.cisecurity.org/benchmark/docker/](http://www.cisecurity.org/benchmark/docker/)

Wherever you end up running your container images in Azure, you should use Azure Container Registry or a similar private container registry to control your container image life cycle.

## Access Control

Access to container registry is controlled using Azure Active Directory or admin keys (sometimes called master keys). **Whenever possible, you should use Azure Active Directory-based authentication.** If you are dealing with legacy workloads or third-party solutions, you might need to revert to using admin keys.

## Data-Plane Role-Based Access Control

Access to the Azure Container Registry control plane is controlled using Azure role-based access control. The default RBAC roles Owner, Contributor, and Reader allow you to manage access to, configure, and the content of your Azure Container Registry.

---

**Note** The Reader role grants access to the data plane. Specifically, it lets you pull images from the Azure Container Registry. Keep this in mind when considering the resource group placement and RBAC access of your Azure Container Registry instance.

---

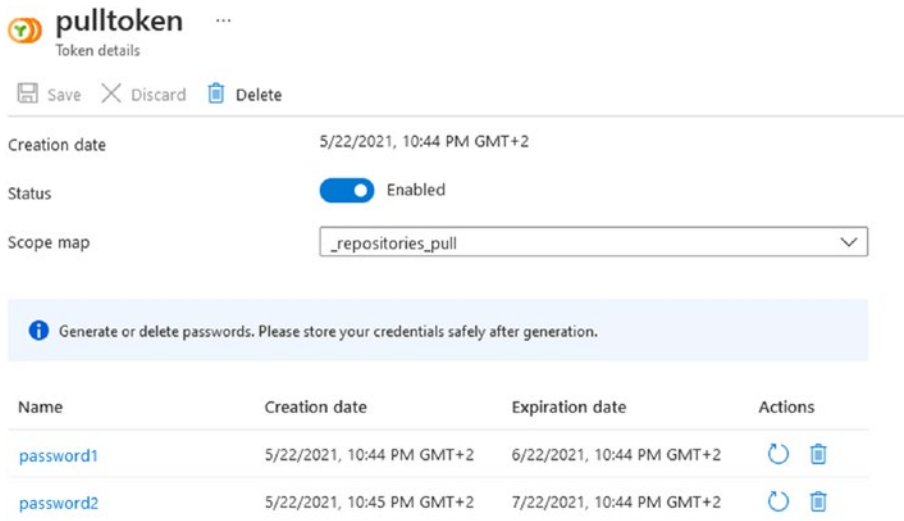
The following built-in roles complement the default roles and grant access to the data plane of Azure Container Registry:

- **AcrPush:** Grants access to push and pull images
- **AcrPull:** Grants access to pull images
- **AcrDelete:** Grants access to delete images or other artefacts
- **AcrImageSigner:** Grants access to sign images if content trust is enabled. Typically used together with the AcrPush role, as part of automated CI/CD access

## Access Control Without Azure Active Directory

In addition to Azure Active Directory authentication, Azure Container Registry supports authentication with an admin account. The admin account is similar to the shared key access of Azure Blob storage: it applies to the whole registry and provides both push and pull access. Admin user is meant for testing purposes. Using admin user is not recommended, and it is disabled by default. Regretfully, as we went to press, the portal experiences of some Azure services such as Azure Container Instance and App Service Web App for Containers are still using the admin account to control access to the Azure Container Registry.

When Azure Active Directory authentication is not viable, instead of admin account, you can control third-party access to containers in your Azure Container Registry using repository-scoped tokens, as illustrated in Figure 7-1. This method is similar to the delegated access of Azure blob storage. A token along with a generated password lets the user authenticate with the registry using `docker login` simple authentication. You can set an expiration date for a token password and revoke them when needed.



The screenshot shows the 'pulltoken' interface in the Azure portal. At the top, there's a header with the 'pulltoken' logo and 'Token details'. Below this are three buttons: 'Save', 'Discard', and 'Delete'. The main content area displays the following details:

- Creation date:** 5/22/2021, 10:44 PM GMT+2
- Status:** Enabled (indicated by a blue toggle switch)
- Scope map:** A dropdown menu showing '\_repositories\_pull'

Below these details is a light blue informational box with an 'i' icon: 'Generate or delete passwords. Please store your credentials safely after generation.'

