

# Reference

- 1: [Glossary](#)
- 2: [API Overview](#)
  - 2.1: [Kubernetes API Concepts](#)
  - 2.2: [Server-Side Apply](#)
  - 2.3: [Client Libraries](#)
  - 2.4: [Common Expression Language in Kubernetes](#)
  - 2.5: [Kubernetes Deprecation Policy](#)
  - 2.6: [Deprecated API Migration Guide](#)
  - 2.7: [Kubernetes API health endpoints](#)
- 3: [API Access Control](#)
  - 3.1: [Authenticating](#)
  - 3.2: [Authenticating with Bootstrap Tokens](#)
  - 3.3: [Authorization](#)
  - 3.4: [Using RBAC Authorization](#)
  - 3.5: [Using Node Authorization](#)
  - 3.6: [Webhook Mode](#)
  - 3.7: [Using ABAC Authorization](#)
  - 3.8: [Admission Controllers Reference](#)
  - 3.9: [Dynamic Admission Control](#)
  - 3.10: [Managing Service Accounts](#)
  - 3.11: [Certificates and Certificate Signing Requests](#)
  - 3.12: [Mapping PodSecurityPolicies to Pod Security Standards](#)
  - 3.13: [Kubelet authentication/authorization](#)
  - 3.14: [TLS bootstrapping](#)
  - 3.15: [Validating Admission Policy](#)
- 4: [Well-Known Labels, Annotations and Taints](#)
  - 4.1: [Audit Annotations](#)
- 5: [Kubernetes API](#)
  - 5.1: [Workload Resources](#)
    - 5.1.1: [Pod](#)
    - 5.1.2: [PodTemplate](#)
    - 5.1.3: [ReplicationController](#)
    - 5.1.4: [ReplicaSet](#)
    - 5.1.5: [Deployment](#)
    - 5.1.6: [StatefulSet](#)
    - 5.1.7: [ControllerRevision](#)
    - 5.1.8: [DaemonSet](#)
    - 5.1.9: [Job](#)
    - 5.1.10: [CronJob](#)
    - 5.1.11: [HorizontalPodAutoscaler](#)
    - 5.1.12: [HorizontalPodAutoscaler](#)
    - 5.1.13: [PriorityClass](#)
    - 5.1.14: [PodSchedulingContext v1alpha2](#)
    - 5.1.15: [ResourceClaim v1alpha2](#)
    - 5.1.16: [ResourceClaimTemplate v1alpha2](#)
    - 5.1.17: [ResourceClass v1alpha2](#)
  - 5.2: [Service Resources](#)

- 5.2.1: [Service](#)
- 5.2.2: [Endpoints](#)
- 5.2.3: [EndpointSlice](#)
- 5.2.4: [Ingress](#)
- 5.2.5: [IngressClass](#)
- 5.3: [Config and Storage Resources](#)
  - 5.3.1: [ConfigMap](#)
  - 5.3.2: [Secret](#)
  - 5.3.3: [Volume](#)
  - 5.3.4: [PersistentVolumeClaim](#)
  - 5.3.5: [PersistentVolume](#)
  - 5.3.6: [StorageClass](#)
  - 5.3.7: [VolumeAttachment](#)
  - 5.3.8: [CSIDriver](#)
  - 5.3.9: [CSINode](#)
  - 5.3.10: [CSIStorageCapacity](#)
- 5.4: [Authentication Resources](#)
  - 5.4.1: [ServiceAccount](#)
  - 5.4.2: [TokenRequest](#)
  - 5.4.3: [TokenReview](#)
  - 5.4.4: [CertificateSigningRequest](#)
  - 5.4.5: [ClusterTrustBundle v1alpha1](#)
  - 5.4.6: [SelfSubjectReview](#)
- 5.5: [Authorization Resources](#)
  - 5.5.1: [LocalSubjectAccessReview](#)
  - 5.5.2: [SelfSubjectAccessReview](#)
  - 5.5.3: [SelfSubjectRulesReview](#)
  - 5.5.4: [SubjectAccessReview](#)
  - 5.5.5: [ClusterRole](#)
  - 5.5.6: [ClusterRoleBinding](#)
  - 5.5.7: [Role](#)
  - 5.5.8: [RoleBinding](#)
- 5.6: [Policy Resources](#)
  - 5.6.1: [LimitRange](#)
  - 5.6.2: [ResourceQuota](#)
  - 5.6.3: [NetworkPolicy](#)
  - 5.6.4: [PodDisruptionBudget](#)
  - 5.6.5: [IPAddress v1alpha1](#)
- 5.7: [Extend Resources](#)
  - 5.7.1: [CustomResourceDefinition](#)
  - 5.7.2: [MutatingWebhookConfiguration](#)
  - 5.7.3: [ValidatingWebhookConfiguration](#)
  - 5.7.4: [ValidatingAdmissionPolicy v1beta1](#)
- 5.8: [Cluster Resources](#)
  - 5.8.1: [Node](#)
  - 5.8.2: [Namespace](#)
  - 5.8.3: [Event](#)
  - 5.8.4: [APIService](#)
  - 5.8.5: [Lease](#)
  - 5.8.6: [LeaseCandidate](#)
  - 5.8.7: [RuntimeClass](#)
  - 5.8.8: [FlowSchema v1beta3](#)

- 5.8.9: [PriorityLevelConfiguration v1beta3](#)
- 5.8.10: [Binding](#)
- 5.8.11: [ComponentStatus](#)
- 5.8.12: [ClusterCIDR v1alpha1](#)
- 5.9: [Common Definitions](#)
  - 5.9.1: [DeleteOptions](#)
  - 5.9.2: [LabelSelector](#)
  - 5.9.3: [ListMeta](#)
  - 5.9.4: [LocalObjectReference](#)
  - 5.9.5: [NodeSelectorRequirement](#)
  - 5.9.6: [ObjectFieldSelector](#)
  - 5.9.7: [ObjectMeta](#)
  - 5.9.8: [ObjectReference](#)
  - 5.9.9: [Patch](#)
  - 5.9.10: [Quantity](#)
  - 5.9.11: [ResourceFieldSelector](#)
  - 5.9.12: [Status](#)
  - 5.9.13: [TypedLocalObjectReference](#)
- 5.10: [Other Resources](#)
  - 5.10.1: [ValidatingAdmissionPolicyBindingList v1beta1](#)
- 5.11: [Common Parameters](#)
- 6: [Instrumentation](#)
  - 6.1: [Kubernetes Component SLI Metrics](#)
  - 6.2: [CRI Pod & Container Metrics](#)
  - 6.3: [Node metrics data](#)
  - 6.4: [Kubernetes Metrics Reference](#)
- 7: [Kubernetes Issues and Security](#)
  - 7.1: [Kubernetes Issue Tracker](#)
  - 7.2: [Kubernetes Security and Disclosure Information](#)
  - 7.3: [Official CVE Feed](#)
- 8: [Node Reference Information](#)
  - 8.1: [Kubelet Checkpoint API](#)
  - 8.2: [Articles on dockershim Removal and on Using CRI-compatible Runtimes](#)
  - 8.3: [Node Labels Populated By The Kubelet](#)
  - 8.4: [Kubelet Configuration Directory Merging](#)
  - 8.5: [Kubelet Device Manager API Versions](#)
  - 8.6: [Node Status](#)
- 9: [Networking Reference](#)
  - 9.1: [Protocols for Services](#)
  - 9.2: [Ports and Protocols](#)
  - 9.3: [Virtual IPs and Service Proxies](#)
- 10: [Setup tools](#)
  - 10.1: [Kubeadm](#)
    - 10.1.1: [Kubeadm Generated](#)
      - 10.1.1.1:
      - 10.1.1.2:
        - 10.1.1.2.1:
        - 10.1.1.2.2:
        - 10.1.1.2.3:
        - 10.1.1.2.4:
        - 10.1.1.2.5:

- 10.1.1.2.6:
- 10.1.1.2.7:
- 10.1.1.2.8:
- 10.1.1.2.9:
- 10.1.1.2.10:
- 10.1.1.2.11:
- 10.1.1.2.12:
- 10.1.1.2.13:
- 10.1.1.2.14:
- 10.1.1.2.15:
- 10.1.1.2.16:
- 10.1.1.3:
- 10.1.1.4:
  - 10.1.1.4.1:
  - 10.1.1.4.2:
  - 10.1.1.4.3:
  - 10.1.1.4.4:
  - 10.1.1.4.5:
  - 10.1.1.4.6:
  - 10.1.1.4.7:
  - 10.1.1.4.8:
  - 10.1.1.4.9:
  - 10.1.1.4.10:
- 10.1.1.5:
  - 10.1.1.5.1:
  - 10.1.1.5.2:
  - 10.1.1.5.3:
  - 10.1.1.5.4:
  - 10.1.1.5.5:
  - 10.1.1.5.6:
  - 10.1.1.5.7:
  - 10.1.1.5.8:
  - 10.1.1.5.9:
  - 10.1.1.5.10:
  - 10.1.1.5.11:
  - 10.1.1.5.12:
  - 10.1.1.5.13:
  - 10.1.1.5.14:
  - 10.1.1.5.15:
  - 10.1.1.5.16:
  - 10.1.1.5.17:
  - 10.1.1.5.18:
  - 10.1.1.5.19:
  - 10.1.1.5.20:
  - 10.1.1.5.21:
  - 10.1.1.5.22:
  - 10.1.1.5.23:
  - 10.1.1.5.24:
  - 10.1.1.5.25:
  - 10.1.1.5.26:
  - 10.1.1.5.27:
  - 10.1.1.5.28:

- 10.1.1.5.29:
- 10.1.1.5.30:
- 10.1.1.5.31:
- 10.1.1.5.32:
- 10.1.1.5.33:
- 10.1.1.5.34:
- 10.1.1.5.35:
- 10.1.1.5.36:
- 10.1.1.5.37:
- 10.1.1.5.38:
- 10.1.1.5.39:
- 10.1.1.5.40:
- 10.1.1.5.41:
- 10.1.1.5.42:
- 10.1.1.5.43:
- 10.1.1.5.44:
- 10.1.1.5.45:
- 10.1.1.5.46:
- 10.1.1.6:
  - 10.1.1.6.1:
  - 10.1.1.6.2:
  - 10.1.1.6.3:
  - 10.1.1.6.4:
  - 10.1.1.6.5:
  - 10.1.1.6.6:
  - 10.1.1.6.7:
  - 10.1.1.6.8:
  - 10.1.1.6.9:
  - 10.1.1.6.10:
  - 10.1.1.6.11:
  - 10.1.1.6.12:
  - 10.1.1.6.13:
  - 10.1.1.6.14:
- 10.1.1.7:
  - 10.1.1.7.1:
- 10.1.1.8:
  - 10.1.1.8.1:
  - 10.1.1.8.2:
  - 10.1.1.8.3:
  - 10.1.1.8.4:
- 10.1.1.9:
  - 10.1.1.9.1:
  - 10.1.1.9.2:
  - 10.1.1.9.3:
  - 10.1.1.9.4:
- 10.1.1.10:
  - 10.1.1.10.1:
  - 10.1.1.10.2:
  - 10.1.1.10.3:
  - 10.1.1.10.4:
  - 10.1.1.10.5:
  - 10.1.1.10.6:

- 10.1.1.10.7:
    - 10.1.1.10.8:
      - 10.1.1.11:
      - 10.1.1.12:
  - 10.1.2: [kubeadm init](#)
  - 10.1.3: [kubeadm join](#)
  - 10.1.4: [kubeadm upgrade](#)
  - 10.1.5: [kubeadm config](#)
  - 10.1.6: [kubeadm reset](#)
  - 10.1.7: [kubeadm token](#)
  - 10.1.8: [kubeadm version](#)
  - 10.1.9: [kubeadm alpha](#)
  - 10.1.10: [kubeadm certs](#)
  - 10.1.11: [kubeadm init phase](#)
  - 10.1.12: [kubeadm join phase](#)
  - 10.1.13: [kubeadm kubeconfig](#)
  - 10.1.14: [kubeadm reset phase](#)
  - 10.1.15: [kubeadm upgrade phase](#)
  - 10.1.16: [Implementation details](#)
- 11: [Command line tool \(kubectl\)](#)
    - 11.1: [Introduction to kubectl](#)
    - 11.2: [kubectl Quick Reference](#)
    - 11.3: [kubectl reference](#)
      - 11.3.1: [kubectl](#)
      - 11.3.2: [kubectl annotate](#)
      - 11.3.3: [kubectl api-resources](#)
      - 11.3.4: [kubectl api-versions](#)
      - 11.3.5: [kubectl apply](#)
        - 11.3.5.1: [kubectl apply edit-last-applied](#)
        - 11.3.5.2: [kubectl apply set-last-applied](#)
        - 11.3.5.3: [kubectl apply view-last-applied](#)
      - 11.3.6: [kubectl attach](#)
      - 11.3.7: [kubectl auth](#)
        - 11.3.7.1: [kubectl auth can-i](#)
        - 11.3.7.2: [kubectl auth reconcile](#)
        - 11.3.7.3: [kubectl auth whoami](#)
      - 11.3.8: [kubectl autoscale](#)
      - 11.3.9: [kubectl certificate](#)
        - 11.3.9.1: [kubectl certificate approve](#)
        - 11.3.9.2: [kubectl certificate deny](#)
      - 11.3.10: [kubectl cluster-info](#)
        - 11.3.10.1: [kubectl cluster-info dump](#)
      - 11.3.11: [kubectl completion](#)
      - 11.3.12: [kubectl config](#)
        - 11.3.12.1: [kubectl config current-context](#)
        - 11.3.12.2: [kubectl config delete-cluster](#)
        - 11.3.12.3: [kubectl config delete-context](#)
        - 11.3.12.4: [kubectl config delete-user](#)
        - 11.3.12.5: [kubectl config get-clusters](#)
        - 11.3.12.6: [kubectl config get-contexts](#)
        - 11.3.12.7: [kubectl config get-users](#)
        - 11.3.12.8: [kubectl config rename-context](#)

