StackRox

Definitive Guide to Azure Kubernetes Service (AKS) Security

Definitive Guide to Azure Kubernetes Service (AKS) Security

The accelerating pace of adoption of the Kubernetes container orchestrator shows no signs of slowing. That explosion has also fueled the creation of managed Kubernetes services by all the major cloud service providers. Microsoft Azure is no exception, introducing its Azure Kubernetes Service (AKS) in June 2018.

While Kubernetes itself has become a standard of sorts for deploying and managing container-based microservices, the various managed Kubernetes offerings do not always exhibit the same consistency. In particular, the responsibilities and decisions required of the cloud customer can vary greatly. AKS brings its own set of service management overhead. Of particular importance, AKS users need to understand the configuration and maintenance required to maintain the security of AKS clusters.

Azure, like all major cloud service providers, operates under a shared model of responsibility. The provider takes responsibility for maintaining the reliability and security of their infrastructure and software, while the customers using their platform must follow best practices for using those services and for their own applications that run on and integrate with the provider's products.

While securing any production or critical application is important, a cloud-native platform like Kubernetes has some specific requirements and controls. Because the capabilities of Kubernetes particularly complement microservice-based application architectures, a single cluster can have large numbers of diverse containerized applications running on the same hosts. Maintaining the integrity of both a cluster's control plane and its application workload requires both protection from external threats and controlling possible damage from internal threats stemming from malicious or misconfigured code.

AKS clusters share these requirements with all Kubernetes clusters. However, in addition to universal best practices for securing Kubernetes clusters, AKS also has some provider-specific requirements. We will cover what AKS customers need to know and do to harden their clusters and safeguard their containerized workloads.

Cluster Design

Secure AKS clusters require a security-minded design. Understanding the fundamentals of Kubernetes security and specific AKS security options before creating clusters will make it easier to secure and manage clusters.

Some critical AKS security features can only be enabled at cluster creation time. For existing clusters which were not created with those features, it is highly advisable to create new clusters and migrate existing workloads.

Using consistent configurations across all clusters will also make them easier to manage via automation and prevent issues stemming from an incorrect assumption that all clusters have the same protections. In particular, clusters used for different application lifecycles, like development and staging, should have the same security settings as production. Identical environments allow for testing the security posture of a cluster and its workload before promotion to a production environment. This practice also helps to ensure applications running on clusters with those settings will still function as expected by the time they are deployed to production.

Enable Kubernetes RBAC

Why: Kubernetes Role-Based Access Control provides the preferred method for controlling authorization for a cluster's Kubernetes API, both for users and for workloads in the cluster. AKS adds the option of integrating K8s RBAC with Azure Active Directory, which can be enabled at any time for a cluster.

What to do: RBAC currently can only be enabled at AKS cluster creation time. It is enabled by default for new clusters.

Enable Network Policies

Why: Kubernetes Network Policies provide firewall controls to segment network traffic to and from workloads in a cluster.

What to do: Network Policy can only be enabled at AKS cluster creation time. Your options depend on your cluster's network type.

- Clusters using basic networking, with the kubenet plugin for the cluster network, support using the Calico CNI (Container Network Interface) to implement network policies.
- For clusters with advanced networking, which uses the Azure CNI, users can choose between Calico and Azure Network Policy.

Regardless of which implementation you use, both should function identically, and you can use the same Kubernetes network policy manifests with either provider.

Create Private Clusters

This feature is currently in preview and therefore may not be suitable for production use until it reaches general availability.

Why: Private AKS clusters have all their control plane components, including the cluster's Kubernetes API service, in a private RFC1918 network space. This limits access and keeps all traffic within Azure's networks. Access to the API can then be locked down by the user to specific VNets. Without this feature, the cluster's API has a public IPv4 address and all traffic to it, including from the cluster's node pools, goes over public network space.

