aqua Microsoft Azure

The Definitive Guide to Securing Kubernetes

Liz Rice | TECHNOLOGY EVANGELIST | AQUA SECURITY
Brendan Burns | DISTINGUISHED ENGINEER AND KUBERNETES CO-CREATOR | MICROSOFT

Introduction

As the adoption of Kubernetes (K8s) continues to expand, one topic that often comes up is security. Since K8s is used to run applications on a large scale and controls who and what can be done with these applications, security should be well-understood and managed.

The goals of a secure K8s environment are first and foremost to ensure that the applications it runs are protected, that security issues can be identified and addressed quickly, and that future similar issues will be prevented—all without affecting developer productivity or negating the inherit benefits of K8s, such as resilience, scalability, speed, and automation.

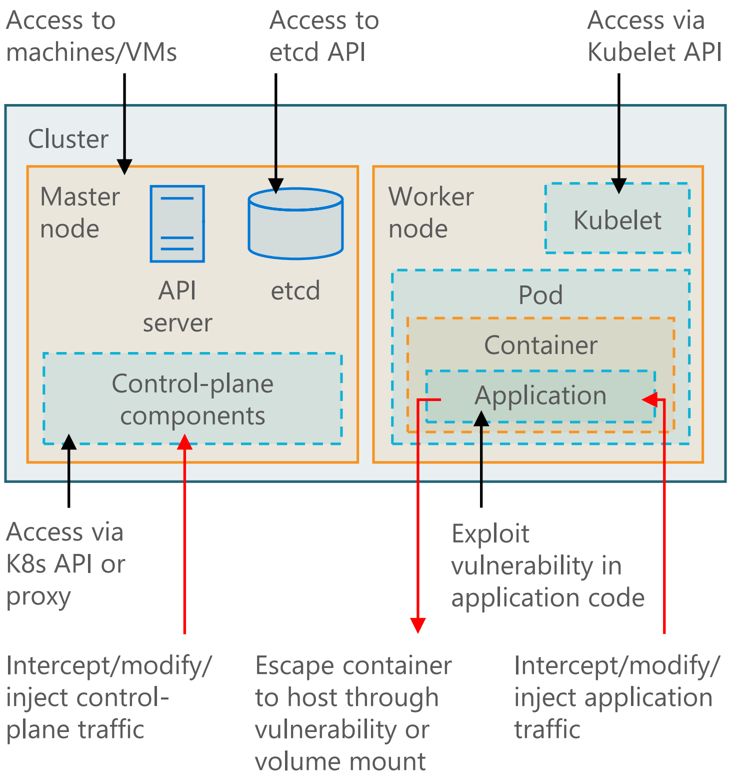
We assume the reader is familiar with basic K8s concepts. If you're not, we recommend visiting aka.ms/K8slearning for the basics or kubernetes.io for tutorials.

This whitepaper provides an overview of key aspects and best practices of K8s security, including:

- Securing the cluster
- Managing authorization and authentication
- Implementing a trusted software supply chain
- Securing workloads in runtime
- Managing secrets

K8s is available in multiple distributions and delivery models. In this paper, the best practices we list apply to the open source K8s (often referred to as "upstream K8s"), but occasionally we call out specific features of a managed service like Azure Kubernetes Service (AKS) and additional tools.

At the time of publishing this paper, K8s is at Version 1.15. For most purposes, the recommendations here apply to K8s 1.11 onwards. Where K8s settings are noted, use them to search kubernetes.io/docs/ for additional information.


Access to machines/VMs
Access to etcd API
Access via Kubelet API
Cluster
Master node
API server
etcd
Control-plane components
Worker node
Kubelet
Pod
Container
Application
Access via K8s API or proxy
Exploit vulnerability in application code
Intercept/modify/ inject control-plane traffic
Escape container to host through vulnerability or volume mount
Intercept/modify/ inject application traffic

Securing the cluster

The Master Node controls the configuration and operation of the entire cluster and is therefore a key area to secure.