- 11.3.12.9: [kubectl config set](#)
- 11.3.12.10: [kubectl config set-cluster](#)
- 11.3.12.11: [kubectl config set-context](#)
- 11.3.12.12: [kubectl config set-credentials](#)
- 11.3.12.13: [kubectl config unset](#)
- 11.3.12.14: [kubectl config use-context](#)
- 11.3.12.15: [kubectl config view](#)
- 11.3.13: [kubectl cordon](#)
- 11.3.14: [kubectl cp](#)
- 11.3.15: [kubectl create](#)
  - 11.3.15.1: [kubectl create clusterrole](#)
  - 11.3.15.2: [kubectl create clusterrolebinding](#)
  - 11.3.15.3: [kubectl create configmap](#)
  - 11.3.15.4: [kubectl create cronjob](#)
  - 11.3.15.5: [kubectl create deployment](#)
  - 11.3.15.6: [kubectl create ingress](#)
  - 11.3.15.7: [kubectl create job](#)
  - 11.3.15.8: [kubectl create namespace](#)
  - 11.3.15.9: [kubectl create poddisruptionbudget](#)
  - 11.3.15.10: [kubectl create priorityclass](#)
  - 11.3.15.11: [kubectl create quota](#)
  - 11.3.15.12: [kubectl create role](#)
  - 11.3.15.13: [kubectl create rolebinding](#)
  - 11.3.15.14: [kubectl create secret](#)
  - 11.3.15.15: [kubectl create secret docker-registry](#)
  - 11.3.15.16: [kubectl create secret generic](#)
  - 11.3.15.17: [kubectl create secret tls](#)
  - 11.3.15.18: [kubectl create service](#)
  - 11.3.15.19: [kubectl create service clusterip](#)
  - 11.3.15.20: [kubectl create service externalname](#)
  - 11.3.15.21: [kubectl create service loadbalancer](#)
  - 11.3.15.22: [kubectl create service nodeport](#)
  - 11.3.15.23: [kubectl create serviceaccount](#)
  - 11.3.15.24: [kubectl create token](#)
- 11.3.16: [kubectl debug](#)
- 11.3.17: [kubectl delete](#)
- 11.3.18: [kubectl describe](#)
- 11.3.19: [kubectl diff](#)
- 11.3.20: [kubectl drain](#)
- 11.3.21: [kubectl edit](#)
- 11.3.22: [kubectl events](#)
- 11.3.23: [kubectl exec](#)
- 11.3.24: [kubectl explain](#)
- 11.3.25: [kubectl expose](#)
- 11.3.26: [kubectl get](#)
- 11.3.27: [kubectl kustomize](#)
- 11.3.28: [kubectl label](#)
- 11.3.29: [kubectl logs](#)
- 11.3.30: [kubectl options](#)
- 11.3.31: [kubectl patch](#)
- 11.3.32: [kubectl plugin](#)
  - 11.3.32.1: [kubectl plugin list](#)

- 11.3.33: [kubectl port-forward](#)
  - 11.3.34: [kubectl proxy](#)
  - 11.3.35: [kubectl replace](#)
  - 11.3.36: [kubectl rollout](#)
    - 11.3.36.1: [kubectl rollout history](#)
    - 11.3.36.2: [kubectl rollout pause](#)
    - 11.3.36.3: [kubectl rollout restart](#)
    - 11.3.36.4: [kubectl rollout resume](#)
    - 11.3.36.5: [kubectl rollout status](#)
    - 11.3.36.6: [kubectl rollout undo](#)
  - 11.3.37: [kubectl run](#)
  - 11.3.38: [kubectl scale](#)
  - 11.3.39: [kubectl set](#)
    - 11.3.39.1: [kubectl set env](#)
    - 11.3.39.2: [kubectl set image](#)
    - 11.3.39.3: [kubectl set resources](#)
    - 11.3.39.4: [kubectl set selector](#)
    - 11.3.39.5: [kubectl set serviceaccount](#)
    - 11.3.39.6: [kubectl set subject](#)
  - 11.3.40: [kubectl taint](#)
  - 11.3.41: [kubectl top](#)
    - 11.3.41.1: [kubectl top node](#)
    - 11.3.41.2: [kubectl top pod](#)
  - 11.3.42: [kubectl uncordon](#)
  - 11.3.43: [kubectl version](#)
  - 11.3.44: [kubectl wait](#)
  - 11.4: [kubectl Commands](#)
  - 11.5: [kubectl](#)
  - 11.6: [JSONPath Support](#)
  - 11.7: [kubectl for Docker Users](#)
  - 11.8: [kubectl Usage Conventions](#)
- 12: [Component tools](#)
    - 12.1: [Feature Gates](#)
    - 12.2: [Feature Gates \(removed\)](#)
    - 12.3: [kubelet](#)
    - 12.4: [kube-apiserver](#)
    - 12.5: [kube-controller-manager](#)
    - 12.6: [kube-proxy](#)
    - 12.7: [kube-scheduler](#)
  - 13: [Debug cluster](#)
    - 13.1: [Flow control](#)
  - 14: [Configuration APIs](#)
    - 14.1: [Client Authentication \(v1\)](#)
    - 14.2: [Client Authentication \(v1beta1\)](#)
    - 14.3: [Event Rate Limit Configuration \(v1alpha1\)](#)
    - 14.4: [Image Policy API \(v1alpha1\)](#)
    - 14.5: [kube-apiserver Admission \(v1\)](#)
    - 14.6: [kube-apiserver Audit Configuration \(v1\)](#)
    - 14.7: [kube-apiserver Configuration \(v1\)](#)
    - 14.8: [kube-apiserver Configuration \(v1alpha1\)](#)
    - 14.9: [kube-apiserver Configuration \(v1beta1\)](#)
    - 14.10: [kube-controller-manager Configuration \(v1alpha1\)](#)

- 14.11: [kube-proxy Configuration \(v1alpha1\)](#)
- 14.12: [kube-scheduler Configuration \(v1\)](#)
- 14.13: [kubeadm Configuration \(v1beta3\)](#)
- 14.14: [kubeadm Configuration \(v1beta4\)](#)
- 14.15: [kubeconfig \(v1\)](#)
- 14.16: [Kubelet Configuration \(v1\)](#)
- 14.17: [Kubelet Configuration \(v1alpha1\)](#)
- 14.18: [Kubelet Configuration \(v1beta1\)](#)
- 14.19: [Kubelet CredentialProvider \(v1\)](#)
- 14.20: [WebhookAdmission Configuration \(v1\)](#)
- 15: [External APIs](#)
  - 15.1: [Kubernetes Custom Metrics \(v1beta2\)](#)
  - 15.2: [Kubernetes External Metrics \(v1beta1\)](#)
  - 15.3: [Kubernetes Metrics \(v1beta1\)](#)
- 16: [Scheduling](#)
  - 16.1: [Scheduler Configuration](#)
  - 16.2: [Scheduling Policies](#)
- 17: [Other Tools](#)
  - 17.1: [Mapping from dockercli to cubectl](#)

This section of the Kubernetes documentation contains references.

## API Reference

- [Glossary](#) - a comprehensive, standardized list of Kubernetes terminology
- [Kubernetes API Reference](#)
- [One-page API Reference for Kubernetes v1.31](#)
- [Using The Kubernetes API](#) - overview of the API for Kubernetes.
- [API access control](#) - details on how Kubernetes controls API access
- [Well-Known Labels, Annotations and Taints](#)

## Officially supported client libraries

To call the Kubernetes API from a programming language, you can use [client libraries](#).

Officially supported client libraries:

- [Kubernetes Go client library](#)
- [Kubernetes Python client library](#)
- [Kubernetes Java client library](#)
- [Kubernetes JavaScript client library](#)
- [Kubernetes C# client library](#)
- [Kubernetes Haskell client library](#)

## CLI

- [kubectl](#) - Main CLI tool for running commands and managing Kubernetes clusters.
  - [JSONPath](#) - Syntax guide for using [JSONPath expressions](#) with kubectl.
- [kubeadm](#) - CLI tool to easily provision a secure Kubernetes cluster.

## Components

- [kubelet](#) - The primary agent that runs on each node. The kubelet takes a set of PodSpecs and ensures that the described containers are running and healthy.
- [kube-apiserver](#) - REST API that validates and configures data for API objects such as pods, services, replication controllers.
- [kube-controller-manager](#) - Daemon that embeds the core control loops shipped with

Kubernetes.

- [kube-proxy](#) - Can do simple TCP/UDP stream forwarding or round-robin TCP/UDP forwarding across a set of back-ends.
- [kube-scheduler](#) - Scheduler that manages availability, performance, and capacity.
  - [Scheduler Policies](#)
  - [Scheduler Profiles](#)
- List of [ports and protocols](#) that should be open on control plane and worker nodes

## Config APIs

This section hosts the documentation for "unpublished" APIs which are used to configure kubernetes components or tools. Most of these APIs are not exposed by the API server in a RESTful way though they are essential for a user or an operator to use or manage a cluster.

- [kubeconfig \(v1\)](#)
- [kube-apiserver admission \(v1\)](#)
- [kube-apiserver configuration \(v1alpha1\)](#) and
- [kube-apiserver configuration \(v1beta1\)](#) and [kube-apiserver configuration \(v1\)](#)
- [kube-apiserver event rate limit \(v1alpha1\)](#)
- [kubelet configuration \(v1alpha1\)](#) and [kubelet configuration \(v1beta1\)](#) [kubelet configuration \(v1\)](#)
- [kubelet credential providers \(v1\)](#)
- [kube-scheduler configuration \(v1beta3\)](#) and [kube-scheduler configuration \(v1\)](#)
- [kube-controller-manager configuration \(v1alpha1\)](#)
- [kube-proxy configuration \(v1alpha1\)](#)
- [audit.k8s.io/v1 API](#)
- [Client authentication API \(v1beta1\)](#) and [Client authentication API \(v1\)](#)
- [WebhookAdmission configuration \(v1\)](#)
- [ImagePolicy API \(v1alpha1\)](#)

## Config API for kubeadm

- [v1beta3](#)
- [v1beta4](#)

## External APIs

These are the APIs defined by the Kubernetes project, but are not implemented by the core project:

- [Metrics API \(v1beta1\)](#)
- [Custom Metrics API \(v1beta2\)](#)
- [External Metrics API \(v1beta1\)](#)

## Design Docs

An archive of the design docs for Kubernetes functionality. Good starting points are [Kubernetes Architecture](#) and [Kubernetes Design Overview](#).

# 1 - Glossary

## 2 - API Overview

This section provides reference information for the Kubernetes API.

The REST API is the fundamental fabric of Kubernetes. All operations and communications between components, and external user commands are REST API calls that the API Server handles. Consequently, everything in the Kubernetes platform is treated as an API object and has a corresponding entry in the [API](#).

The [Kubernetes API reference](#) lists the API for Kubernetes version v1.31.

For general background information, read [The Kubernetes API](#).

[Controlling Access to the Kubernetes API](#) describes how clients can authenticate to the Kubernetes API server, and how their requests are authorized.

## API versioning

The JSON and Protobuf serialization schemas follow the same guidelines for schema changes. The following descriptions cover both formats.

The API versioning and software versioning are indirectly related. The [API and release versioning proposal](#) describes the relationship between API versioning and software versioning.

Different API versions indicate different levels of stability and support. You can find more information about the criteria for each level in the [API Changes documentation](#).

Here's a summary of each level:

- Alpha:
  - The version names contain `alpha` (for example, `v1alpha1` ).
  - Built-in alpha API versions are disabled by default and must be explicitly enabled in the `kube-apiserver` configuration to be used.
  - The software may contain bugs. Enabling a feature may expose bugs.
  - Support for an alpha API may be dropped at any time without notice.
  - The API may change in incompatible ways in a later software release without notice.
  - The software is recommended for use only in short-lived testing clusters, due to increased risk of bugs and lack of long-term support.
- Beta:
  - The version names contain `beta` (for example, `v2beta3` ).
  - Built-in beta API versions are disabled by default and must be explicitly enabled in the `kube-apiserver` configuration to be used (**except** for beta versions of APIs introduced prior to Kubernetes 1.22, which were enabled by default).
  - Built-in beta API versions have a maximum lifetime of 9 months or 3 minor releases (whichever is longer) from introduction to deprecation, and 9 months or 3 minor releases (whichever is longer) from deprecation to removal.
  - The software is well tested. Enabling a feature is considered safe.

