Security Overview

Google Kubernetes Engine provides many ways to help secure your workloads (https://cloud.google.com/kubernetes-engine/docs/how-to/deploying-workloads-overview). Protecting workloads in Google Kubernetes Engine involves many layers of the stack, including the contents of your container image, the container runtime, the cluster network, and access to the cluster API server.

It's best to take a layered approach to protecting your clusters and workloads. You can apply the principle of least privilege (https://en.wikipedia.org/wiki/Principle_of_least_privilege) to the level of access provided to your users and your application. In each layer there may be different tradeoffs that must be made that allow the right level of flexibility and security for your organization to securely deploy and maintain their workloads. For example, some security settings may be too constraining for certain types of applications or use cases to function without significant refactoring.

This document provides an overview of each layer of your infrastructure, and shows how you can configure its security features to best suit your needs.

Authentication and authorization

Kubernetes supports two types of authentication (https://kubernetes.io/docs/reference/access-authn-authz/authentication/#users-in-kubernetes):

1. **User accounts** are accounts that are known to Kubernetes, but are not managed by Kubernetes - for example, you cannot create or delete them using `kubectl`.
2. **Service accounts** are accounts that are created and managed by Kubernetes, but can only be used by Kubernetes-created entities, such as pods (https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/).

In a Google Kubernetes Engine cluster (https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture), Kubernetes user accounts are managed by Google Cloud Platform (GCP), and may be one of the following two types:

1. Google Account (https://support.google.com/accounts/answer/27441?hl=en)

2. Google Cloud Platform (GCP) service account (https://cloud.google.com/compute/docs/access/service-accounts)

Once authenticated, you need to authorize these identities to create, read, update or delete Kubernetes resources.

Despite the similar names, Kubernetes service accounts and GCP service accounts are different entities. Kubernetes service accounts are part of the cluster in which they are defined and are typically used within that cluster. By constrast, GCP service accounts are part of a GCP project, and can easily be granted permissions both within clusters and to Google Cloud Platform project clusters themselves, as well as to any GCP resource using Cloud Identity & Access Management (Cloud IAM) (https://cloud.google.com/kubernetes-engine/docs/how-to/iam-integration#managing_iam_roles) . This make GCP service accounts more powerful than Kubernetes service accounts; in order to follow the security principle of least privilege, you should consider using GCP service accounts only when their capabilities are required.

To configure more granular access to Kubernetes resources at the cluster level or within Kubernetes namespaces, you use Role-Based Access Control (RBAC) (https://cloud.google.com/kubernetes-engine/docs/how-to/role-based-access-control). RBAC allows you to create detailed policies that define which operations and resources you allow users and service accounts to access. With RBAC, you can control access for Google Accounts, GCP service accounts, and Kubernetes service accounts. To further simplify and streamline your authentication and authorization strategy for Google Kubernetes Engine, you should ensure that the legacy Attribute Based Access Control (https://cloud.google.com/kubernetes-engine/docs/reference/rest/v1/projects.zones.clusters#legacyabac) is disabled so that Kubernetes RBAC and Cloud IAM are the source of truth.

For more information:

- Read the Google Kubernetes Engine RBAC documentation (https://cloud.google.com/kubernetes-engine/docs/how-to/role-based-access-control)
- Follow best practices in the IAM section of Preparing Google Kubernetes Engine for Production (https://cloud.google.com/solutions/prep-kubernetes-engine-for-prod#managing_identity_and_access)

Control plane security

In Google Kubernetes Engine, the Kubernetes master components (https://kubernetes.io/docs/concepts/overview/components/#master-components) are managed and maintained by Google. The master components host the software that runs the Kubernetes control plane, including the API server, scheduler, controller manager and the etcd database (https://github.com/coreos/etcd) where your Kubernetes configuration is persisted.