At the bottom, there is a table listing generated passwords:

Name	Creation date	Expiration date	Actions
<a href="#">password1</a>	5/22/2021, 10:44 PM GMT+2	6/22/2021, 10:44 PM GMT+2	
<a href="#">password2</a>	5/22/2021, 10:45 PM GMT+2	7/22/2021, 10:44 PM GMT+2	

**Figure 7-1.** Azure Container Registry repository-scoped tokens

Lastly, Azure Container Registry can also be configured to allow anonymous pull access. When enabled, the feature allows anonymous pull operations to all the artefacts in your repository, so it should not be enabled in a registry you are using to manage your private container images. To prevent anonymous pull access, enforce the `anonymousPullEnabled` property to `False`.

## Network

Network access to the data-plane Azure Container Registry is unrestricted by default: the container images can be pulled from any network location. To protect network access to your Azure Container Registry, you can set multiple controls in place.

First, you can enable the registry firewall. This changes `publicNetworkAccess` from Enabled to Disabled. After enabling the firewall, you need to specify allowed network locations with `ipRules`. Additionally, you can control access to your registry using private endpoints. This minimizes exposure to public networks and helps you prevent data exfiltration.

---

**Note** Azure Container Registry supports a maximum of 100 network access rules and 10 private endpoints per instance.

---

To allow access from other Azure services without using IP rules or private endpoints, you can enable the Allow trusted services feature (configuring `networkRuleBypassOptions` to `AzureServices`). This enables access from other Azure Container Registries or Azure Machine Learning workspaces. Other registries can either import images directly or use images in your registry as a base image to build their application images. Azure Machine Learning workspaces can use your registry to deploy or train their models.

Enabling the Allow trusted services feature does not allow access from other managed Azure services, such as App Service, Azure Container Instances, or Azure Defender image scanning. To control access to those services, consider applying compensating controls.

## Logging

Azure Container Registry provides security logs for management and data planes. Management-plane, or platform, logs are created within Azure activity log. These logs include

- Administrative operations, such as deleting or restarting the registry
- Role-based access control changes (`roleAssignments/write`)

- Data-plane Access control changes, such as changes to the `adminUserEnabled` property or someone performing the `Microsoft.ContainerRegistry/registries/listCredentials/action` operation
- Network control operations, such as changes to `publicNetworkAccess`

Azure Container Registry does not keep data-plane logs by default. To enable data-plane logging, you need to configure the Azure Container Registry resource logs (under diagnostic settings) to be sent to your centralized log store. These logs include

- **ContainerRegistryLoginEvents:** Logs from registry authentication events and success status, including the identity and IP address
- **ContainerRegistryRepositoryEvents:** Logs from operations on content in registry repositories. The following operations are logged: push, pull, untag, delete (including repository delete), purge tag, and purge manifest. Includes identity and IP address information

To log network access to your Azure Container Registry, enable network security group flow logs for the subnets where the private endpoints that can access your Azure Container Registry are placed.

Additionally, you can use Azure Defender to automatically scan your images using Qualys. Once Azure Defender is enabled, Azure Defender pulls the images from your container registry and scans them in an isolated sandbox with the Qualys scanner. Images are scanned on push or import. If an image has been pulled in the last 30 days, it is also scanned weekly. The scan results are published as recommendations in Azure Security Center. To manage signal noise, you can also suppress findings by adding Disable rules.

Disable rules can be set using the following filters:

- Finding ID
- Finding category
- CVSS v3 score
- Severity
- Patchable status

**Note** Azure Defender cannot currently perform image vulnerability scanning in a registry that restricts networks access to private endpoints, virtual networks, or IP addresses.

---

## Best Practices

In this section, we discuss additional security controls and best practices.