- The support for a feature will not be dropped, though the details may change.
- The schema and/or semantics of objects may change in incompatible ways in a subsequent beta or stable API version. When this happens, migration instructions are provided. Adapting to a subsequent beta or stable API version may require editing or re-creating API objects, and may not be straightforward. The migration may require downtime for applications that rely on the feature.
- The software is not recommended for production uses. Subsequent releases may introduce incompatible changes. Use of beta API versions is required to transition to subsequent beta or stable API versions once the beta API version is deprecated and no longer served.

**Note:**

Please try beta features and provide feedback. After the features exit beta, it may not be practical to make more changes.

- Stable:

- The version name is `vx` where `x` is an integer.
- Stable API versions remain available for all future releases within a Kubernetes major version, and there are no current plans for a major version revision of Kubernetes that removes stable APIs.

## API groups

[API groups](#) make it easier to extend the Kubernetes API. The API group is specified in a REST path and in the `apiVersion` field of a serialized object.

There are several API groups in Kubernetes:

- The `core` (also called `legacy`) group is found at REST path `/api/v1`. The `core` group is not specified as part of the `apiVersion` field, for example, `apiVersion: v1`.
- The named groups are at REST path `/apis/$GROUP_NAME/$VERSION` and use `apiVersion: $GROUP_NAME/$VERSION` (for example, `apiVersion: batch/v1`). You can find the full list of supported API groups in [Kubernetes API reference](#).

## Enabling or disabling API groups

Certain resources and API groups are enabled by default. You can enable or disable them by setting `--runtime-config` on the API server. The `--runtime-config` flag accepts comma separated `<key>[=<value>]` pairs describing the runtime configuration of the API server. If the `=<value>` part is omitted, it is treated as if `=true` is specified. For example:

- to disable `batch/v1`, set `--runtime-config=batch/v1=false`
- to enable `batch/v2alpha1`, set `--runtime-config=batch/v2alpha1`
- to enable a specific version of an API, such as `storage.k8s.io/v1beta1/csistoragecapacities`, set `--runtime-config=storage.k8s.io/v1beta1/csistoragecapacities`

**Note:**

When you enable or disable groups or resources, you need to restart the API server and controller manager to pick up the `--runtime-config` changes.

## Persistence

Kubernetes stores its serialized state in terms of the API resources by writing them into etcd.

## What's next

- Learn more about [API conventions](#)
- Read the design documentation for [aggregator](#)

## 2.1 - Kubernetes API Concepts

The Kubernetes API is a resource-based (RESTful) programmatic interface provided via HTTP. It supports retrieving, creating, updating, and deleting primary resources via the standard HTTP verbs (POST, PUT, PATCH, DELETE, GET).

For some resources, the API includes additional subresources that allow fine grained authorization (such as separate views for Pod details and log retrievals), and can accept and serve those resources in different representations for convenience or efficiency.

Kubernetes supports efficient change notifications on resources via *watches*. Kubernetes also provides consistent list operations so that API clients can effectively cache, track, and synchronize the state of resources.

You can view the [API reference](#) online, or read on to learn about the API in general.

## Kubernetes API terminology

Kubernetes generally leverages common RESTful terminology to describe the API concepts:

- A *resource type* is the name used in the URL ( `Pods` , `Namespaces` , `Services` )
- All resource types have a concrete representation (their object schema) which is called a *kind*
- A list of instances of a resource type is known as a *collection*
- A single instance of a resource type is called a *resource*, and also usually represents an *object*
- For some resource types, the API includes one or more *sub-resources*, which are represented as URI paths below the resource

Most Kubernetes API resource types are *objects* – they represent a concrete instance of a concept on the cluster, like a pod or namespace. A smaller number of API resource types are *virtual* in that they often represent operations on objects, rather than objects, such as a permission check (use a POST with a JSON-encoded body of `SubjectAccessReview` to the `SubjectAccessReviews` resource), or the `Eviction` sub-resource of a Pod (used to trigger [API-initiated eviction](#)).

### Object names

All objects you can create via the API have a unique object *name* to allow idempotent creation and retrieval, except that virtual resource types may not have unique names if they are not retrievable, or do not rely on idempotency. Within a *namespace*, only one object of a given kind can have a given name at a time. However, if you delete the object, you can make a new object with the same name. Some objects are not namespaced (for example: Nodes), and so their names must be unique across the whole cluster.

### API verbs

Almost all object resource types support the standard HTTP verbs - GET, POST, PUT, PATCH, and DELETE. Kubernetes also uses its own verbs, which are often written lowercase to distinguish them from HTTP verbs.

Kubernetes uses the term **list** to describe returning a [collection](#) of resources to distinguish from retrieving a single resource which is usually called a **get**. If you sent an HTTP GET request with the `?watch` query parameter, Kubernetes calls this a **watch** and not a **get** (see [Efficient detection of changes](#) for more details).

For PUT requests, Kubernetes internally classifies these as either **create** or **update** based on the state of the existing object. An **update** is different from a **patch**; the HTTP verb for a **patch** is PATCH.

## Resource URLs

All resource types are either scoped by the cluster (`/apis/GROUP/VERSION/*`) or to a namespace (`/apis/GROUP/VERSION/namespaces/NAMESPACE/*`). A namespace-scoped resource type will be deleted when its namespace is deleted and access to that resource type is controlled by authorization checks on the namespace scope.

Note: core resources use `/api` instead of `/apis` and omit the GROUP path segment.

Examples:

- `/api/v1/namespaces`
- `/api/v1/pods`
- `/api/v1/namespaces/my-namespace/pods`
- `/apis/apps/v1/deployments`
- `/apis/apps/v1/namespaces/my-namespace/deployments`
- `/apis/apps/v1/namespaces/my-namespace/deployments/my-deployment`

You can also access collections of resources (for example: listing all Nodes). The following paths are used to retrieve collections and resources:

- Cluster-scoped resources:
  - `GET /apis/GROUP/VERSION/RESOURCETYPE` - return the collection of resources of the resource type
  - `GET /apis/GROUP/VERSION/RESOURCETYPE/NAME` - return the resource with NAME under the resource type
- Namespace-scoped resources:
  - `GET /apis/GROUP/VERSION/RESOURCETYPE` - return the collection of all instances of the resource type across all namespaces
  - `GET /apis/GROUP/VERSION/namespaces/NAMESPACE/RESOURCETYPE` - return collection of all instances of the resource type in NAMESPACE
  - `GET /apis/GROUP/VERSION/namespaces/NAMESPACE/RESOURCETYPE/NAME` - return the instance of the resource type with NAME in NAMESPACE

Since a namespace is a cluster-scoped resource type, you can retrieve the list (“collection”) of all namespaces with `GET /api/v1/namespaces` and details about a particular namespace with `GET /api/v1/namespaces/NAME`.

- Cluster-scoped subresource: `GET /apis/GROUP/VERSION/RESOURCETYPE/NAME/SUBRESOURCE`
- Namespace-scoped subresource: `GET /apis/GROUP/VERSION/namespaces/NAMESPACE/RESOURCETYPE/NAME/SUBRESOURCE`

The verbs supported for each subresource will differ depending on the object - see the [API reference](#) for more information. It is not possible to

access sub-resources across multiple resources - generally a new virtual resource type would be used if that becomes necessary.

## Efficient detection of changes

The Kubernetes API allows clients to make an initial request for an object or a collection, and then to track changes since that initial request: a **watch**. Clients can send a **list** or a **get** and then make a follow-up **watch** request.

To make this change tracking possible, every Kubernetes object has a `resourceVersion` field representing the version of that resource as stored in the underlying persistence layer. When retrieving a collection of resources (either namespace or cluster scoped), the response from the API server contains a `resourceVersion` value. The client can use that `resourceVersion` to initiate a **watch** against the API server.

When you send a **watch** request, the API server responds with a stream of changes. These changes itemize the outcome of operations (such as **create**, **delete**, and **update**) that occurred after the `resourceVersion` you specified as a parameter to the **watch** request. The overall **watch** mechanism allows a client to fetch the current state and then subscribe to subsequent changes, without missing any events.

If a client **watch** is disconnected then that client can start a new **watch** from the last returned `resourceVersion`; the client could also perform a fresh **get** / **list** request and begin again. See [Resource Version Semantics](#) for more detail.

For example:

1. List all of the pods in a given namespace.

```
GET /api/v1/namespaces/test/pods
---
200 OK
Content-Type: application/json

{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {"resourceVersion": "10245"},
  "items": [...]
}
```

2. Starting from resource version 10245, receive notifications of any API operations (such as **create**, **delete**, **patch** or **update**) that affect Pods in the `test` namespace. Each change notification is a JSON document. The HTTP response body (served as `application/json`) consists a series of JSON documents.

```
GET /api/v1/namespaces/test/pods?watch=1&resourceVersion=10245
---
200 OK
Transfer-Encoding: chunked
Content-Type: application/json

{
  "type": "ADDED",
  "object": {"kind": "Pod", "apiVersion": "v1", "metadata": {"re
}
{
  "type": "MODIFIED",
  "object": {"kind": "Pod", "apiVersion": "v1", "metadata": {"re
}
...
...
```

A given Kubernetes server will only preserve a historical record of changes for a limited time. Clusters using etcd 3 preserve changes in the last 5 minutes by default. When the requested **watch** operations fail because the historical version of that resource is not available, clients must handle the case by recognizing the status code `410 Gone`, clearing their local cache, performing a new **get** or **list** operation, and starting the **watch** from the `resourceVersion` that was returned.

For subscribing to collections, Kubernetes client libraries typically offer some form of standard tool for this **list-then-watch** logic. (In the Go client library, this is called a `Reflector` and is located in the `k8s.io/client-go/tools/cache` package.)

## Watch bookmarks

To mitigate the impact of short history window, the Kubernetes API provides a watch event named `BOOKMARK`. It is a special kind of event to mark that all changes up to a given `resourceVersion` the client is requesting have already been sent. The document representing the `BOOKMARK` event is of the type requested by the request, but only includes a `.metadata.resourceVersion` field. For example:

As a client, you can request `BOOKMARK` events by setting the `allowWatchBookmarks=true` query parameter to a **watch** request, but you shouldn't assume bookmarks are returned at any specific interval, nor can clients assume that the API server will send any `BOOKMARK` event even when requested.

# Streaming lists

## ⓘ FEATURE STATE: Kubernetes v1.27 [alpha]

On large clusters, retrieving the collection of some resource types may result in a significant increase of resource usage (primarily RAM) on the control plane. In order to alleviate its impact and simplify the user experience of the **list** + **watch** pattern, Kubernetes v1.27 introduces as an alpha feature the support for requesting the initial state (previously requested via the **list** request) as part of the **watch** request.

Provided that the `WatchList` [feature gate](#) is enabled, this can be achieved by specifying `sendInitialEvents=true` as query string parameter in a **watch** request. If set, the API server starts the watch stream with synthetic init events (of type `ADDED`) to build the whole state of all existing objects followed by a [BOOKMARK event](#) (if requested via `allowWatchBookmarks=true` option). The bookmark event includes the resource version to which it is synced. After sending the bookmark event, the API server continues as for any other **watch** request.

When you set `sendInitialEvents=true` in the query string, Kubernetes also requires that you set `resourceVersionMatch` to `NotOlderThan` value. If you provided `resourceVersion` in the query string without providing a value or don't provide it at all, this is interpreted as a request for *consistent read*; the bookmark event is sent when the state is synced at least to the moment of a consistent read from when the request started to be processed. If you specify `resourceVersion` (in the query string), the bookmark event is sent when the state is synced at least to the provided resource version.

## Example

An example: you want to watch a collection of Pods. For that collection, the current resource version is 10245 and there are two pods: `foo` and `bar`. Then sending the following request (explicitly requesting *consistent read* by setting empty resource version using `resourceVersion=`) could result in the following sequence of events:

```
GET /api/v1/namespaces/test/pods?watch=1&sendInitialEvents=true&allow
---
200 OK
Transfer-Encoding: chunked
Content-Type: application/json

{
  "type": "ADDED",
  "object": {"kind": "Pod", "apiVersion": "v1", "metadata": {"resource
}
{
  "type": "ADDED",
  "object": {"kind": "Pod", "apiVersion": "v1", "metadata": {"resource
}
{
  "type": "BOOKMARK",
  "object": {"kind": "Pod", "apiVersion": "v1", "metadata": {"resource
}
...
<followed by regular watch stream starting from resourceVersion="1024
```

## Response compression

## ⓘ FEATURE STATE: Kubernetes v1.16 [beta]

`APIResponseCompression` is an option that allows the API server to compress the responses for `get` and `list` requests, reducing the network bandwidth and improving the performance of large-scale clusters. It is enabled by default since Kubernetes 1.16 and it can be disabled by including `APIResponseCompression=false` in the `--feature-gates` flag on the API server.

API response compression can significantly reduce the size of the response, especially for large resources or [collections](#). For example, a `list` request for pods can return hundreds of kilobytes or even megabytes of data, depending on the number of pods and their attributes. By compressing the response, the network bandwidth can be saved and the latency can be reduced.