API SERVER

The API server offers REST API access to control the cluster. From v1.10 onwards, the kubeadm installer disables the API server's insecure port so that API access is restricted to encrypted TLS connections made over a secured port by default, but is not limited by default to authenticated users. To further limit access to the API server, you can:

- Prevent unauthenticated users from accessing it, by setting `--anonymous-auth=false`. This means that all API server access, including health checks and service discovery, must be authenticated.
- Permit unauthenticated user access but limit it using role-based access control (RBAC). By default, the RBAC setting permits very limited access to anonymous users, so that a client can make health checks and service discovery can be performed without providing certificates. This is the default approach, but it does rely on you maintaining sensible RBAC policies that restrict what anonymous users can do.
- Further protect API server access with additional measures, such as a traditional firewall or VPN.
- If you plan to install the Kubernetes Dashboard and use it to connect to the API server, make sure it too has restricted access and is not exposed on the Internet.

ETCD

K8s stores cluster configuration and state information in a distributed key-value store named etcd. Unauthorized access to etcd may jeopardize the entire cluster, which is why access to it should be strictly limited.

- Set `--cert-file` and `--key-file` to enable HTTPS connections to etcd.
- Set `--client-cert-auth=true` to that ensure only authenticated access to etcd is allowed.
- Set `--trusted-ca-file` to specify the certificate authority and set `--auto-tls=false` to prevent the use of self-signed certificates.
- Set `--peer-client-cert-auth=true` to force etcd nodes to communicate securely with each other.

KUBELET

The kubelet is an agent that runs on each worker node and interacts with the container runtime to launch pods and report node and pod status. Unauthorized access to a kubelet can allow starting and stopping pods, as well as executing unauthorized code.

- Disable anonymous access with `--anonymous-auth=false` and have the API server identify itself by setting the `--kubelet-client-certificate` and `--kubelet-client-key` flags.

VALIDATING CLUSTER CONFIGURATION

The Center for Internet Security (CIS) publishes a benchmark document for K8s (as well as Docker, which you might want to follow as well). The benchmark lists more than 100 recommended configurations and is currently only valid for the open source K8s distribution. Using it on some commercial distributions and managed offerings might yield partial or inaccurate results.

It is recommended to check your cluster often against these benchmarks. Doing so manually is very time-consuming, but the open source kube-bench can automate this task and provide pass/fail results on all the benchmark's tests.

DID YOU KNOW? While the CIS benchmark recommends configuration best practices, you can also perform penetration testing on your cluster to test its resilience against real-world attack vectors with the open source kube-hunter.

Managing authorization and authentication

Since K8s is a distributed and sometimes sprawling environment, it's important to use authentication for multiple components (not just the API server) to prevent unwanted users or service accounts from accessing cluster components and data (such as kubelets, kube proxies, and secrets). Likewise, authorization controls should be used to prevent authenticated users from having blanket access to capabilities they don't need, enforcing the least privilege principle of security.

DID YOU KNOW? With Azure Active Directory (AD)-integrated AKS clusters, you can grant users or groups access to K8s resources within a namespace or across the cluster based on your existing AD configuration.

AUTHENTICATION MODELS

There are several ways to authenticate access. We do not recommend using static password files, since they can be accessed too easily and are difficult to manage over time.

- **X.509 Client Certificates** use a 3rd party certificate authority (CA) to verify the user identity. Client certificate authentication is enabled by passing the `--client-ca-file=SOMEFILE` option to the API server. The referenced file must contain one or more certificates authorities to validate client certificates presented to the API server. As of K8s Version 1.14, you can include the user's group memberships by including multiple organization fields in the certificate.

DID YOU KNOW? In AKS, cluster security is provided by Azure, including nightly patches. Since the cluster master node is managed by Azure, users cannot access it directly—making it very hard for potential intruders to access, too.