By default, the master components use a public IP address. You can protect the Kubernetes API server by using master authorized networks (https://cloud.google.com/kubernetes-engine/docs/how-to/authorized-networks), and private clusters (https://cloud.google.com/kubernetes-engine/docs/how-to/private-clusters), which allow you to assign a private IP address to the master and disable access on the public IP address.

You can handle cluster authentication in Google Kubernetes Engine by using Cloud IAM as the identity provider. However, legacy username-and-password-based authentication (https://cloud.google.com/kubernetes-engine/docs/how-to/iam-integration#using_legacy_cluster_certificate_or_user_credentials) is enabled by default in Google Kubernetes Engine. For enhanced authentication security, you should ensure that you have disabled Basic Authentication (https://kubernetes.io/docs/admin/authentication/#static-password-file) by setting an empty username and password for the MasterAuth (https://cloud.google.com/kubernetes-engine/docs/reference/rest/v1/projects.zones.clusters#setmasterauth) configuration. In the same configuration, you can also disable the client certificate (https://cloud.google.com/kubernetes-engine/docs/reference/rest/v1beta1/projects.locations.clusters#clientcertificateconfig) which ensures that you have one less key to think about when locking down access to your cluster.

Another way to help secure your Kubernetes master is to ensure that you are doing credential rotation (https://cloud.google.com/kubernetes-engine/docs/how-to/credential-rotation) on a regular basis. When credential rotation is initiated, the SSL certificates and cluster certificate authority are rotated. This process is automated by Google Kubernetes Engine and also ensures that your master IP address rotates.

For more information:

- Read the Role-Based Access Control documentation (https://cloud.google.com/kubernetes-engine/docs/how-to/role-based-access-control)
- Follow the Credential Rotation guide (https://cloud.google.com/kubernetes-engine/docs/how-to/credential-rotation)

Node security

Google Kubernetes Engine deploys your workloads on Compute Engine instances running in your GCP project. These instances are attached to your Google Kubernetes Engine cluster as nodes (https://kubernetes.io/docs/concepts/architecture/nodes/). The following sections show you how leverage the node-level security features available to you in GCP.

Container-Optimized OS

By default, Google Kubernetes Engine nodes (https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture#nodes) use Google's Container-Optimized OS (https://cloud.google.com/container-optimized-os/docs/concepts/features-and-benefits) as the operating system on which to run Kubernetes and its components. Container-Optimized OS implements several advanced features (https://cloud.google.com/container-optimized-os/docs/concepts/security) for enhancing the security of Google Kubernetes Engine clusters, including:

- Locked-down firewall
- Read-only filesystem where possible
- Limited user accounts and disabled root login

Node upgrades

A best practice is to patch your OS on a regular basis. From time to time, security issues in the container runtime, Kubernetes itself, or the node operating system might require you to upgrade your nodes more urgently. When you upgrade your node, the node's software is upgraded to their latest versions.

You can manually upgrade (https://cloud.google.com/kubernetes-engine/docs/how-to/upgrading-a-container-cluster) the nodes in your cluster, but Google Kubernetes Engine also allows you to enable automatic upgrades (https://cloud.google.com/kubernetes-engine/docs/concepts/node-auto-upgrades).

Securing instance metadata

Google Kubernetes Engine nodes run as Compute Engine instances, and as such they have access to instance metadata (https://cloud.google.com/compute/docs/storing-retrieving-metadata) by default.

Instance metadata is used to provide nodes with credentials and configurations used in bootstrapping and connecting to the Kubernetes master nodes. However, a Pod running on a node does not necessarily need this information, which contains sensitive data, like the node's service account key. You can lock down sensitive instance metadata paths by disabling legacy APIs and by using metadata concealment (https://cloud.google.com/kubernetes-engine/docs/how-to/protecting-cluster-metadata#concealment). Metadata concealment ensures that Pods running in your cluster are not able to access sensitive data by filtering requests to fields such as the `kube-env`.

Network security

Most workloads running in Google Kubernetes Engine need to communicate with other services that could be running either inside or outside of the cluster. You can use several different methods to control what traffic is allowed to flow through your clusters and their Pods.