To verify if `APIResponseCompression` is working, you can send a `get` or `list` request to the API server with an `Accept-Encoding` header, and check the response size and headers. For example:

```
GET /api/v1/pods
Accept-Encoding: gzip
---
200 OK
Content-Type: application/json
content-encoding: gzip
...
```

The `content-encoding` header indicates that the response is compressed with `gzip`.

## Retrieving large results sets in chunks

### ⓘ FEATURE STATE: Kubernetes v1.29 [stable]

On large clusters, retrieving the collection of some resource types may result in very large responses that can impact the server and client. For instance, a cluster may have tens of thousands of Pods, each of which is equivalent to roughly 2 KiB of encoded JSON. Retrieving all pods across all namespaces may result in a very large response (10-20MB) and consume a large amount of server resources.

The Kubernetes API server supports the ability to break a single large collection request into many smaller chunks while preserving the consistency of the total request. Each chunk can be returned sequentially which reduces both the total size of the request and allows user-oriented clients to display results incrementally to improve responsiveness.

You can request that the API server handles a `list` by serving single collection using pages (which Kubernetes calls *chunks*). To retrieve a single collection in chunks, two query parameters `limit` and `continue` are supported on requests against collections, and a response field `continue` is returned from all `list` operations in the collection's `metadata` field. A client should specify the maximum results they wish to receive in each chunk with `limit` and the server will return up to `limit` resources in the result and include a `continue` value if there are more resources in the collection.

As an API client, you can then pass this `continue` value to the API server on the next request, to instruct the server to return the next page (*chunk*) of results. By continuing until the server returns an empty `continue` value, you can retrieve the entire collection.

Like a **watch** operation, a `continue` token will expire after a short amount of time (by default 5 minutes) and return a `410 Gone` if more results cannot be returned. In this case, the client will need to start from the beginning or omit the `limit` parameter.

For example, if there are 1,253 pods on the cluster and you want to receive chunks of 500 pods at a time, request those chunks as follows:

1. List all of the pods on a cluster, retrieving up to 500 pods each time.

```
GET /api/v1/pods?limit=500
---
200 OK
Content-Type: application/json

{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "10245",
    "continue": "ENCODED_CONTINUE_TOKEN",
    "remainingItemCount": 753,
    ...
  },
  "items": [...] // returns pods 1-500
}
```

2. Continue the previous call, retrieving the next set of 500 pods.

```
GET /api/v1/pods?limit=500&continue=ENCODED_CONTINUE_TOKEN
---
200 OK
Content-Type: application/json

{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "10245",
    "continue": "ENCODED_CONTINUE_TOKEN_2",
    "remainingItemCount": 253,
    ...
  },
  "items": [...] // returns pods 501-1000
}
```

3. Continue the previous call, retrieving the last 253 pods.

```
GET /api/v1/pods?limit=500&continue=ENCODED_CONTINUE_TOKEN_2
---
200 OK
Content-Type: application/json

{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "10245",
    "continue": "", // continue token is empty because we have no
    ...
  },
  "items": [...] // returns pods 1001-1253
}
```

Notice that the `resourceVersion` of the collection remains constant across each request, indicating the server is showing you a consistent snapshot of the pods. Pods that are created, updated, or deleted after version `10245` would not be shown unless you make a separate **list** request without the `continue` token. This allows you to break large requests into smaller chunks and then perform a **watch** operation on the full set without missing any updates.

`remainingItemCount` is the number of subsequent items in the collection that are not included in this response. If the **list** request contained label or field selectors then the number of remaining items is unknown and the API server does not include a `remainingItemCount` field in its response. If the **list** is complete (either because it is not chunking, or because this is the last chunk), then there are no more remaining items and the API server does not include a `remainingItemCount` field in its response. The intended use of the `remainingItemCount` is estimating the size of a collection.

## Collections

In Kubernetes terminology, the response you get from a **list** is a *collection*. However, Kubernetes defines concrete kinds for collections of different types of resource. Collections have a kind named for the resource kind, with `List` appended.

When you query the API for a particular type, all items returned by that query are of that type. For example, when you **list** Services, the collection response has `kind` set to [ServiceList](#); each item in that collection represents a single Service. For example:

```
GET /api/v1/services
```

```
{  
  "kind": "ServiceList",  
  "apiVersion": "v1",  
  "metadata": {  
    "resourceVersion": "2947301"  
  },  
  "items": [  
    {  
      "metadata": {  
        "name": "kubernetes",  
        "namespace": "default",  
        ...  
      },  
      "metadata": {  
        "name": "kube-dns",  
        "namespace": "kube-system",  
        ...  
      },  
      ...  
    }  
  ]  
}
```

There are dozens of collection types (such as `PodList`, `ServiceList`, and `NodeList`) defined in the Kubernetes API. You can get more information about each collection type from the [Kubernetes API](#) documentation.

Some tools, such as `kubectl`, represent the Kubernetes collection mechanism slightly differently from the Kubernetes API itself. Because the output of `kubectl` might include the response from multiple **list** operations at the API level, `kubectl` represents a list of items using `kind: List`. For example:

```
kubectl get services -A -o yaml
```

```
apiVersion: v1  
kind: List  
metadata:  
  resourceVersion: ""  
  selfLink: ""  
items:  
- apiVersion: v1  
  kind: Service  
  metadata:  
    creationTimestamp: "2021-06-03T14:54:12Z"  
    labels:  
      component: apiserver  
      provider: kubernetes  
    name: kubernetes  
    namespace: default  
    ...  
- apiVersion: v1  
  kind: Service  
  metadata:  
    annotations:  
      prometheus.io/port: "9153"  
      prometheus.io/scrape: "true"  
    creationTimestamp: "2021-06-03T14:54:14Z"  
    labels:  
      k8s-app: kube-dns  
      kubernetes.io/cluster-service: "true"  
      kubernetes.io/name: CoreDNS  
    name: kube-dns  
    namespace: kube-system
```

**Note:**

Keep in mind that the Kubernetes API does not have a `kind` named `List`.

`kind: List` is a client-side, internal implementation detail for processing collections that might be of different kinds of object. Avoid depending on `kind: List` in automation or other code.

## Receiving resources as Tables

When you run `kubectl get`, the default output format is a simple tabular representation of one or more instances of a particular resource type. In the past, clients were required to reproduce the tabular and `describe` output implemented in `kubectl` to perform simple lists of objects. A few limitations of that approach include non-trivial logic when dealing with certain objects. Additionally, types provided by API aggregation or third party resources are not known at compile time. This means that generic implementations had to be in place for types unrecognized by a client.

In order to avoid potential limitations as described above, clients may request the Table representation of objects, delegating specific details of printing to the server. The Kubernetes API implements standard HTTP content type negotiation: passing an `Accept` header containing a value of `application/json;as=Table;g=meta.k8s.io;v=v1` with a `GET` call will request that the server return objects in the Table content type.

For example, list all of the pods on a cluster in the Table format.

```
GET /api/v1/pods
Accept: application/json;as=Table;g=meta.k8s.io;v=v1
---
200 OK
Content-Type: application/json

{
  "kind": "Table",
  "apiVersion": "meta.k8s.io/v1",
  ...
  "columnDefinitions": [
    ...
  ]
}
```

For API resource types that do not have a custom Table definition known to the control plane, the API server returns a default Table response that consists of the resource's `name` and `creationTimestamp` fields.

```
GET /apis/crd.example.com/v1alpha1/namespaces/default/resources
---
200 OK
Content-Type: application/json
...

{
  "kind": "Table",
  "apiVersion": "meta.k8s.io/v1",
  ...
  "columnDefinitions": [
    {
      "name": "Name",
      "type": "string",
      ...
    },
    {
      "name": "Created At",
      "type": "date",
      ...
    }
  ]
}
```

Not all API resource types support a Table response; for example, a `CustomResourceDefinitions` might not define field-to-table mappings, and an `APIService` that [extends the core Kubernetes API](#) might not serve Table responses at all. If you are implementing a client that uses the Table information and must work against all resource types, including extensions, you should make requests that specify multiple content types in the `Accept` header. For example:

```
Accept: application/json;as=Table;g=meta.k8s.io;v=v1, application/jso
```

## Alternate representations of resources

By default, Kubernetes returns objects serialized to JSON with content type `application/json`. This is the default serialization format for the API. However, clients may request the more efficient [Protobuf representation](#) of these objects for better performance at scale. The Kubernetes API implements standard HTTP content type negotiation: passing an `Accept` header with a `GET` call will request that the server tries to return a response in your preferred media type, while sending an object in Protobuf to the server for a `PUT` or `POST` call means that you must set the `Content-Type` header appropriately.

The server will return a response with a `Content-Type` header if the requested format is supported, or the `406 Not acceptable` error if none of the media types you requested are supported. All built-in resource types support the `application/json` media type.

See the Kubernetes [API reference](#) for a list of supported content types for each API.

For example:

1. List all of the pods on a cluster in Protobuf format.

```
GET /api/v1/pods
Accept: application/vnd.kubernetes.protobuf
---
200 OK
Content-Type: application/vnd.kubernetes.protobuf

... binary encoded PodList object
```

2. Create a pod by sending Protobuf encoded data to the server, but request a response in JSON.

```
POST /api/v1/namespaces/test/pods
Content-Type: application/vnd.kubernetes.protobuf
Accept: application/json
... binary encoded Pod object
---
200 OK
Content-Type: application/json

{
  "kind": "Pod",
  "apiVersion": "v1",
  ...
}
```

Not all API resource types support Protobuf; specifically, Protobuf isn't available for resources that are defined as [CustomResourceDefinitions](#) or are served via the [aggregation layer](#). As a client, if you might need to work with extension types you should specify multiple content types in the request `Accept` header to support fallback to JSON. For example:

```
Accept: application/vnd.kubernetes.protobuf, application/json
```

## Kubernetes Protobuf encoding

Kubernetes uses an envelope wrapper to encode Protobuf responses. That wrapper starts with a 4 byte magic number to help identify content in disk or in etcd as Protobuf (as opposed to JSON), and then is followed by a Protobuf encoded wrapper message, which describes the encoding and type of the underlying object and then contains the object.

The wrapper format is:

```
A four byte magic number prefix:  
Bytes 0-3: "k8s\x00" [0x6b, 0x38, 0x73, 0x00]  
  
An encoded Protobuf message with the following IDL:  
message Unknown {  
    // typeMeta should have the string values for "kind" and "apiVersion"  
    optional TypeMeta typeMeta = 1;  
  
    // raw will hold the complete serialized object in protobuf. See  
    optional bytes raw = 2;  
  
    // contentEncoding is encoding used for the raw data. Unspecified  
    optional string contentEncoding = 3;  
  
    // contentType is the serialization method used to serialize 'raw'  
    // omitted.  
    optional string contentType = 4;  
}  
  
message TypeMeta {  
    // apiVersion is the group/version for this type  
    optional string apiVersion = 1;  
    // kind is the name of the object schema. A protobuf definition should  
    optional string kind = 2;  
}
```

**Note:**

Clients that receive a response in `application/vnd.kubernetes.protobuf` that does not match the expected prefix should reject the response, as future versions may need to alter the serialization format in an incompatible way and will do so by changing the prefix.

## Resource deletion

When you **delete** a resource this takes place in two phases.

1. *finalization*
2. removal

```
{  
    "kind": "ConfigMap",  
    "apiVersion": "v1",  
    "metadata": {  
        "finalizers": ["url.io/neat-finalization", "other-url.io/my-finalizer"],  
        "deletionTimestamp": nil,  
    }  
}
```

When a client first sends a **delete** to request the removal of a resource, the `.metadata.deletionTimestamp` is set to the current time. Once the `.metadata.deletionTimestamp` is set, external controllers that act on finalizers may start performing their cleanup work at any time, in any order.

Order is **not** enforced between finalizers because it would introduce significant risk of stuck `.metadata.finalizers`.

The `.metadata.finalizers` field is shared: any actor with permission can reorder it. If the finalizer list were processed in order, then this might

lead to a situation in which the component responsible for the first finalizer in the list is waiting for some signal (field value, external system, or other) produced by a component responsible for a finalizer later in the list, resulting in a deadlock.

Without enforced ordering, finalizers are free to order amongst themselves and are not vulnerable to ordering changes in the list.

Once the last finalizer is removed, the resource is actually removed from etcd.

## Single resource API

The Kubernetes API verbs **get**, **create**, **update**, **patch**, **delete** and **proxy** support single resources only. These verbs with single resource support have no support for submitting multiple resources together in an ordered or unordered list or transaction.

When clients (including kubectl) act on a set of resources, the client makes a series of single-resource API requests, then aggregates the responses if needed.

By contrast, the Kubernetes API verbs **list** and **watch** allow getting multiple resources, and **deletecollection** allows deleting multiple resources.

## Field validation

Kubernetes always validates the type of fields. For example, if a field in the API is defined as a number, you cannot set the field to a text value. If a field is defined as an array of strings, you can only provide an array. Some fields allow you to omit them, other fields are required. Omitting a required field from an API request is an error.

If you make a request with an extra field, one that the cluster's control plane does not recognize, then the behavior of the API server is more complicated.

By default, the API server drops fields that it does not recognize from an input that it receives (for example, the JSON body of a `PUT` request).

There are two situations where the API server drops fields that you supplied in an HTTP request.