- **Token Files** can be read by the API server when given the `--token-auth-file=SOMEFILE` option on the command line. Currently, tokens last indefinitely—this is a limitation of this approach—and the token list cannot be changed without restarting the API server.
- **Bootstrap Tokens** offer an alternative approach, but this feature is currently (v1.14) in beta. They allow for dynamic management of tokens, which are stored as secrets in etcd. To use them, you must enable the Bootstrap Token Authenticator with the `--enable-bootstrap-token-auth` flag on the API server. Then you must enable the TokenCleaner controller using `--controllers=*,tokencleaner` on the Controller Manager.
- Service accounts use **Service Account Tokens** to identify themselves. Service accounts are associated with pods running in the cluster through the ServiceAccount Admission Controller. Bearer tokens are mounted into pods at well-known locations and allow in-cluster processes to talk to the API server.
- Finally, **OpenID Connect Tokens** can be a convenient way of using OAuth2 providers, including Azure Active Directory (AD). This token is a JSON Web Token (JWT) with well-known fields, like a user's email, signed by the server.

You can use multiple authentication methods within a single environment, and the API server will accept the first approved authentication, with no particular order specified. This is useful if you don't want to rely on a single service, or if you already use diverse sources for identity management.

ROLE-BASED ACCESS CONTROL (RBAC)

The K8s RBAC model uses several objects to govern resource authorization.

- **Entity**: A user, group or service account
- **Resource**: Something the entity will access, like a pod, secret, or service

  Role: Used to define rules specifying a set of actions that are permitted on a set of resources
- **RoleBinding**: Attaches a role to an entity, defining the actions the entity can perform on resources

Roles and RoleBindings are namespaced resources that apply only within the specific namespace where they're defined, and they have cluster-wide equivalents called ClusterRoles and ClusterRoleBindings.

The actions on a resource are described by verbs as follows.

- **Read-only actions**: `get, list, watch`
- **Read-write actions**: `create, update, patch, delete, proxy, redirect, deletecollection`

While K8s has predefined roles for system administration purposes, you should:

- Define your own roles in the context of your application needs, team structure, and security and compliance mandates. The principle of least privilege should apply.
- Differentiate between namespace-wide and cluster-wide roles and use the latter sparingly.
- Differentiate between service accounts and human users, defining them as distinct entities.

The K8s RBAC model has built-in privilege escalation barriers, whereby a user can only create or update a role if they already have all the permissions contained in that role. They cannot create new roles with privileges they themselves don't have.

RBAC should be used to create segregation of duties. This security principle prevents a single user from having too much authority. For example, it is a good idea to prevent cluster admins from having read-write access to K8s audit logs and audit policy files, so that they cannot overwrite or delete log files or stop event auditing altogether.

Implementing a trusted software supply chain

K8s is ultimately used to run software in the form of containers. Where those containers came from and what they contain has a direct impact on the security of your K8s applications. This means implementing controls across the pipeline to ensure that what goes in is validated, and that code integrity is maintained throughout.

DID YOU KNOW? Aqua MicroScanner is a free Docker image vulnerability scanning tool that you can embed into your Dockerfile and your CI pipeline to automate scanning and provide developers with immediate contextual feedback on CVEs found.

SOURCE CONTROL

Ensure that container images are created using an approved set of base images, and do not allow developers to access unvetted open source components and use them in images. This can be achieved using traditional IP-blocking methods (for example to Docker Hub, which holds hundreds of thousands of images), as well as with image scanning to detect base image packages.

IMAGE SCANNING

It is strongly recommended to regularly scan images, both during CI/CD builds as well as in the registry, to detect known vulnerabilities as well as other issues such as embedded secrets and image configuration issues.

Known vulnerabilities present a potentially high risk that's easy to detect and manage, and eliminating or mitigating them (especially high severity ones) should be step one of any security program.

NOT USING ROOT USER

Unless required for a specific use case, you should never configure your images to run as a root user. Running a container as root will make it easier for an attacker to use a compromised container to control the host. Include a `USER` command in the Dockerfile to define a user identity for the container. Even better would be to set a different user ID for different images, which would make auditing and forensics more accurate.

DID YOU KNOW? When using Azure Pipelines as your CI/CD platform, you can define and apply Azure Policy to surface security violation pre-deployment, and monitor for compliance post-deployment, providing feedback to Pipelines users.

IMAGE INTEGRITY CONTROLS

Images are constantly being updated and pushed through the pipeline. It's important to ensure that the containers that end up running in your cluster are instantiated from the correct images, with no tampering or drift, especially in applications that are mission-critical or that handle sensitive data.