Note that private clusters do have some limitations, and an Azure VM is required to use a bastion VM for connections to the cluster API not originating from the cluster's node pools. Azure also charges for the Private Endpoint resource needed to make the Kubernetes API available to VNets.

What to do: The private cluster option can only be selected at AKS cluster creation time.

Networking

Zero-trust principles have gained a great deal of acceptance as security best practice, especially for orchestrators like Kubernetes that can host multi-tenant workloads on shared infrastructure. The practice involves using tight access controls, following the principle of least privilege when granting access, encrypting all in-flight data, and requiring proof of identity from other services instead of blindly accepting connections just because they originated from the same shared network. Adopting a strict zero-trust environment requires a great deal of planning and management of the infrastructure and applications, but even achieving a subset of those goals can greatly benefit the security of your AKS clusters and their workloads.

Limit Node SSH Access

Why: By default, the SSH port on the nodes is open to all pods running in the cluster. Preventing direct SSH access from the pod network to the nodes helps limit the potential blast radius of damage if a container in a pod becomes compromised.

Azure's recommended method of getting ssh access to nodes, via a jump pod deployed in the AKS cluster, relies on allowing SSH access from the pod network to the nodes. You can create and use a bastion VM instead.

What to do: Find the Network Security Group(s) for your AKS subnet(s). Add an inbound security rule with a low, unused priority number and the following values:

- Source address prefixes: The CIDR block(s) of the AKS subnet(s) in the VNet
- Destination port ranges: 22
- Destination source prefixes: VirtualNetwork
- Protocol: TCP
- Access: Deny

Note that Azure does not officially support using network security group rules to limit a subnet's internal traffic for AKS (see the note in the introduction section of the linked page), but the above rule still works as expected.

You can also block pod access to the nodes' SSH ports using a Kubernetes Network Policy, if enabled in your cluster. Combining both methods provides the best protection. However, the Kubernetes Network Policy API does not support cluster-wide egress policies; network policies are namespace-scoped, which requires making sure a policy is added for each namespace, which requires ongoing vigilance. For clusters using the Calico CNI for network policy, users have another option. Calico supports additional Kubernetes resources beyond the standard Kubernetes network policy types, including GlobalNetworkPolicy, which can apply to the entire cluster.

Limit Network Access to the Kubernetes API

Why: By default in AKS, the Kubernetes API service for each cluster has a public IP address and has no firewall restrictions limiting access. Securing the API endpoint by limiting access only to the IP addresses which absolutely require access reduces risk due to unpatched or future vulnerabilities in the Kubernetes API service, exploitation of stolen credentials, and DDoS attacks.

What to do: If you aren't using a private AKS cluster, which does not have a public endpoint for the Kubernetes API, you can to restrict the public IP addresses able to access an AKS cluster's Kubernetes API.

Block Pod Access to VM Instance Metadata

Why: The Azure VM instance metadata endpoint, when accessed from an Azure VM, returns a great deal of information about the VM's configuration, including, depending on the VM and cluster configuration, Azure Active Directory tokens. This endpoint is accessible by any AKS containers on the node by default. Most workloads will not need this information and having access to that information can carry substantial risks.