### Encryption at Rest

While the data stored in Azure Container Registry is always encrypted with Microsoft-managed keys (MMK) using AES 256-bit encryption, your organization might have requirements to control the key length, operations, or storage. To do that, you need to specify the encryption type as customer-managed keys (CMK) and manage the encryption keys using Azure Key Vault. Using customer-managed keys, Azure Container Registry supports RSA encryption keys of sizes up to 4096 bits.

### Automate Base Image Updates

You can set up an Azure Container Registry Task to track a dependency on a base image when it builds an application image. When the updated base image is pushed to your registry, or a base image is updated in a public repository, Azure Container Registry Task can automatically build the application images based on it. If your base image is hosted in another Azure Container Registry, the Task is triggered immediately. If your base image is stored in a public repository, such as Docker Hub or Microsoft Container Registry, the Task is triggered at a random interval between 10 and 60 minutes.

### Image Signing

To enable support for signed images, Docker content trust, enable content trust on your Azure Container Registry. This sets the `TrustPolicy` status to `enabled`. Once enabled, signed images can be pushed to the container registry. When a signed image is pulled, the Docker client of the pull host verifies the integrity of the image.



## High Availability

As a managed PaaS service, Microsoft is responsible for most of the high availability implementation of Azure Container Registry. Microsoft offers 99.9% availability SLA for Azure Container Registry out of the box. To improve the resiliency of your registry, you can configure it in the zone redundancy. To improve global performance and resiliency against regional outages, you can implement geo-replication to other regions.

### EXERCISE

You are designing security controls for a high-priority business application. The application includes a containerized version of your latest machine learning models. The container images are stored in Azure Container Registry. You are required to provide an audit trail of each user that has had access to the model. How will you configure the container registry?

#### Bonus Exercise

You are required to provide network access logs of each successful and unsuccessful attempt at downloading the container image. How will you configure the container registry, and which data sources will you use for this?

---

## Azure Container Instance

Azure Container Instance is one of the container-as-a-service options for running containers in Azure. With Azure Container Instance, Microsoft takes care of hardening and operating the underlying operating system, networking, and log integration. You are left with managing the container image: application and its supporting middleware. Compared to platform as a service, Azure Container Instance provides you with fewer features and looser platform integration. For example, network features are not as advanced, and there is no built-in authentication.

Azure Container Instance applications are defined in Container Groups, which are functionally like Kubernetes Pods.

## Access Control

To create the Azure Container Instance, you need to authenticate the Azure Container Registry. As the Azure Container Instance resource does not exist at that point, it cannot be assigned a managed identity to access Azure Container Registry. To access the Azure Container Registry to pull your container image for the Azure Container Instance, you can use either service principal or repository-scoped tokens. Both approaches rely on a username and password to authenticate. The least privileged method of authenticating would be to use a token, which would be assigned the `repositories_pull` scope map. Service principal should be avoided as an authentication method for Azure Container Registry, as adversaries may add additional service principal credentials to maintain persistent access.<sup>2</sup>

## Network

To protect access to and from your Azure Container Instance application, you can place it in a virtual network and limit it to use only a private IP address. The virtual network placement lets you protect an Internet accessible application in an Azure Container Instance by deploying a Web Application Firewall in front of your application. Additionally, with user-defined routes, you can enforce all outbound application traffic to go through a firewall.

---

**Note** Azure Container Instance is unable to access a network-protected container registry to pull images.

---

If you prefer to host the network controls in a sidecar and deploy it together with your application, you can expose the Azure Container Instance with a public IP address and even configure Azure to create a fully qualified domain name for it.

Whether you use private or public IP addresses, you will need to expose your application port on the IP address from the Container Group.

---

<sup>2</sup><https://attack.mitre.org/techniques/T1098/001/>

## Logging

Azure Container Instance provides security logs for management and data planes. Management-plane, or platform, logs are created within Azure activity log. These logs include

- Administrative operations, such as deleting the Azure Container Instance resource
- Role-based access control changes (roleAssignments/write)

For data plane, container diagnostic events can be pulled with the `az container show` command. These logs include container deployment events, such as image pull, or container restart. Application-level logs from within your application can be pulled with the `az container logs` command.