Several commercial platforms (including Aqua CSP) provide lifecycle controls for images as part of their workflow. Another approach is using image signing to ensure the integrity of your images from build to runtime. The open source TUF project (theupdateframework.github.io/) and its implementation Notary (docs.docker.com/notary/) provide image signing capabilities.

ENFORCING THE USE OF TRUSTED IMAGES

Create a set of criteria and implement controls that only allow K8s to deploy trusted images. It's possible to do this by only allowing approved images from CI/CD to be deployed, while only allowing K8s to pull images from your registry. The Open Policy Agent (OPA)-based Gatekeeper project provides an easy way to implement this.

You can also use the `AlwaysPullImages` admission control to force nodes to pull images on every instantiation and not use cached versions (though this may cause delays in container instantiation and is not recommended for very ephemeral workloads).

DID YOU KNOW? Azure Container Registry (ACR) allows you to quarantine new or updated images until they have been scanned by 3rd party scanners, including Aqua, and deemed safe to use. A quarantined image will be invisible to developers until it's released from quarantine.

SECURING THE REGISTRY

Registries hold images that could be used to "poison the well" with malicious code before they're instantiated. It is highly recommended to restrict registry access—especially write access—with authentication. Since K8s will pull images from a registry, configure its registry credentials to be read-only. Managed registries such as Azure Container Registry (ACR) require authentication by default.

Securing workloads in runtime

In addition to the best practices discussed so far, it is recommended to place boundaries and controls that limit what an application can do in runtime. This defense in depth can limit the damage of an attack or prevent an intruder from getting past their initial intrusion point.

K8s allows you to do that with several of its native policies that, when properly configured, can together create a very secure environment.

SECURITY CONTEXT

K8s security context defines privileges and access control settings either at the pod or the container level. The security context allows you to:

- **Implement access control** based on user ID or group ID.
- Determine if a container runs as **privileged or unprivileged**.
- Set **Linux capabilities**, which is useful when capabilities associated with root privilege are required but avoids granting blanket root privilege.
- Apply a **seccomp profile** for a pod or container, restrict their access to Linux system calls, or use **AppArmor** or **SELinux** to restrict capabilities and processes they can use on the system.

DID YOU KNOW? Aqua runtime controls include the ability to detect and prevent drift between running containers and their originating images. This is used to prevent code injection into running containers, enforcing the concept of immutability.

POD SECURITY POLICY

While security context can be set individually for pods and containers, `PodSecurityPolicy` allows you to set a cluster- or namespace-wide security context. It allows setting parameters such as access to files, volumes, host namespaces, host ports, use of privileged containers, Linux capabilities, SELinux context, and more. To enable this, you must activate the `PodSecurityPolicy` admission controller.

Using `SecurityContext` and `PodSecurity Policy` in tandem is a great way of ensuring least privilege security settings at the pod level. You can limit certain types of pods to access the file system as read-only or disable privileged containers from running in certain pods.

In conjunction with other admission controllers, such as `DenyEscalatingExec` (disallow a pod from gaining privileges not already present) or `PodNodeSelector` (restrict pods to run on specific nodes by label), you can programmatically protect the cluster from pods behaving unexpectedly, curb damage from developer errors, and create barriers against both external intruders and insider abuse.

DID YOU KNOW? Aqua offers image assurance policies that allow setting risk threshold for images based on CVE score or severity, hard-coded secrets, malware, image configuration, and custom compliance checks. If an image fails policy, it will not be allowed to run.

NETWORK POLICY

Since pods are the base networking unit in K8s environments, placing guardrails on pod network traffic is a great way to prevent unwanted "East-West" traffic (between nodes on a cluster) as well as "North-South" traffic (between the pod and other layers or external resources).

Different distributions of K8s may use different flavors of Container Network Interface (CNI) to implement network policy. Popular ones include Calico, Weave, Flannel, and Cilium.