What to do: Add a network policy in all user namespaces to block pod egress to the metadata endpoint.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-instance-metadata
spec:
  podSelector:
    matchLabels: {}
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0     # Preferably something smaller here
        except:
        - 169.254.169.254/32
```

Or, if you are already using network policy whitelists to control pod egress traffic, be sure to **exclude** `169.254.169.254/32` from the allowed IP blocks.

If some workloads absolutely need access to the VM metadata, you can either add exceptions with label-scoped policies, or use a service mesh which can allow path-based routing restrictions to an endpoint.

Restrict Cluster Egress Traffic

Why: By limiting egress network traffic only to known, necessary external endpoints, you can limit the potential for exploitation by compromised workloads in your cluster.

What to do: AKS provides several options for controlling cluster egress traffic. They can be used separately or together for better protection.

1. Use Kubernetes network policies to limit pod egress endpoints. Policies need to be created for every namespace or workload.
2. Use the Calico Network Policy option in AKS, which adds additional resource types to Kubernetes Network Policy, including a non-namespaced `GlobalNetworkPolicy`.
3. Use an Azure firewall to control cluster egress from the VNet. If using this method, note the external endpoints that the cluster's nodes (not necessarily the workload pods) need to reach for proper functionality and make firewall exceptions as needed.

Disable HTTP Application Routing

Why: The AKS HTTP Application Routing add-on is an ingress controller that creates public DNS entries for Kubernetes Services deployed in a cluster and exposes them on the cluster load balancer. While this feature can simplify development and testing of applications, it lacks options for securing the exposed services, such as being able to secure HTTP services with TLS. It should never be enabled in a production cluster, and even for development clusters, it can pose a risk by exposing untested, unsecured applications to the Internet.

What to do: Do not enable the HTTP Application Routing add-on for any cluster. If it has already been enabled in a cluster, remove it.

Deploy a Service Mesh

Why: Service meshes have emerged in the past few years to address the networking complexities of microservices. Different service mesh offerings have different feature sets, but most of them offer advanced traffic management, observability, and critical zero-trust network support by providing seamless pod-to-pod encryption, authentication, and authorization.

What to do: Evaluate which service mesh offering best meets your organization's needs. Istio, Linkerd, and Consul can all be deployed to AKS clusters. Service meshes can be quite powerful, but they can also require a great deal of configuration to secure workloads properly.

Container Images

Kubernetes workloads are built around container images, therefore ensuring that those images are difficult to exploit and are kept free of security vulnerabilities should be a cornerstone of your AKS security strategy.

Build Secure Images

Why: Following a few best practices for building secure container images will minimize the exploitability of running containers and simplify both security updates and scanning. Images that contain only the files required for the application's runtime and which do not include frequently exploited or vulnerable tools like Linux package managers, web or other network clients, or Unix shells make it much more difficult for malicious attacks to compromise or exploit the containers in your cluster.

What to do:

1. Use a minimal, secure base image. Google's "Distroless" images are a good choice because they do not install OS package managers or shells.
2. Only install tools needed for the container's application. Debugging tools should be omitted from production containers.
3. Instead of putting exploitable tools like curl in your image for long-running applications, if you only need network tools at pod start-time, consider using separate init containers or delivering the data using a more Kubernetes-native method, such as ConfigMaps.
4. If you do need to install additional OS packages, remove the package manager from the image in a later build step.
5. Keep your images up-to-date. This practice includes watching for new versions of both the base image and any third-party tools you install.

Use a Vulnerability Scanner

Why: Using containers free of known software security vulnerabilities requires ongoing vigilance. All the images deployed to a cluster should be scanned regularly by a scanner that keeps an up-to-date database of CVEs (Common Vulnerabilities and Exposure).

What to do: Use an image scanner. Azure has a scanner service available in preview mode, or you can choose your own paid or open source scanner. Be sure to choose a solution that scans for vulnerabilities in OS packages and in third-party runtime libraries for the programming languages your software uses. For example, the Apache Struts vulnerability exploited in the Equifax hack would not have been detected by an OS package scan alone, because Struts is a set of Java libraries, and most Linux distributions no longer manage software library dependencies that are not required by system services.

To address CVEs when they are found in your internally maintained images, your organization should have a policy for updating and replacing images known to have serious, fixable vulnerabilities for images that are already deployed. Image scanning should be part of your CI/CD pipeline process and images with high-severity, fixable CVEs should generate an alert and fail a build.

If you also deploy third-party container images in your cluster, scan those as well. If those images have serious fixable vulnerabilities that do not seem to get addressed by the maintainer, you should consider creating your own images for those tools.