Limiting Pod-to-Pod communication

By default, all Pods in a cluster can be reached over the network via their Pod IP address. Similarly, by default, egress traffic allows outbound connections to any address accessible in the VPC into which the cluster was deployed.

Cluster administrators and users can lock down the ingress and egress connections created to and from the Pods in a namespace by using network policies (https://kubernetes.io/docs/concepts/services-networking/network-policies/). By default, when there are no network policies defined, all ingress and egress traffic is allowed to flow into and out of all Pods. Network policies allow you to use tags to define the traffic flowing through your Pods.

Once a network policy is applied in a namespace, all traffic is dropped to and from Pods that don't match the configured labels. As part of your creation of clusters and/or namespaces, you can apply the default deny traffic to both ingress and egress (https://kubernetes.io/docs/concepts/services-networking/network-policies/#default-deny-all-ingress-and-all-egress-traffic) of every Pod to ensure that all new workloads added to the cluster must explicitly authorize the traffic they require.

For more information:

- Read more about network policies

(https://cloud.google.com/kubernetes-engine/docs/how-to/network-policy)

- Follow the network policy tutorial (https://cloud.google.com/kubernetes-engine/docs/tutorials/network-policy)
- Read more about default policies (https://kubernetes.io/docs/concepts/services-networking/network-policies/#default-policies)

Filtering load balanced traffic

To load balance your Kubernetes Pods with a network load balancer (https://cloud.google.com/compute/docs/load-balancing/network/), you need to create a Service of type `LoadBalancer` that matches your Pod's labels. With the Service created, you will have an external-facing IP that maps to ports on your Kubernetes Pods. Filtering authorized traffic is achieved at the node level by kube-proxy (https://kubernetes.io/docs/reference/generated/kube-proxy/), which filters based on IP address.

To configure this filtering, you can use the `loadBalancerSourceRanges` configuration of the Service object. With this configuration parameter, you can provide a list of CIDR ranges that you would like to whitelist for access to the Service. If you do not configure `loadBalancerSourceRanges`, all addresses are allowed to access the Service via its external IP.

For cases in which access to the Service is not required, consider using an internal load balancer (https://cloud.google.com/compute/docs/load-balancing/internal/). The internal load balancer also respects the `loadBalancerSourceRanges` when it is necessary to filter out traffic from inside of the VPC.

For more information:

- Follow the internal load balancing tutorial (https://cloud.google.com/kubernetes-engine/docs/how-to/internal-load-balancing)
- Read more about loadBalancerSourceRanges (https://kubernetes.io/docs/tasks/access-application-cluster/configure-cloud-provider-firewall/#restrict-access-for-loadbalancer-service)

Securing your workloads

Kubernetes allows users to quickly provision, scale, and update container-based workloads. This section describes tactics that administrators and users can use to limit the abilities of the running containers to affect other containers in the cluster, the hosts on which they run, and the GCP services enabled in their project.

Limiting Pod container process privileges

Limiting the privileges of containerized processes is important for the overall security of your cluster. Google Kubernetes Engine allows you to set security-related options via the Security Context (https://kubernetes.io/docs/tasks/configure-pod-container/security-context/) on both Pods and containers. These settings allow you to change security settings of your processes like:

- User and group to run as
- Available Linux capabilities
- Ability to escalate privileges

In order to change these settings at the cluster level rather than at the Pod or container, you need to implement a PodSecurityPolicy (https://cloud.google.com/kubernetes-engine/docs/how-to/pod-security-policies). Cluster administrators can use PodSecurityPolicies to ensure that all Pods in a cluster adhere to a minimum baseline policy that you define.