You can prevent traffic, both ingress (inbound) or egress (outbound), using Kubernetes Network Policy. You can also limit traffic to specific destination or origin to ensure that traffic only flows according to the application requirements. This type of whitelisting is also being implemented through service meshes, which can be easier for developers to use to define those requirements.

Note that *K8s namespaces*, while enabling tenancies within a cluster, should not be considered a substitute for network security controls. They provide isolation for administration purposes but are insufficient to prevent (or detect) network traversal attempts.

Managing secrets

Secrets, such as private keys or passwords, are often needed for a container to access services or data. The challenge is ensuring that the secret is accessible only from the intended container and isn't exposed elsewhere. You probably don't want to expose production credentials to team members, even if they have permissions to manage pods (or other resources) in the cluster.

PRACTICES TO AVOID

We often see two types of "shortcuts" used for reasons of convenience that we strongly recommend avoiding completely.

- **Hard-coding or embedding secrets in images.** Not only does this expose the secret to anyone with access to the registry or CI/CD environment, it also breaks secret lifecycle controls, such as rotation, update, and revocation.
- **Using unencrypted environment variables**. If secrets are defined as environment variables in pod specifications, they are exposed to anyone with read access on the pod.

STORING SECRETS IN ETCD AND 3RD PARTY VAULTS

It is possible to store secrets in etcd. However, by default etcd is not encrypted, only base64 encoded, which can be read by anyone with a base64 reader. This means that anyone with access to etcd (or its data store) might access secrets.

From Version 1.13 or later, you can avoid this by encrypting your cluster on disk using the `--encryption-provider-config` command.

Many organizations already use 3rd party vaults to manage secrets used by non-K8s applications, such as HashiCorp Vault or Azure Key Vault. Such vaults are highly secure and have built-in lifecycle controls to avoid unnecessary transfer or duplication of secrets.

DID YOU KNOW? You can integrate Azure Key Vault into your AKS cluster using FlexVolume, which mount multiple secrets, keys, and certs stored in Key Vault into pods as a volume. Once the Volume is attached, the data in it is mounted into the container's file system.

PASSING SECRETS TO CONTAINERS

K8s can pass secrets to containers using one of two methods.

- **Environment variables**. It is possible to pass K8s secrets using environment variables. However, they will be visible to any user with permission to run the `kubectl describe` or `docker inspect` commands, so stricter RBAC should be used to limit user access to pods. It's also common for applications to log their environment in the event of a crash, which exposes secrets held in environment variables to anyone with access to the log output.
- **Volume mount**. This is the recommended method, since the file is not readable using `kubectl describe` and `docker inspect`. The volume is only accessible in a specific container, making it much harder to access even to users with access to the namespace. Using a temporary file system also means the secret does not persist on disk.

Either way, you should limit access to the secrets API using RBAC authorization policies, strictly limiting use of the `watch` and `list` commands to specific users.

DID YOU KNOW? Aqua CSP provides a solution for securely injecting secrets into containers, using 3rd party vaults, with both in-transit and at-rest encryption, using temporary volume mounts, and allowing secrets rotation/revocation with no container restart.

Summary

Kubernetes is an evolving platform, but has achieved a level of maturity of its security features that makes it a viable choice for running critical applications. It is still a complex system that requires some understanding of key elements of its security model. While the community is constantly improving security controls and implementing more fool-proof defaults, leveraging managed services like Azure AD or Azure Policy that integrate well with your existing security controls can be a good strategy. Together with commercial security platforms such as Aqua CSP, they provide additional security layers as well as easier visibility and management, reducing time to market and making secure K8s applications more accessible.

FOR MORE INFORMATION ABOUT AQUA
Go to our website at aquasec.com

FOR MORE INFORMATION ABOUT KUBERNETES ON AZURE
Visit aka.ms/aks/page to learn how AKS simplifies Kubernetes development

NEW TO KUBERNETES?
Visit aka.ms/k8slearning to learn more about Kubernetes

WANT TO GAIN HANDS-ON EXPERIENCE?
Check out devsecops.aksworkshop.io

START FOR FREE WITH AZURE CREDITS
Go to aka.ms/aksfree to deploy and manage Kubernetes with a free Azure account