roduct or feature is in a pre-release state and might change or have limited support. For more information, see the product launch stages lucts/#product-launch-stages).

This page explains how to use PodSecurityPolicies in Google Kubernetes Engine.

A **PodSecurityPolicy** is an admission controller resource you create that validates requests to create and update Pods (/kubernetes-engine/docs/concepts/pod) on your cluster (/kubernetes-engine/docs/concepts/cluster-architecture). The PodSecurityPolicy defines a set of conditions that Pods must meet to be accepted by the cluster; when a request to create or update a Pod does not meet the conditions in the PodSecurityPolicy, that request is rejected and an error is returned.

To use PodSecurityPolicy, you must first create and define policies that new and updated Pods must meet. Then, you must enable the PodSecurityPolicy (https://kubernetes.io/docs/concepts/policy/pod-security-policy/) admission controller, which validates requests to create and update Pods against the defined policies.

When multiple PodSecurityPolicies are available, the admission controller uses the first policy that successfully validates. Policies are ordered alphabetically, and the controller prefers non-mutating policies (policies that don't change the Pod) over mutating policies.

PodSecurityPolicy is available in GKE clusters running Kubernetes version 1.8.6 or later.

Before you start, make sure you have performed the following tasks:

- Ensure that you have enabled the Google Kubernetes Engine API.

  Enable Google Kubernetes Engine API (https://console.cloud.google.com/apis/library/container.googleapis.com?q=kubernetes%20engine)
- Ensure that you have installed the Cloud SDK (/sdk/downloads).

Set up default `gcloud` settings using one of the following methods:

- Using `gcloud init`, if you want to be walked through setting defaults.
- Using `gcloud config`, to individually set your project ID, zone, and region.

- Ensure that you understand how to use role-based access control (/kubernetes-engine/docs/how-to/role-based-access-control) in GKE.

You need to define PodSecurityPolicy resources in your cluster before the PodSecurityPolicy controller can validate and accept Pods into the cluster.

PodSecurityPolicies specify a list of restrictions, requirements, and defaults for Pods created under the policy. Examples include restricting the use of privileged containers, `hostPath` volumes, and host networking, or defaulting all containers to run with a seccomp profile. The PodSecurityPolicy admission controller validates requests against available PodSecurityPolicies.

The following example PodSecurityPolicy, `my-psp.yaml`, simply prevents the creation of privileged Pods. The policy also affects several other control aspects, such as allowing access to all available volumes:

`privileged: false` is not sufficient for preventing Pods from *performing* privileged operations. For more comprehensive restrictions, refer to the le restricted policy in Example Policies (https://kubernetes.io/docs/concepts/policy/pod-security-policy/#example-policies).

The PodSecurityPolicy specification (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#podsecuritypolicy-v1beta1-extensions) can secure numerous control aspects. The control aspects specified in this example—`seLinux`, `supplementalGroups`, `runAsUser`, and `fsGroup`—are all set to `RunAsAny`, indicating that any valid values for these fields can be used with this policy.

For more information about PodSecurityPolicies and their control aspects, refer to What is a Pod Security Policy? (https://kubernetes.io/docs/concepts/policy/pod-security-policy/#what-is-a-pod-security-policy) and the policy reference (https://kubernetes.io/docs/concepts/policy/pod-security-policy/#policy-reference) in the Kubernetes documentation.

You create this resource using the `kubectl` command-line tool:

For more examples of configuring PodSecurityPolicies, refer to Example (https://kubernetes.io/docs/concepts/policy/pod-security-policy/#example) on the PodSecurityPolicy page of the Kubernetes documentation.

Accounts with the ***cluster-admin*** (/kubernetes-engine/docs/how-to/role-based-access-control#before_you_begin) role can use role-based access control (/kubernetes-engine/docs/how-to/role-based-access-control) to create a Role (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#role-v1-rbac-authorization-k8s-io/) or ClusterRole (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#clusterrole-v1-rbac-authorization-k8s-io) that grants the desired service accounts access to PodSecurityPolicies. A ClusterRole grants *cluster-wide* permissions, and a Role grants permissions within a *namespace* that you define.

For example, the following ClusterRole, **`my-clusterrole.yaml`**, grants access to the **`my-psp`** PodSecurityPolicy, as indicated by **`verb: use`**:

Create the ClusterRole by running the following command:

After creating a Role (or ClusterRole), you associate it with the desired service accounts by creating a RoleBinding (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#rolebinding-v1-rbac-authorization-k8s-io) (or ClusterRoleBinding (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#clusterrolebinding-v1-rbac-authorization-k8s-io)) resource.

The following RoleBinding, **`my-rolebinding.yaml`**, binds the ClusterRole, **`my-clusterrole`**, to the service accounts in a specific namespace, **`my-namespace`**:

In this RoleBinding:

- The `subjects` field specifies to which accounts the ClusterRole is bound.
- The first subject is a Group, `system:serviceaccounts`, which encompasses all service accounts in the cluster.
- The second subject is an individual ServiceAccount, `default`, which specifies the default service account in the namespace.

Create the RoleBinding by running the following command:

For more information about RBAC, refer to Using RBAC Authorization (https://kubernetes.io/docs/admin/authorization/rbac/).

To use the PodSecurityPolicy admission controller, you must create a new cluster or update an existing cluster with the `--enable-pod-security-policy` flag.

ng: If you enable the PodSecurityPolicy controller without first defining and authorizing any actual policies, no users, controllers, or service accounts
or update Pods. If you are working with an existing cluster, you should define (#define_policies) and authorize (#authorize_policies) policies *before*
ng the controller.

To create a new cluster with PodSecurityPolicy, run the following command:

To update an existing cluster:

You disable the PodSecurityPolicy controller by running the following command:

Disabling the controller causes the cluster to stop validating and defaulting existing policies, but does not delete them. Bindings are also not deleted.

If you are using a NetworkPolicy (/kubernetes-engine/docs/how-to/network-policy), and you have a Pod that is subject to a PodSecurityPolicy, create an RBAC Role or ClusterRole (https://kubernetes.io/docs/reference/access-authn-authz/rbac/) that has permission to use the PodSecurityPolicy. Then bind the Role or ClusterRole to the Pod's service account. Granting permissions to user accounts is not sufficient in this case. For more information, see Authorizing policies (/kubernetes-engine/docs/how-to/pod-security-policies#authorize_policies).

- Learn more about role-based access control (/kubernetes-engine/docs/how-to/role-based-access-control).
- Learn about securing your cluster using identity and access management (/kubernetes-engine/docs/how-to/iam-integration).
- Learn about Sandbox Pods (/kubernetes-engine/docs/how-to/sandbox-pods)