The Google Kubernetes Engine node operating systems, both Container-Optimized OS and Ubuntu, apply the default Docker AppArmor security policies (https://cloud.google.com/container-optimized-os/docs/how-to/secure-apparmor#using_the_default_docker_apparmor_security_profile) to all containers started by Kubernetes. You can view the profile's template on GitHub (https://github.com/moby/moby/blob/master/profiles/apparmor/template.go). Among other things, the profile **denies** the following abilities to containers:

- Write files directly in /proc/
- Write to files that are not in a process ID directory (/proc/)
- Write to files in /proc/sys other than /proc/sys/kernel/shm*
- Mount filesystems

For more information:

- Read the Pod Security Context documentation (https://kubernetes.io/docs/tasks/configure-pod-container/security-context/)
- Read the Pod Security Policy documentation

(https://cloud.google.com/kubernetes-engine/docs/how-to/pod-security-policies)

- Learn more about existing protections in the Container-Optimized OS AppArmor documentation (https://cloud.google.com/container-optimized-os/docs/how-to/secure-apparmor)

Giving Pods access to GCP resources

Your containers and Pods might need access to other resources in GCP. One way for applications to get access to credentials for GCP resources is to use the Kubernetes clusters' service account (https://cloud.google.com/compute/docs/access/service-accounts) credentials from metadata. However, these credentials can be reached by any Pods running on the cluster. You should create and configure a custom service account for your clusters that has the minimum IAM roles (https://cloud.google.com/iam/docs/understanding-roles) that should apply to all the Pods running in the cluster.

Another way to provide application credentials is by using a service account JSON key (https://cloud.google.com/iam/docs/creating-managing-service-account-keys). You should use application-specific GCP service accounts to provide credentials (https://cloud.google.com/kubernetes-engine/docs/tutorials/authenticating-to-cloud-platform) so that applications have the minimal necessary permissions. Each service account is assigned only the Cloud IAM roles that are needed for its paired application to operate successfully. Keeping the service account application-specific makes it easier to revoke its access in the case of a compromise without affecting other applications. Once you have assigned your service account the correct Cloud IAM roles, you can create a JSON service account key, and then mount that into your Pod using a Kubernetes Secret (https://cloud.google.com/kubernetes-engine/docs/concepts/secret).

For more information:

- Follow the Authenticating to Cloud Platform with Service Accounts tutorial (https://cloud.google.com/kubernetes-engine/docs/tutorials/authenticating-to-cloud-platform)
- Read more about GCP Service Accounts (https://cloud.google.com/iam/docs/understanding-service-accounts)
- Read the Google Kubernetes Engine secrets documentation (https://cloud.google.com/kubernetes-engine/docs/concepts/secret)

Using Binary Authorization

Beta

This is a Beta release of Binary Authorization. This feature is not covered by any SLA or deprecation policy and might be subject to backward-incompatible changes.

Binary Authorization (https://cloud.google.com/binary-authorization/docs/overview) is a service on GCP that provides software supply-chain security for applications that run in the Cloud. Binary Authorization works with images that you deploy to GKE from Container Registry or another container image registry. With Binary Authorization, you can ensure that internal processes that safeguard the quality and integrity of your software have successfully completed before an application is deployed to your production environment.

For instructions about creating a cluster with Binary Authorization enabled, visit Creating a Cluster (https://cloud.google.com/binary-authorization/docs/creating-cluster) in the Binary Authorization documentation (https://cloud.google.com/binary-authorization/docs/).

Audit logging

Audit logging provides a way for administrators to retain, query, process, and alert on events that occur in your Google Kubernetes Engine environments. Administrators can use the logged information to do forensic analysis, real-time alerting, or for cataloging how a fleet of Google Kubernetes Engine clusters are being used and by whom.

By default, Google Kubernetes Engine logs Admin Activity logs. You can optionally also log Data Access events, depending on the types of operations you are interested in inspecting.

For more information:

- Follow the Google Kubernetes Engine audit logging tutorial (https://cloud.google.com/kubernetes-engine/docs/how-to/audit-logging)
- Read more about Cloud Audit Logging (https://cloud.google.com/logging/docs/audit/)

Dernière mise à jour : décembre 18, 2018