These situations are:

1. The field is unrecognized because it is not in the resource's OpenAPI schema. (One exception to this is for CRDs that explicitly choose not to prune unknown fields via `x-kubernetes-preserve-unknown-fields` ).
2. The field is duplicated in the object.

## Validation for unrecognized or duplicate fields

 **FEATURE STATE:** Kubernetes v1.27 [stable]

From 1.25 onward, unrecognized or duplicate fields in an object are detected via validation on the server when you use HTTP verbs that can submit data (`POST`, `PUT`, and `PATCH`). Possible levels of validation are `Ignore`, `Warn` (default), and `Strict`.

**Ignore**

The API server succeeds in handling the request as it would without the erroneous fields being set, dropping all unknown and duplicate fields and giving no indication it has done so.

**Warn**

(Default) The API server succeeds in handling the request, and reports a warning to the client. The warning is sent using the [Warning](#): response header, adding one warning item for each unknown or duplicate field. For more information about warnings and the Kubernetes API, see the blog article [Warning: Helpful Warnings Ahead](#).

**Strict**

The API server rejects the request with a 400 Bad Request error when it detects any unknown or duplicate fields. The response message from the API server specifies all the unknown or duplicate fields that the API server has detected.

The field validation level is set by the `fieldValidation` query parameter.

**Note:**

If you submit a request that specifies an unrecognized field, and that is also invalid for a different reason (for example, the request provides a string value where the API expects an integer for a known field), then the API server responds with a 400 Bad Request error, but will not provide any information on unknown or duplicate fields (only which fatal error it encountered first).

You always receive an error response in this case, no matter what field validation level you requested.

Tools that submit requests to the server (such as `kubectl`), might set their own defaults that are different from the `Warn` validation level that the API server uses by default.

The `kubectl` tool uses the `--validate` flag to set the level of field validation. It accepts the values `ignore`, `warn`, and `strict` while also accepting the values `true` (equivalent to `strict`) and `false` (equivalent to `ignore`). The default validation setting for `kubectl` is `--validate=true`, which means strict server-side field validation.

When `kubectl` cannot connect to an API server with field validation (API servers prior to Kubernetes 1.27), it will fall back to using client-side validation. Client-side validation will be removed entirely in a future version of `kubectl`.

**Note:**

Prior to Kubernetes 1.25 `kubectl --validate` was used to toggle client-side validation on or off as a boolean flag.

## Dry-run

ⓘ **FEATURE STATE:** Kubernetes v1.19 [stable]

When you use HTTP verbs that can modify resources ( `POST` , `PUT` , `PATCH` , and `DELETE` ), you can submit your request in a *dry run* mode. Dry run

mode helps to evaluate a request through the typical request stages (admission chain, validation, merge conflicts) up until persisting objects to storage. The response body for the request is as close as possible to a non-dry-run response. Kubernetes guarantees that dry-run requests will not be persisted in storage or have any other side effects.

## Make a dry-run request

Dry-run is triggered by setting the `dryRun` query parameter. This parameter is a string, working as an enum, and the only accepted values are:

### [no value set]

Allow side effects. You request this with a query string such as `?dryRun` or `?dryRun&pretty=true`. The response is the final object that would have been persisted, or an error if the request could not be fulfilled.

### All

Every stage runs as normal, except for the final storage stage where side effects are prevented.

When you set `?dryRun=All`, any relevant admission controllers are run, validating admission controllers check the request post-mutation, merge is performed on `PATCH`, fields are defaulted, and schema validation occurs. The changes are not persisted to the underlying storage, but the final object which would have been persisted is still returned to the user, along with the normal status code.

If the non-dry-run version of a request would trigger an admission controller that has side effects, the request will be failed rather than risk an unwanted side effect. All built in admission control plugins support dry-run. Additionally, admission webhooks can declare in their [configuration object](#) that they do not have side effects, by setting their `sideEffects` field to `None`.

### Note:

If a webhook actually does have side effects, then the `sideEffects` field should be set to "NoneOnDryRun". That change is appropriate provided that the webhook is also be modified to understand the `DryRun` field in `AdmissionReview`, and to prevent side effects on any request marked as dry runs.

Here is an example dry-run request that uses `?dryRun=All`:

```
POST /api/v1/namespaces/test/pods?dryRun=All
Content-Type: application/json
Accept: application/json
```

The response would look the same as for non-dry-run request, but the values of some generated fields may differ.

## Generated values

Some values of an object are typically generated before the object is persisted. It is important not to rely upon the values of these fields set by a dry-run request, since these values will likely be different in dry-run mode from when the real request is made. Some of these fields are:

- `name` : if `generateName` is set, `name` will have a unique random name

- `creationTimestamp` / `deletionTimestamp` : records the time of creation/deletion
- `UID` : [uniquely identifies](#) the object and is randomly generated (non-deterministic)
- `resourceVersion` : tracks the persisted version of the object
- Any field set by a mutating admission controller
- For the `Service` resource: Ports or IP addresses that the kube-apiserver assigns to Service objects

## Dry-run authorization

Authorization for dry-run and non-dry-run requests is identical. Thus, to make a dry-run request, you must be authorized to make the non-dry-run request.

For example, to run a dry-run **patch** for a Deployment, you must be authorized to perform that **patch**. Here is an example of a rule for Kubernetes RBAC that allows patching Deployments:

```
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["patch"]
```

See [Authorization Overview](#).

## Updates to existing resources

Kubernetes provides several ways to update existing objects. You can read [choosing an update mechanism](#) to learn about which approach might be best for your use case.

You can overwrite (**update**) an existing resource - for example, a ConfigMap - using an HTTP PUT. For a PUT request, it is the client's responsibility to specify the `resourceVersion` (taking this from the object being updated). Kubernetes uses that `resourceVersion` information so that the API server can detect lost updates and reject requests made by a client that is out of date with the cluster. In the event that the resource has changed (the `resourceVersion` the client provided is stale), the API server returns a `409 Conflict` error response.

Instead of sending a PUT request, the client can send an instruction to the API server to **patch** an existing resource. A **patch** is typically appropriate if the change that the client wants to make isn't conditional on the existing data. Clients that need effective detection of lost updates should consider making their request conditional on the existing `resourceVersion` (either HTTP PUT or HTTP PATCH), and then handle any retries that are needed in case there is a conflict.

The Kubernetes API supports four different PATCH operations, determined by their corresponding HTTP `Content-Type` header:

### `application/apply-patch+yaml`

Server Side Apply YAML (a Kubernetes-specific extension, based on YAML). All JSON documents are valid YAML, so you can also submit JSON using this media type. See [Server Side Apply serialization](#) for more details.

To Kubernetes, this is a **create** operation if the object does not exist, or a **patch** operation if the object already exists.

### application/json-patch+json

JSON Patch, as defined in [RFC6902](#). A JSON patch is a sequence of operations that are executed on the resource; for example `{"op": "add", "path": "/a/b/c", "value": [ "foo", "bar" ]}`.

To Kubernetes, this is a **patch** operation.

A **patch** using `application/json-patch+json` can include conditions to validate consistency, allowing the operation to fail if those conditions are not met (for example, to avoid a lost update).

### application/merge-patch+json

JSON Merge Patch, as defined in [RFC7386](#). A JSON Merge Patch is essentially a partial representation of the resource. The submitted JSON is combined with the current resource to create a new one, then the new one is saved.

To Kubernetes, this is a **patch** operation.

### application/strategic-merge-patch+json

Strategic Merge Patch (a Kubernetes-specific extension based on JSON). Strategic Merge Patch is a custom implementation of JSON Merge Patch. You can only use Strategic Merge Patch with built-in APIs, or with aggregated API servers that have special support for it. You cannot use `application/strategic-merge-patch+json` with any API defined using a [CustomResourceDefinition](#).

#### Note:

The Kubernetes *server side apply* mechanism has superseded Strategic Merge Patch.

Kubernetes' [Server Side Apply](#) feature allows the control plane to track managed fields for newly created objects. Server Side Apply provides a clear pattern for managing field conflicts, offers server-side **apply** and **update** operations, and replaces the client-side functionality of `kubectl apply`.

For Server-Side Apply, Kubernetes treats the request as a **create** if the object does not yet exist, and a **patch** otherwise. For other requests that use PATCH at the HTTP level, the logical Kubernetes operation is always **patch**.

See [Server Side Apply](#) for more details.

## Choosing an update mechanism

### HTTP PUT to replace existing resource

The **update** (HTTP `PUT`) operation is simple to implement and flexible, but has drawbacks:

- You need to handle conflicts where the `resourceVersion` of the object changes between your client reading it and trying to write it back. Kubernetes always detects the conflict, but you as the client author need to implement retries.
- You might accidentally drop fields if you decode an object locally (for example, using client-go, you could receive fields that your client does not know how to handle - and then drop them as part of your update).
- If there's a lot of contention on the object (even on a field, or set of fields, that you're not trying to edit), you might have trouble sending the update. The problem is worse for larger objects and for objects

with many fields.

## HTTP PATCH using JSON Patch

A **patch** update is helpful, because:

- As you're only sending differences, you have less data to send in the `PATCH` request.
- You can make changes that rely on existing values, such as copying the value of a particular field into an annotation.
- Unlike with an **update** (`HTTP PUT`), making your change can happen right away even if there are frequent changes to unrelated fields): you usually would not need to retry.
  - You might still need to specify the `resourceVersion` (to match an existing object) if you want to be extra careful to avoid lost updates
  - It's still good practice to write in some retry logic in case of errors.
- You can use test conditions to carefully craft specific update conditions. For example, you can increment a counter without reading it if the existing value matches what you expect. You can do this with no lost update risk, even if the object has changed in other ways since you last wrote to it. (If the test condition fails, you can fall back to reading the current value and then write back the changed number).

However:

- you need more local (client) logic to build the patch; it helps a lot if you have a library implementation of JSON Patch, or even for making a JSON Patch specifically against Kubernetes
- as the author of client software, you need to be careful when building the patch (the HTTP request body) not to drop fields (the order of operations matters)

## HTTP PATCH using Server-Side Apply

Server-Side Apply has some clear benefits:

- A single round trip: it rarely requires making a `GET` request first.
  - and you can still detect conflicts for unexpected changes
  - you have the option to force override a conflict, if appropriate
- Client implementations are easy to make
- You get an atomic create-or-update operation without extra effort (similar to `UPSERT` in some SQL dialects)

However:

- Server-Side Apply does not work at all for field changes that depend on a current value of the object
- You can only apply updates to objects. Some resources in the Kubernetes HTTP API are not objects (they do not have a `.metadata` field), and Server-Side Apply is only relevant for Kubernetes objects.

## Resource versions

Resource versions are strings that identify the server's internal version of an object. Resource versions can be used by clients to determine when objects have changed, or to express data consistency requirements when getting, listing and watching resources. Resource versions must be

treated as opaque by clients and passed unmodified back to the server.

You must not assume resource versions are numeric or collatable. API clients may only compare two resource versions for equality (this means that you must not compare resource versions for greater-than or less-than relationships).

## resourceVersion fields in metadata

Clients find resource versions in resources, including the resources from the response stream for a **watch**, or when using **list** to enumerate resources.

[v1.meta/ObjectMeta](#) - The `metadata.resourceVersion` of a resource instance identifies the resource version the instance was last modified at.

[v1.meta/ListMeta](#) - The `metadata.resourceVersion` of a resource collection (the response to a **list**) identifies the resource version at which the collection was constructed.

## resourceVersion parameters in query strings

The **get**, **list**, and **watch** operations support the `resourceVersion` parameter. From version v1.19, Kubernetes API servers also support the `resourceVersionMatch` parameter on *list* requests.

The API server interprets the `resourceVersion` parameter differently depending on the operation you request, and on the value of `resourceVersion`. If you set `resourceVersionMatch` then this also affects the way matching happens.

## Semantics for **get** and **list**

For **get** and **list**, the semantics of `resourceVersion` are:

### get:

resourceVersion	resourceVersion="0"	resourceVersion="{value other than 0}"
unset	Any	Not older than

### list:

From version v1.19, Kubernetes API servers support the `resourceVersionMatch` parameter on *list* requests. If you set both `resourceVersion` and `resourceVersionMatch`, the `resourceVersionMatch` parameter determines how the API server interprets `resourceVersion`.

You should always set the `resourceVersionMatch` parameter when setting `resourceVersion` on a **list** request. However, be prepared to handle the case where the API server that responds is unaware of `resourceVersionMatch` and ignores it.

Unless you have strong consistency requirements, using `resourceVersionMatch=NotOlderThan` and a known `resourceVersion` is preferable since it can achieve better performance and scalability of your cluster than leaving `resourceVersion` and `resourceVersionMatch` unset, which requires quorum read to be served.

Setting the `resourceVersionMatch` parameter without setting `resourceVersion` is not valid.