Runtime Security for Workloads

The security health of your AKS clusters depends just as much on the workloads deployed to the cluster as on the secure configuration of the AKS infrastructure. This section covers how to plan and deploy your applications to keep your nodes, network, cluster, and applications safe.

Namespaces!

Why: Kubernetes namespaces provide crucial scope for cluster objects, allowing for fine-grained cluster object management. Kubernetes RBAC rules for most resource types apply at the namespace level. Controls like network policies and many add-on tools and frameworks like service meshes also often apply at the namespace scope.

What to do: Plan out how you want to assign namespaces before you start deploying workloads to your clusters. Having one namespace per application provides the most opportunity for control, although it does bring extra management overhead when assigning RBAC role privileges and default network policies. If you do decide to group more than one application into a namespace, the main consideration should be whether those applications have common RBAC requirements and whether it would be safe to grant those privileges to the service accounts and users who need Kubernetes API access in that namespace.

Configure Kubernetes RBAC

Why: Kubernetes Role-Based Access Control provides the standard method for managing authorization for the Kubernetes API endpoints. The practice of creating and managing comprehensive RBAC roles that follow the principle of least privilege, in addition to performing regular audits of how those roles are delegated with role bindings, provides some of the most critical protections possible for your AKS clusters, both from external bad actors and internal misconfigurations and accidents.

What to do: Configuring Kubernetes RBAC effectively and securely requires some understanding of the Kubernetes API. You can start with the official documentation, read about some best practices, and you may also want to work through some tutorials.

Once your team has solid working knowledge of RBAC, create some internal policies and guidelines. Make sure you also regularly audit your `Role` permissions and `RoleBindings`. Pay special attention to minimizing the use of `ClusterRoles` and `ClusterRoleBindings`, as these apply globally across all namespaces and to resources that do not support namespaces. (You can use the output of kubectl `api-resources` in your cluster to see which resources are not namespace-scoped.)

Azure also supports using Azure Active Directory with your AKS cluster RBAC for user authentication and authorization management.

Use Kubernetes Network Policies

Why: By default, all pods in a Kubernetes cluster can make network connections to any listening ports in containers in other pods. The Kubernetes Network Policy API provides the ability to create network firewall rules for the ingress and egress traffic for your cluster's pods. Limiting access to pods based on your distributed applications' requirements reduces the potential for damage if a pod in the cluster becomes compromised by a malicious agent.

What to do: As noted in the Cluster Design section, a network policy option must be selected at cluster creation time.

See our posts on guidelines for writing ingress and egress network policies.

Limit Container Runtime Privileges

Why: Most containerized applications will not need any special host privileges on the node to function properly. By following the principle of least privilege and minimizing the capabilities of your cluster's running containers, you can greatly reduce the risk of exploitability or accidental damage by misbehaving applications.

What to do: Use the PodSpec Security Context to define the exact runtime requirements for each workload. Use Pod Security Policies and/or admission controllers like Open Policy Agent (OPA) Gatekeeper to enforce those best practices by the Kubernetes API at object creation time.

Some guidelines:

- Do not run application processes as root.
- Do not allow privilege escalation.
- Use a read-only root filesystem.
- Use the default (masked) /proc filesystem mount.
- Do not use the host network or process space.
- Drop unused Linux capabilities and do not add optional capabilities that your application does not absolutely require. (The available capabilities depend on the container runtime in use. AKS uses the Docker/Moby runtime, which supports these capabilities. The first table lists capabilities loaded by default, while the second table shows optional capabilities that may be added.)
- Use SELinux options for more fine-grained process controls.
- Give each application its own Kubernetes Service Account.
- Do not mount the service account credentials in a container if it does not need to access the Kubernetes API.

Use Pod Security Policies

Note that AKS support for Kubernetes Pod Security Policy is currently in preview.

Why: Kubernetes Pod Security Policy provides a method to enforce best practices around minimizing container runtime privileges, including not running as the root user, not sharing the host node's process or network space, not being able to access the host filesystem, enforcing SELinux, and other options. Most cluster workloads will not special permissions and by forcing containers to use the least-required privilege, their potential for malicious exploitability or accidental damage can be minimized.

What to do: Follow the instructions for enabling Pod Security Policy in your AKS cluster. Make sure you test your PSPs in a non-production cluster before enabling them in production. Pod Security Policy support can be enabled in an existing AKS cluster at any time.

Use an Admission Controller to Enforce Best Practices

Note that Azure Policy for AKS is currently in preview.