To send application logs to your centralized log store, you need to implement log ingestion in the application. If you use Log Analytics Workspace, you can add your workspace credentials in the `properties.diagnostics` field of your Container Group deployment configuration. As of the writing of this book, this integration did not yet support managed identities.

## Azure Kubernetes Service

Azure Kubernetes Service is a managed Kubernetes service in Azure. It is not, however, platform as a service. Microsoft is responsible for creating, configuring, and operating the **Kubernetes control plane** of your Azure Kubernetes Service. This includes Kubernetes API servers, Etcd, kube-dns, and other system components in the `kube-system` namespace. You are still responsible for parts of your Azure Kubernetes Service, such as network controls or agent node patching. Figure 7-2 illustrates this.

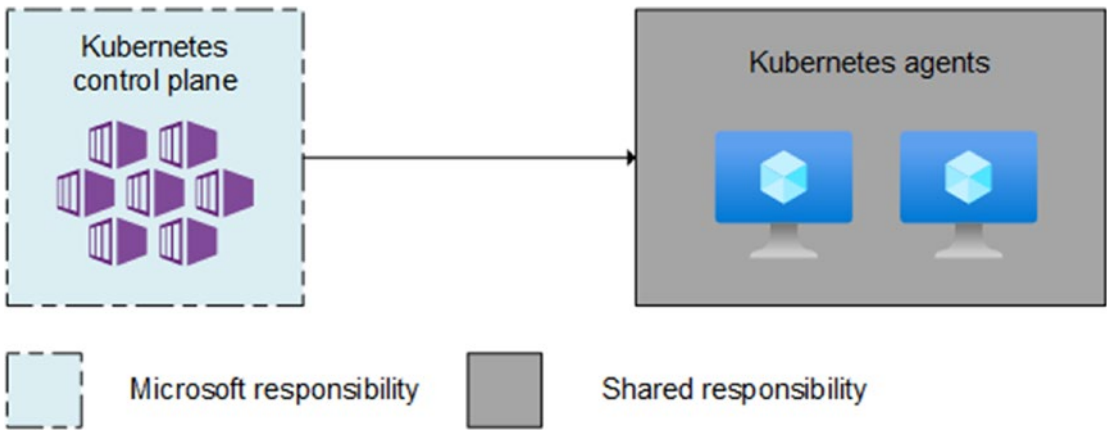


Figure 7-2. Shared responsibility of Azure Kubernetes Service

## Access Control

Administrative access to Azure Kubernetes Service can be controlled using Azure role-based access control, Azure Active Directory, and Kubernetes role-based access control. Figure 7-3 illustrates these access controls.