This table explains the behavior of **list** requests with various

combinations of `resourceVersion` and `resourceVersionMatch`:

<code>resourceVersionMatch</code> param	<code>paging</code> params	<code>resourceVersion</code> not set	<code>resourceVersion</code> set
<code>unset</code>	<code>limit unset</code>	Most Recent	Any
<code>unset</code>	<code>limit=&lt;n&gt;,</code> <code>continue unset</code>	Most Recent	Any
<code>unset</code>	<code>limit=&lt;n&gt;,</code> <code>continue=&lt;token&gt;</code>	Continue Token, Exact	Inval Cont Exact
<code>resourceVersionMatch=Exact</code>	<code>limit unset</code>	Invalid	Inval
<code>resourceVersionMatch=Exact</code>	<code>limit=&lt;n&gt;,</code> <code>continue unset</code>	Invalid	Inval
<code>resourceVersionMatch=NotOlderThan</code>	<code>limit unset</code>	Invalid	Any
<code>resourceVersionMatch=NotOlderThan</code>	<code>limit=&lt;n&gt;,</code> <code>continue unset</code>	Invalid	Any

#### Note:

If your cluster's API server does not honor the `resourceVersionMatch` parameter, the behavior is the same as if you did not set it.

The meaning of the **get** and **list** semantics are:

#### Any

Return data at any resource version. The newest available resource version is preferred, but strong consistency is not required; data at any resource version may be served. It is possible for the request to return data at a much older resource version that the client has previously observed, particularly in high availability configurations, due to partitions or stale caches. Clients that cannot tolerate this should not use this semantic.

#### Most recent

Return data at the most recent resource version. The returned data must be consistent (in detail: served from etcd via a quorum read). For etcd v3.4.31+ and v3.5.13+ Kubernetes 1.31 serves “most recent” reads from the *watch cache*: an internal, in-memory store within the API server that caches and mirrors the state of data persisted into etcd. Kubernetes requests progress notification to maintain cache consistency against the etcd persistence layer. Kubernetes versions v1.28 through to v1.30 also supported this feature, although as Alpha it was not recommended for production nor enabled by default until the v1.31 release.

#### Not older than

Return data at least as new as the provided `resourceVersion`. The newest available data is preferred, but any data not older than the provided `resourceVersion` may be served. For **list** requests to servers that honor the `resourceVersionMatch` parameter, this guarantees that the collection's `.metadata.resourceVersion` is not older than the requested `resourceVersion`, but does not make any guarantee about

the `.metadata.resourceVersion` of any of the items in that collection.

### Exact

Return data at the exact resource version provided. If the provided `resourceVersion` is unavailable, the server responds with HTTP 410 "Gone". For `list` requests to servers that honor the `resourceVersionMatch` parameter, this guarantees that the collection's `.metadata.resourceVersion` is the same as the `resourceVersion` you requested in the query string. That guarantee does not apply to the `.metadata.resourceVersion` of any items within that collection.

### Continue Token, Exact

Return data at the resource version of the initial paginated `list` call. The returned *continue tokens* are responsible for keeping track of the initially provided resource version for all paginated `list` calls after the initial paginated `list`.

#### Note:

When you `list` resources and receive a collection response, the response includes the [list metadata](#) of the collection as well as [object metadata](#) for each item in that collection. For individual objects found within a collection response, `.metadata.resourceVersion` tracks when that object was last updated, and not how up-to-date the object is when served.

When using `resourceVersionMatch=NotOlderThan` and `limit` is set, clients must handle HTTP 410 "Gone" responses. For example, the client might retry with a newer `resourceVersion` or fall back to `resourceVersion=""`.

When using `resourceVersionMatch=Exact` and `limit` is unset, clients must verify that the collection's `.metadata.resourceVersion` matches the requested `resourceVersion`, and handle the case where it does not. For example, the client might fall back to a request with `limit` set.

## Semantics for `watch`

For `watch`, the semantics of resource version are:

`watch`:

<code>resourceVersion</code> unset	<code>resourceVersion="0"</code>	<code>resourceVersion="{value other than 0}"</code>
Get State and Start at Most Recent	Get State and Start at Any	Start at Exact

The meaning of those `watch` semantics are:

### Get State and Start at Any

#### Caution:

Watches initialized this way may return arbitrarily stale data. Please review this semantic before using it, and favor the other semantics where possible.

Start a `watch` at any resource version; the most recent resource version available is preferred, but not required. Any starting resource version is allowed. It is possible for the `watch` to start at a much older resource version than the client has previously observed, particularly in high availability configurations, due to partitions or stale caches.

Clients that cannot tolerate this apparent rewinding should not start a **watch** with this semantic. To establish initial state, the **watch** begins with synthetic "Added" events for all resource instances that exist at the starting resource version. All following watch events are for all changes that occurred after the resource version the **watch** started at.

### Get State and Start at Most Recent

Start a **watch** at the most recent resource version, which must be consistent (in detail: served from etcd via a quorum read). To establish initial state, the **watch** begins with synthetic "Added" events of all resources instances that exist at the starting resource version. All following watch events are for all changes that occurred after the resource version the **watch** started at.

### Start at Exact

Start a **watch** at an exact resource version. The watch events are for all changes after the provided resource version. Unlike "Get State and Start at Most Recent" and "Get State and Start at Any", the **watch** is not started with synthetic "Added" events for the provided resource version. The client is assumed to already have the initial state at the starting resource version since the client provided the resource version.

## "410 Gone" responses

Servers are not required to serve all older resource versions and may return a HTTP 410 (Gone) status code if a client requests a `resourceVersion` older than the server has retained. Clients must be able to tolerate 410 (Gone) responses. See [Efficient detection of changes](#) for details on how to handle 410 (Gone) responses when watching resources.

If you request a `resourceVersion` outside the applicable limit then, depending on whether a request is served from cache or not, the API server may reply with a 410 Gone HTTP response.

## Unavailable resource versions

Servers are not required to serve unrecognized resource versions. If you request **list** or **get** for a resource version that the API server does not recognize, then the API server may either:

- wait briefly for the resource version to become available, then timeout with a 504 (Gateway Timeout) if the provided resource versions does not become available in a reasonable amount of time;
- respond with a `Retry-After` response header indicating how many seconds a client should wait before retrying the request.

If you request a resource version that an API server does not recognize, the kube-apiserver additionally identifies its error responses with a "Too large resource version" message.

If you make a **watch** request for an unrecognized resource version, the API server may wait indefinitely (until the request timeout) for the resource version to become available.

## 2.2 - Server-Side Apply

ⓘ **FEATURE STATE:** Kubernetes v1.22 [stable]

Kubernetes supports multiple appliers collaborating to manage the fields of a single [object](#).

Server-Side Apply provides an optional mechanism for your cluster's control plane to track changes to an object's fields. At the level of a specific resource, Server-Side Apply records and tracks information about control over the fields of that object.

Server-Side Apply helps users and [controllers](#) manage their resources through declarative configuration. Clients can create and modify [objects](#) declaratively by submitting their *fully specified intent*.

A fully specified intent is a partial object that only includes the fields and values for which the user has an opinion. That intent either creates a new object (using default values for unspecified fields), or is [combined](#), by the API server, with the existing object.

[Comparison with Client-Side Apply](#) explains how Server-Side Apply differs from the original, client-side `kubectl apply` implementation.

### Field management

The Kubernetes API server tracks *managed fields* for all newly created objects.

When trying to apply an object, fields that have a different value and are owned by another [manager](#) will result in a [conflict](#). This is done in order to signal that the operation might undo another collaborator's changes. Writes to objects with managed fields can be forced, in which case the value of any conflicted field will be overridden, and the ownership will be transferred.

Whenever a field's value does change, ownership moves from its current manager to the manager making the change.

Apply checks if there are any other field managers that also own the field. If the field is not owned by any other field managers, that field is set to its default value (if there is one), or otherwise is deleted from the object. The same rule applies to fields that are lists, associative lists, or maps.

For a user to manage a field, in the Server-Side Apply sense, means that the user relies on and expects the value of the field not to change. The user who last made an assertion about the value of a field will be recorded as the current field manager. This can be done by changing the field manager details explicitly using `HTTP POST (create)`, `PUT (update)`, or non-apply `PATCH (patch)`. You can also declare and record a field manager by including a value for that field in a Server-Side Apply operation.

A Server-Side Apply **patch** request requires the client to provide its identity as a [field manager](#). When using Server-Side Apply, trying to change a field that is controlled by a different manager results in a rejected request unless the client forces an override. For details of overrides, see [Conflicts](#).

When two or more appliers set a field to the same value, they share ownership of that field. Any subsequent attempt to change the value of the shared field, by any of the appliers, results in a conflict. Shared field

owners may give up ownership of a field by making a Server-Side Apply **patch** request that doesn't include that field.

Field management details are stored in a `managedFields` field that is part of an object's [metadata](#) .

If you remove a field from a manifest and apply that manifest, Server-Side Apply checks if there are any other field managers that also own the field. If the field is not owned by any other field managers, it is either deleted from the live object or reset to its default value, if it has one. The same rule applies to associative list or map items.

Compared to the (legacy) [kubectl.kubernetes.io/last-applied-configuration](#) annotation managed by `kubectl` , Server-Side Apply uses a more declarative approach, that tracks a user's (or client's) field management, rather than a user's last applied state. As a side effect of using Server-Side Apply, information about which field manager manages each field in an object also becomes available.

## Example

A simple example of an object created using Server-Side Apply could look like this:

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-cm
  namespace: default
  labels:
    test-label: test
  managedFields:
    - manager: kubectl
      operation: Apply # note capitalization: "Apply" (or "Update")
      apiVersion: v1
      time: "2010-10-10T0:00:00Z"
      fieldsType: FieldsV1
      fieldsV1:
        f:metadata:
          f:labels:
            f:test-label: {}
        f:data:
          f:key: {}
  data:
    key: some value
```

That example ConfigMap object contains a single field management record in `.metadata.managedFields` . The field management record consists of basic information about the managing entity itself, plus details about the fields being managed and the relevant operation ( `Apply` or `Update` ). If the request that last changed that field was a Server-Side Apply **patch** then the value of `operation` is `Apply` ; otherwise, it is `Update` .

There is another possible outcome. A client could submit an invalid request body. If the fully specified intent does not produce a valid object, the request fails.

It is however possible to change `.metadata.managedFields` through an **update**, or through a **patch** operation that does not use Server-Side Apply. Doing so is highly discouraged, but might be a reasonable option to try if, for example, the `.metadata.managedFields` get into an

inconsistent state (which should not happen in normal operations).

The format of `managedFields` is [described](#) in the Kubernetes API reference.

#### Caution:

The `.metadata.managedFields` field is managed by the API server. You should avoid updating it manually.

## Conflicts

A *conflict* is a special status error that occurs when an `Apply` operation tries to change a field that another manager also claims to manage. This prevents an applier from unintentionally overwriting the value set by another user. When this occurs, the applier has 3 options to resolve the conflicts:

- **Overwrite value, become sole manager:** If overwriting the value was intentional (or if the applier is an automated process like a controller) the applier should set the `force` query parameter to true (for `kubectl apply`, you use the `--force-conflicts` command line parameter), and make the request again. This forces the operation to succeed, changes the value of the field, and removes the field from all other managers' entries in `managedFields`.
- **Don't overwrite value, give up management claim:** If the applier doesn't care about the value of the field any more, the applier can remove it from their local model of the resource, and make a new request with that particular field omitted. This leaves the value unchanged, and causes the field to be removed from the applier's entry in `managedFields`.
- **Don't overwrite value, become shared manager:** If the applier still cares about the value of a field, but doesn't want to overwrite it, they can change the value of that field in their local model of the resource so as to match the value of the object on the server, and then make a new request that takes into account that local update. Doing so leaves the value unchanged, and causes that field's management to be shared by the applier along with all other field managers that already claimed to manage it.

## Field managers

Managers identify distinct workflows that are modifying the object (especially useful on conflicts!), and can be specified through the `fieldManager` query parameter as part of a modifying request. When you `Apply` to a resource, the `fieldManager` parameter is required. For other updates, the API server infers a field manager identity from the "User-Agent:" HTTP header (if present).

When you use the `kubectl` tool to perform a Server-Side Apply operation, `kubectl` sets the manager identity to "kubectl" by default.

## Serialization

At the protocol level, Kubernetes represents Server-Side Apply message bodies as [YAML](#), with the media type `application/apply-patch+yaml`.

#### Note:

Whether you are submitting JSON data or YAML data, use `application/apply-patch+yaml` as the `Content-Type` header value.

All JSON documents are valid YAML. However, Kubernetes has a bug where it uses a YAML parser that does not fully implement the YAML specification. Some JSON escapes may not be recognized.

The serialization is the same as for Kubernetes objects, with the exception that clients are not required to send a complete object.

Here's an example of a Server-Side Apply message body (fully specified intent):

```
{  
  "apiVersion": "v1",  
  "kind": "ConfigMap"  
}
```

(this would make a no-change update, provided that it was sent as the body of a **patch** request to a valid `v1/configmaps` resource, and with the appropriate request `Content-Type` ).

## Operations in scope for field management

The Kubernetes API operations where field management is considered are:

1. Server-Side Apply (HTTP `PATCH`, with content type `application/apply-patch+yaml` )
2. Replacing an existing object (**update** to Kubernetes; `PUT` at the HTTP level)

Both operations update `.metadata.managedFields`, but behave a little differently.

Unless you specify a forced override, an apply operation that encounters field-level conflicts always fails; by contrast, if you make a change using **update** that would affect a managed field, a conflict never provokes failure of the operation.

All Server-Side Apply **patch** requests are required to identify themselves by providing a `fieldManager` query parameter, while the query parameter is optional for **update** operations. Finally, when using the `Apply` operation you cannot define `managedFields` in the body of the request that you submit.

An example object with multiple managers could look like this:

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-cm
  namespace: default
  labels:
    test-label: test
managedFields:
- manager: kubectl
  operation: Apply
  apiVersion: v1
  fields:
    f:metadata:
      f:labels:
        f:test-label: {}
- manager: kube-controller-manager
  operation: Update
  apiVersion: v1
  time: '2019-03-30T16:00:00.000Z'
  fields:
    f:data:
      f:key: {}
data:
  key: new value
```

In this example, a second operation was run as an **update** by the manager called `kube-controller-manager`. The update request succeeded and changed a value in the `data` field, which caused that field's management to change to the `kube-controller-manager`.