Why: Kubernetes supports using admission controllers, which can be configured to evaluate requests to the Kubernetes API. In the case of validating controllers, an admission controller can deny requests that fail to meet certain requirements, or for mutating controllers, make changes to the request, such as injecting a sidecar container to a pod or adding labels to an object, before sending it to the Kubernetes API.

While users can deploy their own admission controllers to perform a variety of tasks to AKS clusters, AKS also now offers an integration with Azure Policy, Azure's governance service. Azure Policy for AKS uses the Open Policy Agent (OPA) Gatekeeper admission controller to enforce a variety of best practices by preventing non-conforming objects from getting created in an AKS cluster. While some overlap of Pod Security Policy capabilities exists, OPA allows restrictions not just on pods, but on virtually any attribute of any cluster resource.

What to do: Follow these instructions for enabling Azure Policy for AKS in your cluster.

At least for now, Azure Policy for AKS only supports using AKS-supplied policy definitions. If you want to leverage the full power of OPA, you can install and manage the Gatekeeper admission controller yourself.

Cluster Maintenance and Other Tasks

AKS brings some specific operational maintenance requirements on top of universal best practices for maintaining secure Kubernetes clusters.

Monitoring

Why: You cannot keep your cluster secure and healthy if you do not know what's happening in it.

What to do: A number of tools and options exist for monitoring your AKS cluster. Some tools are more security-aware than others, though, so make sure you have the following areas covered at a minimum:

1. Monitor your application logs and performance for anomalies.
2. Enable AKS master component logs so you can view and monitor those. At a minimum, you will want to collect logs for the following components:
   - kube-apiserver – Logs all calls to the cluster's Kubernetes API, including source IP addresses
   - kube-audit – Kubernetes audit events
3. Audit your RBAC roles and bindings regularly.

AKS Security Updates

Why: AKS does not fully manage keeping your cluster up-to-date with Kubernetes/AKS security patches. Keeping up with the latest security patches is foundational for good AKS and Kubernetes security.

What Azure manages:

- Makes AKS upgrades available
- Make Windows OS upgrades available (for Windows nodes)
- Applies Linux OS patches nightly

What the user must manage:

- Rebooting/replacing Linux nodes when OS patches require it
- Applying Windows OS patches to nodes and rebooting
- Applying AKS upgrades to masters and all nodes

What to do: Most importantly, you will need to find a reliable way to watch for AKS security update announcements. Your organization will need to automate or create a procedure for these tasks, as applicable.

- Applying AKS upgrades for masters and all nodes
- Rebooting Linux nodes as needed for OS patches (not all patches require reboots)
- Apply AKS and OS updates to Windows nodes and reboot

Managing the Azure Service Principal

Note that the managed identities feature for AKS is currently in preview.

Why: Azure uses an Active Directory service principal to perform the creation and update of the Azure resources needed by an AKS cluster. By default, this service principal's credentials are valid for one year, after which point the user must manage updating the credentials to maintain proper functionality of the associated AKS cluster. Automating the rotation of these static credentials within a shorter lifespan will improve the security posture and reliability of your clusters and Azure resources.

What to do: Follow these instructions for manually rotating the service principal's credentials.

Users can also choose managed identities to avoid the use of static credentials. Managed identities must be enabled at cluster creation time.

Rotate Cluster Certificates

Why: AKS creates a number of TLS certificates for various control plane and node components. For clusters created after March 2019, these certificates are valid for 30 years. Many compliance audits require much shorter lifespans for certificates. Protecting these certificates, especially if you are not using a private cluster, is critical.

What to do: You can rotate these certificates, but be aware, it can require downtime of up to 30 minutes for the AKS cluster.

The Kubernetes Dashboard

Why: AKS installs the Kubernetes Dashboard in every cluster and it cannot be deleted permanently. (If the deployment, role binding, or service objects are deleted, they automatically get recreated.)

The Kubernetes Dashboard is a major source of concern for several reasons.

1. It requires a number of cluster RBAC permissions to function. Even if it has only been granted read-only permissions, the dashboard still shares information that most workloads in the cluster should not need to access. Users who need to access the information should be using their own RBAC credentials and permissions.
2. AKS does not enable any authentication requirement for the dashboard. Any entity that can connect to the endpoint has full access. Although the associated Kubernetes service only has an internal endpoint, any pod in the cluster and any user who can run `kubectl port-forward` in the cluster can access it.
3. The Kubernetes Dashboard has been the subject of a number of CVEs. Because of its level of access to the cluster's Kubernetes API and its lack of internal controls, vulnerabilities can be extremely dangerous.

What to do:

1. Do not grant the `kubernetes-dashboard` service account any additional privileges, and remove any bindings to the service account. (Note that the `kubernetes-dashboard-minimal` role in the `kube-system` namespace is managed by AKS and will get recreated automatically if it gets deleted.)

a. Shell command to find cluster role bindings that apply to the `kubernetes-dashboard` service account (requires the `jq` tool) :