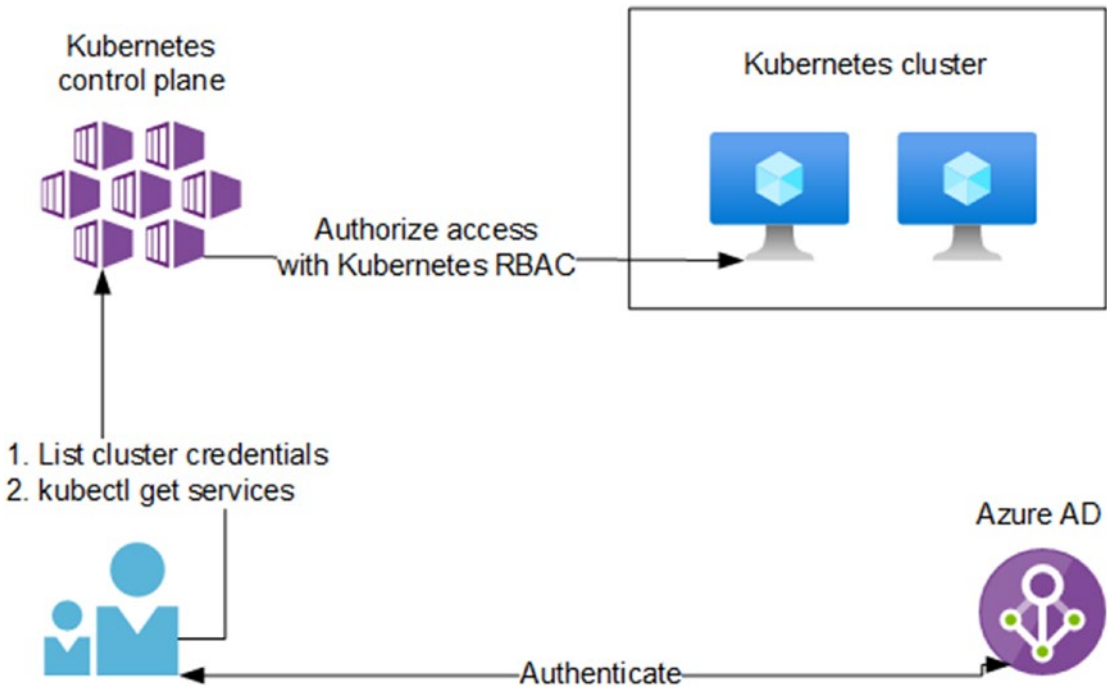


Figure 7-3. Azure Kubernetes Service access control

You can use the `kubectl` tool to authenticate the Azure Kubernetes Service clusters. To authenticate through headless scenarios, such as from a continuous deployment pipeline, you can use the `kubelogin` Kubernetes credential plugin.

## Azure RBAC

Kubernetes control plane can perform operations such as creating, updating, or deleting services in your Kubernetes cluster. Access to the Kubernetes control plane is controlled using Azure role-based access control. The built-in Azure Kubernetes Service Cluster User role grants access to list the Kubernetes **cluster user credentials** and download the kubeconfig file. The built-in Azure Kubernetes Service Cluster Admin role grants access to list the Kubernetes **cluster admin credentials** and download the kubeconfig file. You should use the Azure Kubernetes Service Cluster User role to control access to the Kubernetes control plane.

---

**Note** The admin user bypasses Azure AD sign-in to the Kubernetes control plane. The Contributor role can use the `listClusterAdminCredential` action to get the admin user credentials!

---

## Azure Active Directory Authentication and Kubernetes RBAC

After downloading the cluster user credentials using Azure RBAC roles, you can authenticate your users using Azure Active Directory and authorize them with Kubernetes RBAC.<sup>3</sup>

You can enable Azure Active Directory integration on Azure Kubernetes Service clusters with the `--enable-aad` option, either at the cluster creation or at cluster upgrade. You will also need to specify an Azure AD group that will have access to sign in to the cluster.

To further control access within the cluster, you can define Kubernetes RBAC Roles and assign them to users or groups using `RoleBindings`. The scope of a Kubernetes RBAC assignment is a namespace or the entire AKS cluster.

---

<sup>3</sup><https://kubernetes.io/docs/reference/access-authn-authz/rbac/>

To prevent admin users bypassing the Azure Active Directory authentication, you can disable local accounts by enforcing the `disableLocalAccounts` property. As of the writing of this book, the `disableLocalAccounts` property was in early preview.

## Network

You can protect the administrative access to your Kubernetes control plane by adding network controls that prevent public access. You can also add multiple network controls to protect your applications running in Azure Kubernetes Service. Let's look at these in more detail.

### Kubernetes Control Plane Network Controls

By default, your Kubernetes control plane and your Kubernetes API server and possible Kubernetes dashboard are publicly accessible. You can control network access to the Kubernetes control plane with authorized IP ranges. When you configure the `authorizedIPRanges` property, only requests made to the API server from IP address that you have explicitly listed are allowed.

To keep the traffic between the Kubernetes control plane and your cluster nodes in a private network, you can use the Private Cluster feature by enabling the `enablePrivateCluster` property. With Private Cluster, the control plane communicates with the cluster nodes through Azure Private Link. This prevents Kubernetes control plane for users outside the cluster virtual network, or without a jumpbox.

### Application Network Controls

To meet your application business goals and enterprise security requirements, you can set multiple network controls in place. These allow you to

- Control ingress and egress traffic (north-south)
- Control traffic between your cluster namespaces, nodes, or pods (east-west)

To control ingress traffic, you can implement an ingress controller. An ingress controller can be a container hosting web application proxy logic, such as `nginx`, or it can stay outside your cluster altogether. In the former case, you can configure your application load balancing, authentication, and encryption in transit within the cluster.