If this update has instead been attempted using Server-Side Apply, the request would have failed due to conflicting ownership.

## Merge strategy

The merging strategy, implemented with Server-Side Apply, provides a generally more stable object lifecycle. Server-Side Apply tries to merge fields based on the actor who manages them instead of overruling based on values. This way multiple actors can update the same object without causing unexpected interference.

When a user sends a *fully-specified intent* object to the Server-Side Apply endpoint, the server merges it with the live object favoring the value from the request body if it is specified in both places. If the set of items present in the applied config is not a superset of the items applied by the same user last time, each missing item not managed by any other applicers is removed. For more information about how an object's schema is used to make decisions when merging, see [sig.k8s.io/structured-merge-diff](https://sig.k8s.io/structured-merge-diff).

The Kubernetes API (and the Go code that implements that API for Kubernetes) allows defining *merge strategy markers*. These markers describe the merge strategy supported for fields within Kubernetes objects. For a `CustomResourceDefinition`, you can set these markers when you define the custom resource.

Golang marker	OpenAPI extension	Possible values	Description
---------------	-------------------	-----------------	-------------

Golang marker	OpenAPI extension	Possible values	Description
// +listType	x- kubernetes -list-type	atomic / set / map	Applicable to lists. <code>set</code> applies to lists that include only scalar elements. These elements must be unique. <code>map</code> applies to lists of nested types only. The key values (see <code>listMapKey</code> ) must be unique in the list. <code>atomic</code> can apply to any list. If configured as <code>atomic</code> , the entire list is replaced during merge. At any point in time, a single manager owns the list. If <code>set</code> or <code>map</code> , different managers can manage entries separately.
// +listMapKey	x- kubernetes -list-map- keys	List of field names, e.g. ["port", "protocol"]	Only applicable when <code>+listType=map</code> . A list of field names whose values uniquely identify entries in the list. While there can be multiple keys, <code>listMapKey</code> is singular because keys need to be specified individually in the Go type. The key fields must be scalars.
//+mapType	x- kubernetes -map-type	atomic / granular	Applicable to maps. <code>atomic</code> means that the map can only be entirely replaced by a single manager. <code>granular</code> means that the map supports separate managers updating individual fields.
// +structType	x- kubernetes -map-type	atomic / granular	Applicable to structs; otherwise same usage and OpenAPI annotation as <code>//+mapType</code> .

If `listType` is missing, the API server interprets a `patchStrategy=merge` marker as a `listType=map` and the corresponding `patchMergeKey` marker as a `listMapKey`.

The `atomic` list type is recursive.

(In the [Go](#) code for Kubernetes, these markers are specified as comments and code authors need not repeat them as field tags).

## Custom resources and Server-Side Apply

By default, Server-Side Apply treats custom resources as unstructured data. All keys are treated the same as struct fields, and all lists are considered atomic.

If the CustomResourceDefinition defines a [schema](#) that contains

annotations as defined in the previous [Merge Strategy](#) section, these annotations will be used when merging objects of this type.

## Compatibility across topology changes

On rare occurrences, the author for a CustomResourceDefinition (CRD) or built-in may want to change the specific topology of a field in their resource, without incrementing its API version. Changing the topology of types, by upgrading the cluster or updating the CRD, has different consequences when updating existing objects. There are two categories of changes: when a field goes from `map` / `set` / `granular` to `atomic`, and the other way around.

When the `listType`, `mapType`, or `structType` changes from `map` / `set` / `granular` to `atomic`, the whole list, map, or struct of existing objects will end-up being owned by actors who owned an element of these types. This means that any further change to these objects would cause a conflict.

When a `listType`, `mapType`, or `structType` changes from `atomic` to `map` / `set` / `granular`, the API server is unable to infer the new ownership of these fields. Because of that, no conflict will be produced when objects have these fields updated. For that reason, it is not recommended to change a type from `atomic` to `map` / `set` / `granular`.

Take for example, the custom resource:

```
---
apiVersion: example.com/v1
kind: Foo
metadata:
  name: foo-sample
  managedFields:
    - manager: "manager-one"
      operation: Apply
      apiVersion: example.com/v1
      fields:
        f:spec:
          f:data: {}
spec:
  data:
    key1: val1
    key2: val2
```

Before `spec.data` gets changed from `atomic` to `granular`, `manager-one` owns the field `spec.data`, and all the fields within it (`key1` and `key2`). When the CRD gets changed to make `spec.data` `granular`, `manager-one` continues to own the top-level field `spec.data` (meaning no other managers can delete the map called `data` without a conflict), but it no longer owns `key1` and `key2`, so another manager can then modify or delete those fields without conflict.

## Using Server-Side Apply in a controller

As a developer of a controller, you can use Server-Side Apply as a way to simplify the update logic of your controller. The main differences with a read-modify-write and/or patch are the following:

- the applied object must contain all the fields that the controller

cares about.

- there is no way to remove fields that haven't been applied by the controller before (controller can still send a **patch** or **update** for these use-cases).
- the object doesn't have to be read beforehand; `resourceVersion` doesn't have to be specified.

It is strongly recommended for controllers to always force conflicts on objects that they own and manage, since they might not be able to resolve or act on these conflicts.

## Transferring ownership

In addition to the concurrency controls provided by [conflict resolution](#), Server-Side Apply provides ways to perform coordinated field ownership transfers from users to controllers.

This is best explained by example. Let's look at how to safely transfer ownership of the `replicas` field from a user to a controller while enabling automatic horizontal scaling for a Deployment, using the `HorizontalPodAutoscaler` resource and its accompanying controller.

Say a user has defined Deployment with `replicas` set to the desired value:

```
application/ssa/nginx-deployment.yaml 
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
```

And the user has created the Deployment using Server-Side Apply, like so:

```
kubectl apply -f https://k8s.io/examples/application/ssa/nginx-deploy
```

Then later, automatic scaling is enabled for the Deployment; for example:

```
kubectl autoscale deployment nginx-deployment --cpu-percent=50 --min=
```

Now, the user would like to remove `replicas` from their configuration, so they don't accidentally fight with the HorizontalPodAutoscaler (HPA) and its controller. However, there is a race: it might take some time before the HPA feels the need to adjust `.spec.replicas`; if the user removes `.spec.replicas` before the HPA writes to the field and becomes its owner, then the API server would set `.spec.replicas` to 1 (the default replica count for Deployment). This is not what the user wants to happen, even temporarily - it might well degrade a running workload.

There are two solutions:

- (basic) Leave `replicas` in the configuration; when the HPA eventually writes to that field, the system gives the user a conflict over it. At that point, it is safe to remove from the configuration.
- (more advanced) If, however, the user doesn't want to wait, for example because they want to keep the cluster legible to their colleagues, then they can take the following steps to make it safe to remove `replicas` from their configuration:

First, the user defines a new manifest containing only the `replicas` field:

```
# Save this file as 'nginx-deployment-replicas-only.yaml'.
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
```

#### Note:

The YAML file for SSA in this case only contains the fields you want to change. You are not supposed to provide a fully compliant Deployment manifest if you only want to modify the `spec.replicas` field using SSA.

The user applies that manifest using a private field manager name. In this example, the user picked `handover-to-hpa` :

```
kubectl apply -f nginx-deployment-replicas-only.yaml \
--server-side --field-manager=handover-to-hpa \
--validate=false
```

If the apply results in a conflict with the HPA controller, then do nothing. The conflict indicates the controller has claimed the field earlier in the process than it sometimes does.

At this point the user may remove the `replicas` field from their manifest:

[application/ssa/nginx-deployment-no-replicas.yaml](#) 

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
```

Note that whenever the HPA controller sets the `replicas` field to a new value, the temporary field manager will no longer own any fields and will be automatically deleted. No further clean up is required.

## Transferring ownership between managers

Field managers can transfer ownership of a field between each other by setting the field to the same value in both of their applied configurations, causing them to share ownership of the field. Once the managers share ownership of the field, one of them can remove the field from their applied configuration to give up ownership and complete the transfer to the other field manager.

## Comparison with Client-Side Apply

Server-Side Apply is meant both as a replacement for the original client-side implementation of the `kubectl apply` subcommand, and as simple and effective mechanism for controllers to enact their changes.

Compared to the `last-applied` annotation managed by `kubectl`, Server-Side Apply uses a more declarative approach, which tracks an object's field management, rather than a user's last applied state. This means that as a side effect of using Server-Side Apply, information about which field manager manages each field in an object also becomes available.

A consequence of the conflict detection and resolution implemented by Server-Side Apply is that an applier always has up to date field values in their local state. If they don't, they get a conflict the next time they apply. Any of the three options to resolve conflicts results in the applied configuration being an up to date subset of the object on the server's fields.

This is different from Client-Side Apply, where outdated values which have been overwritten by other users are left in an applier's local config. These values only become accurate when the user updates that specific field, if ever, and an applier has no way of knowing whether their next apply will overwrite other users' changes.

Another difference is that an applier using Client-Side Apply is unable to

change the API version they are using, but Server-Side Apply supports this use case.

## Migration between client-side and server-side apply

### Upgrading from client-side apply to server-side apply

Client-side apply users who manage a resource with `kubectl apply` can start using server-side apply with the following flag.

```
kubectl apply --server-side [--dry-run=server]
```

By default, field management of the object transfers from client-side apply to `kubectl` server-side apply, without encountering conflicts.

#### Caution:

Keep the `last-applied-configuration` annotation up to date. The annotation infers client-side applies managed fields. Any fields not managed by client-side apply raise conflicts.

For example, if you used `kubectl scale` to update the `replicas` field after client-side apply, then this field is not owned by client-side apply and creates conflicts on `kubectl apply --server-side`.

This behavior applies to server-side apply with the `kubectl` field manager. As an exception, you can opt-out of this behavior by specifying a different, non-default field manager, as seen in the following example. The default field manager for `kubectl` server-side apply is `kubectl`.

```
kubectl apply --server-side --field-manager=my-manager [--dry-run=ser
```

### Downgrading from server-side apply to client-side apply

If you manage a resource with `kubectl apply --server-side`, you can downgrade to client-side apply directly with `kubectl apply`.

Downgrading works because `kubectl` Server-Side Apply keeps the `last-applied-configuration` annotation up-to-date if you use `kubectl apply`.

This behavior applies to Server-Side Apply with the `kubectl` field manager. As an exception, you can opt-out of this behavior by specifying a different, non-default field manager, as seen in the following example. The default field manager for `kubectl` server-side apply is `kubectl`.

```
kubectl apply --server-side --field-manager=my-manager [--dry-run=ser
```

## API implementation

The `PATCH` verb for a resource that supports Server-Side Apply can accept the unofficial `application/apply-patch+yaml` content type. Users of Server-Side Apply can send partially specified objects as YAML as the

body of a `PATCH` request to the URI of a resource. When applying a configuration, you should always include all the fields that are important to the outcome (such as a desired state) that you want to define.

All JSON messages are valid YAML. Some clients specify Server-Side Apply requests using YAML request bodies that are also valid JSON.

## Access control and permissions

Since Server-Side Apply is a type of `PATCH`, a principal (such as a Role for Kubernetes RBAC) requires the **patch** permission to edit existing resources, and also needs the **create** verb permission in order to create new resources with Server-Side Apply.

## Clearing `managedFields`

It is possible to strip all `managedFields` from an object by overwriting them using a **patch** (JSON Merge Patch, Strategic Merge Patch, JSON Patch), or through an **update** (HTTP `PUT`); in other words, through every write operation other than **apply**. This can be done by overwriting the `managedFields` field with an empty entry. Two examples are:

```
PATCH /api/v1/namespaces/default/configmaps/example-cm
Accept: application/json
Content-Type: application/merge-patch+json

{
  "metadata": {
    "managedFields": [
      {}
    ]
  }
}
```

```
PATCH /api/v1/namespaces/default/configmaps/example-cm
Accept: application/json
Content-Type: application/json-patch+json
If-Match: 1234567890123456789