```
kubectl get clusterrolebindings -ojson | \
  jq -r ‘.items[] | select(.subjects[]? | \
 (.name == “kubernetes-dashboard”) and \
 (.kind == “ServiceAccount”) and \
 (.namespace == “kube-system”)) | \
  .metadata.name’
```

b. Shell command to find all role bindings in all namespaces that bind to the `kubernetes-dashboard` service account (requires the `jq` tool):

```
kubectl get rolebindings --all-namespaces -ojson | \
  jq -r ‘.items[] | select(.subjects[]? | \
  (.name == “kubernetes-dashboard”) and \
  (.kind == “ServiceAccount”) and \
  (.namespace == “kube-system”)) | \
  [.metadata.name, .metadata.namespace]’
```

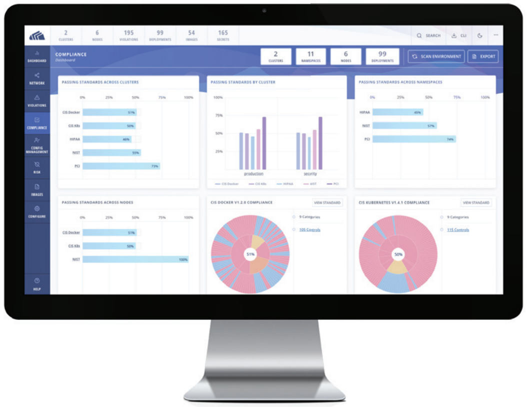
c. Use network policies to block all ingress to the dashboard.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: block-kubernetes-dashboard
  namespace: kube-system
spec:
  podSelector:
     matchLabels:
        k8s-app: kubernetes-dashboard
  policyTypes:
  - Ingress
  ingress: []
```

AKS Virtual Nodes

Why: Some critical Kubernetes controls, including network policies, do not apply to pods running on Virtual Nodes. Given the reduced visibility into and controls for Virtual Nodes, they may not be suitable for many workloads.

What to do: Understand the limitations of Virtual Nodes and decide if the risk is worth the convenience factor. Autoscaling node pools generally provide a suitable alternative for controlling costs and capacity for applications with variable loads or for task-based workloads, without compromising cluster security.



Ready to see StackRox in action?

Get a personalized demo tailored for your business, environment, and needs.

REQUEST DEMO