In the latter case, you can use Azure Kubernetes Service's integration with Azure Application Gateway. With Application Gateway, you can configure encryption in transit and Web Application Firewall using platform-as-a-service components.

To control east-west traffic, that is, traffic within your cluster and between your microservices, you can implement a service mesh such as Istio, Linkerd, or Open Service Mesh. A service mesh can control traffic flows within your cluster. As a service mesh is typically implemented as a proxy, it can also encrypt your traffic in transit.

## Logging

Azure Kubernetes Service provides security logs for management plane and Kubernetes control plane. Management-plane, or platform, logs are created within Azure activity log. These logs include

- Administrative operations, such as deleting the Azure Container Instance resource
- Role-based access control changes (roleAssignments/write)
- Data-plane access control changes, such as someone performing the `Microsoft.ContainerService/managedClusters/listClusterAdminCredential/action` operation
- Network control operations, such as changes to `authorizedIPRanges`

Azure Kubernetes Service does not keep Kubernetes control-plane logs by default. To enable Kubernetes control-plane logging, you need to configure the Azure Kubernetes Service resource logs (under diagnostic settings) to be sent to your centralized log store. These logs include

- **kube-audit** category contains all audit log data for every audit event, including get, list, create, update, delete, patch, and post.
- **kube-audit-admin** category is a subset of the kube-audit log category, excluding the get and list audit events from the log.
- **guard** category contains Azure Active Directory authentication logs.

Additionally, you can use Azure Defender to analyze data-plane logs and create Security Center alerts. For Azure Kubernetes Service, these alerts include<sup>4</sup>

- Exposed Kubernetes dashboard detected
- Privileged container detected
- Suspicious request to Kubernetes API

## Best Practices

Microsoft’s Patterns and Practices team has published a reference implementation of a baseline Azure Kubernetes Service cluster,<sup>5</sup> which is a good collection of best practices.

Microsoft applies **daily patches** (including security patches) to Azure Kubernetes Service virtual machine hosts (nodes). Some security updates, such as kernel updates, require a node reboot to finalize the process. This reboot process does not happen automatically. A Linux node that requires a reboot creates a file named `/var/run/reboot-required`. You are responsible for scheduling the reboots as needed. You can use Kured<sup>6</sup> (KUBernetes REboot Daemon) by weaveworks for this.

You are responsible to keep your Kubernetes version updated and staying within the one-year support window.<sup>7</sup> To upgrade the **Kubernetes version** of your cluster, you need to perform Azure Kubernetes Service upgrade operation, which deploys a new node with the new Kubernetes version, cordons and drains your old node, schedules your Kubernetes pods in the new node, and finally deletes your old node.

In addition to manually upgrading your Azure Kubernetes Service cluster, you can configure auto-upgrade on your cluster. To control when the cluster upgrade operations are performed, you can also configure Planned Maintenance. Planned Maintenance allows you to limit all Azure Kubernetes Service maintenance operations (including cluster upgrades) to a specific weekly time.

As of the writing of this book, auto-upgrades and Planned Maintenance features were in early preview.

---

<sup>4</sup><https://docs.microsoft.com/en-us/azure/security-center/alerts-reference#alerts-akscluster> and <https://docs.microsoft.com/en-us/azure/security-center/alerts-reference#alerts-containerhost>

<sup>5</sup><https://github.com/mspnp/aks-secure-baseline>

<sup>6</sup><https://github.com/weaveworks/kured>

<sup>7</sup><https://kubernetes.io/blog/2020/08/31/kubernetes-1-19-feature-one-year-support/>



## Summary

In this chapter, we looked at the controls available to you for protecting Azure container-as-a-service workloads.

As we have learned, their support for Azure Active Directory is consistent, but there are differences in supporting Azure-native network controls. We have also established that the use of container-as-a-service workload introduced the need for governance and architecture. Specifically, container registries and Kubernetes in-cluster networking controls require careful planning and cross-team collaboration.