[{"op": "replace", "path": "/metadata/managedFields", "value": [{}]}]
```

This will overwrite the `managedFields` with a list containing a single empty entry that then results in the `managedFields` being stripped entirely from the object. Note that setting the `managedFields` to an empty list will not reset the field. This is on purpose, so `managedFields` never get stripped by clients not aware of the field.

In cases where the reset operation is combined with changes to other fields than the `managedFields`, this will result in the `managedFields` being reset first and the other changes being processed afterwards. As a result the applier takes ownership of any fields updated in the same request.

### Note:

Server-Side Apply does not correctly track ownership on sub-resources that don't receive the resource object type. If you are using Server-Side Apply with such a sub-resource, the changed fields may not be tracked.

## What's next

You can read about `managedFields` within the Kubernetes API reference for the [metadata](#) top level field.

## 2.3 - Client Libraries

This page contains an overview of the client libraries for using the Kubernetes API from various programming languages.

To write applications using the [Kubernetes REST API](#), you do not need to implement the API calls and request/response types yourself. You can use a client library for the programming language you are using.

Client libraries often handle common tasks such as authentication for you. Most client libraries can discover and use the Kubernetes Service Account to authenticate if the API client is running inside the Kubernetes cluster, or can understand the [kubeconfig file](#) format to read the credentials and the API Server address.

### Officially-supported Kubernetes client libraries

The following client libraries are officially maintained by [Kubernetes SIG API Machinery](#).

Language	Client Library	Sample Programs
C	<a href="https://github.com/kubernetes-client/c">github.com/kubernetes-client/c</a>	<a href="#">browse</a>
dotnet	<a href="https://github.com/kubernetes-client/csharp">github.com/kubernetes-client/csharp</a>	<a href="#">browse</a>
Go	<a href="https://github.com/kubernetes/client-go/">github.com/kubernetes/client-go/</a>	<a href="#">browse</a>
Haskell	<a href="https://github.com/kubernetes-client/haskell">github.com/kubernetes-client/haskell</a>	<a href="#">browse</a>
Java	<a href="https://github.com/kubernetes-client/java">github.com/kubernetes-client/java</a>	<a href="#">browse</a>
JavaScript	<a href="https://github.com/kubernetes-client/javascript">github.com/kubernetes-client/javascript</a>	<a href="#">browse</a>
Perl	<a href="https://github.com/kubernetes-client/perl/">github.com/kubernetes-client/perl/</a>	<a href="#">browse</a>
Python	<a href="https://github.com/kubernetes-client/python/">github.com/kubernetes-client/python/</a>	<a href="#">browse</a>
Ruby	<a href="https://github.com/kubernetes-client/ruby/">github.com/kubernetes-client/ruby/</a>	<a href="#">browse</a>

### Community-maintained client libraries

**Note:** This section links to third party projects that provide functionality required by Kubernetes. The Kubernetes project authors aren't responsible for these projects, which are listed alphabetically. To add a project to this list, read the [content guide](#) before submitting a change. [More information](#).

The following Kubernetes API client libraries are provided and maintained by their authors, not the Kubernetes team.

Language	Client Library
----------	----------------

Language	Client Library
Clojure	<a href="https://github.com/yanatan16/clj-kubernetes-api">github.com/yanatan16/clj-kubernetes-api</a>
DotNet	<a href="https://github.com/tonnyeremin/kubernetes_gen">github.com/tonnyeremin/kubernetes_gen</a>
DotNet (RestSharp)	<a href="https://github.com/masroorhasan/Kubernetes.DotNet">github.com/masroorhasan/Kubernetes.DotNet</a>
Elixir	<a href="https://github.com/obmarg/kazan">github.com/obmarg/kazan</a>
Elixir	<a href="https://github.com/coryodaniel/k8s">github.com/coryodaniel/k8s</a>
Java (OSGi)	<a href="https://bitbucket.org/amdatulabs/amdatu-kubernetes">bitbucket.org/amdatulabs/amdatu-kubernetes</a>
Java (Fabric8, OSGi)	<a href="https://github.com/fabric8io/kubernetes-client">github.com/fabric8io/kubernetes-client</a>
Java	<a href="https://github.com/manusa/yakc">github.com/manusa/yakc</a>
Lisp	<a href="https://github.com/brendandburns/cl-k8s">github.com/brendandburns/cl-k8s</a>
Lisp	<a href="https://github.com/xh4/cube">github.com/xh4/cube</a>
Node.js (TypeScript)	<a href="https://github.com/Goyoo/node-k8s-client">github.com/Goyoo/node-k8s-client</a>
Node.js	<a href="https://github.com/ajpauwels/easy-k8s">github.com/ajpauwels/easy-k8s</a>
Node.js	<a href="https://github.com/godaddy/kubernetes-client">github.com/godaddy/kubernetes-client</a>
Node.js	<a href="https://github.com/tenxcloud/node-kubernetes-client">github.com/tenxcloud/node-kubernetes-client</a>
Perl	<a href="https://metacpan.org/pod/Net::Kubernetes">metacpan.org/pod/Net::Kubernetes</a>
PHP	<a href="https://github.com/allansun/kubernetes-php-client">github.com/allansun/kubernetes-php-client</a>
PHP	<a href="https://github.com/maclof/kubernetes-client">github.com/maclof/kubernetes-client</a>
PHP	<a href="https://github.com/travisghansen/kubernetes-client-php">github.com/travisghansen/kubernetes-client-php</a>
PHP	<a href="https://github.com/renoki-co/php-k8s">github.com/renoki-co/php-k8s</a>
Python	<a href="https://github.com/fiaas/k8s">github.com/fiaas/k8s</a>
Python	<a href="https://github.com/gt-system/lightkube">github.com/gt-system/lightkube</a>
Python	<a href="https://github.com/kr8s-org/kr8s">github.com/kr8s-org/kr8s</a>
Python	<a href="https://github.com/mnubo/kubernetes-py">github.com/mnubo/kubernetes-py</a>
Python	<a href="https://github.com/tomplus/kubernetes asyncio">github.com/tomplus/kubernetes asyncio</a>
Python	<a href="https://github.com/Frankkkkk/pykorm">github.com/Frankkkkk/pykorm</a>
Ruby	<a href="https://github.com/abonas/kubeclient">github.com/abonas/kubeclient</a>
Ruby	<a href="https://github.com/k8s-ruby/k8s-ruby">github.com/k8s-ruby/k8s-ruby</a>
Ruby	<a href="https://github.com/kontena/k8s-client">github.com/kontena/k8s-client</a>
Rust	<a href="https://github.com/kube-rs/kube">github.com/kube-rs/kube</a>

Language	Client Library
Rust	<a href="https://github.com/ynqa/kubernetes-rust">github.com/ynqa/kubernetes-rust</a>
Scala	<a href="https://github.com/hagay3/skuber">github.com/hagay3/skuber</a>
Scala	<a href="https://github.com/hnaderi/scala-k8s">github.com/hnaderi/scala-k8s</a>
Scala	<a href="https://github.com/joan38/kubernetes-client">github.com/joan38/kubernetes-client</a>
Swift	<a href="https://github.com/swiftkube/client">github.com/swiftkube/client</a>

## 2.4 - Common Expression Language in Kubernetes

The [Common Expression Language \(CEL\)](#) is used in the Kubernetes API to declare validation rules, policy rules, and other constraints or conditions.

CEL expressions are evaluated directly in the [API server](#), making CEL a convenient alternative to out-of-process mechanisms, such as webhooks, for many extensibility use cases. Your CEL expressions continue to execute so long as the control plane's API server component remains available.

### Language overview

The [CEL language](#) has a straightforward syntax that is similar to the expressions in C, C++, Java, JavaScript and Go.

CEL was designed to be embedded into applications. Each CEL "program" is a single expression that evaluates to a single value. CEL expressions are typically short "one-liners" that inline well into the string fields of Kubernetes API resources.

Inputs to a CEL program are "variables". Each Kubernetes API field that contains CEL declares in the API documentation which variables are available to use for that field. For example, in the `x-kubernetes-validations[i].rules` field of CustomResourceDefinitions, the `self` and `oldSelf` variables are available and refer to the previous and current state of the custom resource data to be validated by the CEL expression. Other Kubernetes API fields may declare different variables. See the API documentation of the API fields to learn which variables are available for that field.

Example CEL expressions:

Rule	Purpose
<pre>self.minReplicas &lt;= self.replicas &amp;&amp; self.replicas &lt;= self.maxReplicas</pre>	Validate that the three fields defining replicas are ordered appropriately
<pre>'Available' in self.stateCounts</pre>	Validate that an entry with the 'Available' key exists in a map
<pre>(self.list1.size() == 0) != (self.list2.size() == 0)</pre>	Validate that one of two lists is non-empty, but not both
<pre>self.envars.filter(e, e.name = 'MY_ENV').all(e, e.value.matches('^[a-zA-Z]*\$'))</pre>	Validate the 'value' field of a listMap entry where key field 'name' is 'MY_ENV'
<pre>has(self.expired) &amp;&amp; self.created + self.ttl &lt; self.expired</pre>	Validate that 'expired' date is after a 'create' date plus a 'ttl' duration
<pre>self.health.startsWith('ok')</pre>	Validate a 'health' string field has the prefix 'ok'

Rule	Purpose
<code>self.widgets.exists(w, w.key == 'x' &amp;&amp; w.foo &lt; 10)</code>	Validate that the 'foo' property of a listMap item with a key 'x' is less than 10
<code>type(self) == string ? self == '99%' : self == 42</code>	Validate an int-or-string field for both the int and string cases
<code>self.metadata.name == 'singleton'</code>	Validate that an object's name matches a specific value (making it a singleton)
<code>self.set1.all(e, !(e in self.set2))</code>	Validate that two listSets are disjoint
<code>self.names.size() == self.details.size() &amp;&amp; self.names.all(n, n in self.details)</code>	Validate the 'details' map is keyed by the items in the 'names' listSet
<code>self.details.all(key, key.matches('^[a-zA-Z]*\$'))</code>	Validate the keys of the 'details' map
<code>self.details.all(key, self.details[key].matches('^[a-zA-Z]*\$'))</code>	Validate the values of the 'details' map

## CEL options, language features, and libraries

CEL is configured with the following options, libraries and language features, introduced at the specified Kubernetes versions:

CEL option, library or language feature	Included	Availability
<a href="#">Standard macros</a>	<code>has</code> , <code>all</code> , <code>exists</code> , <code>exists_one</code> , <code>map</code> , <code>filter</code>	All Kubernetes versions
<a href="#">Standard functions</a>	See <a href="#">official list of standard definitions</a>	All Kubernetes versions
<a href="#">Homogeneous Aggregate Literals</a>		All Kubernetes versions
<a href="#">Default UTC Time Zone</a>		All Kubernetes versions
<a href="#">Eagerly Validate Declarations</a>		All Kubernetes versions

CEL option, library or language feature	Included	Availability
<a href="#">extended strings library</a> , Version 1	<code>charAt</code> , <code>indexOf</code> , <code>lastIndexOf</code> , <code>lowerAscii</code> , <code>upperAscii</code> , <code>replace</code> , <code>split</code> , <code>join</code> , <code>substring</code> , <code>trim</code>	All Kubernetes versions
Kubernetes list library	See <a href="#">Kubernetes list library</a>	All Kubernetes versions
Kubernetes regex library	See <a href="#">Kubernetes regex library</a>	All Kubernetes versions
Kubernetes URL library	See <a href="#">Kubernetes URL library</a>	All Kubernetes versions
Kubernetes authorizer library	See <a href="#">Kubernetes authorizer library</a>	All Kubernetes versions
Kubernetes quantity library	See <a href="#">Kubernetes quantity library</a>	Kubernetes versions 1.29+
CEL optional types	See <a href="#">CEL optional types</a>	Kubernetes versions 1.29+
CEL CrossTypeNumericComparisons	See <a href="#">CEL</a> <a href="#">CrossTypeNumericComparisons</a>	Kubernetes versions 1.29+

CEL functions, features and language settings support Kubernetes control plane rollbacks. For example, *CEL Optional Values* was introduced at Kubernetes 1.29 and so only API servers at that version or newer will accept write requests to CEL expressions that use *CEL Optional Values*. However, when a cluster is rolled back to Kubernetes 1.28 CEL expressions using "CEL Optional Values" that are already stored in API resources will continue to evaluate correctly.

## Kubernetes CEL libraries

In addition to the CEL community libraries, Kubernetes includes CEL libraries that are available everywhere CEL is used in Kubernetes.

### Kubernetes list library

The list library includes `indexOf` and `lastIndexOf`, which work similar to the strings functions of the same names. These functions either the first or last positional index of the provided element in the list.

The list library also includes `min`, `max` and `sum`. Sum is supported on all number types as well as the duration type. Min and max are supported on all comparable types.

`isSorted` is also provided as a convenience function and is supported on

all comparable types.

Examples:

CEL Expression	Purpose
<code>names.isSorted()</code>	Verify that a list of names is kept in alphabetical order
<code>items.map(x, x.weight).sum() == 1.0</code>	Verify that the "weights" of a list of objects sum to 1.0
<code>lowPriorities.map(x, x.priority).max() &lt; highPriorities.map(x, x.priority).min()</code>	Verify that two sets of priorities do not overlap
<code>names.indexOf('should-be-first') == 1</code>	Require that the first name in a list is a specific value

See the [Kubernetes List Library](#) godoc for more information.

## Kubernetes regex library

In addition to the `matches` function provided by the CEL standard library, the regex library provides `find` and `findAll`, enabling a much wider range of regex operations.

Examples:

CEL Expression	Purpose
<code>"abc 123".find('[0-9]+')</code>	Find the first number in a string
<code>"1, 2, 3, 4".findAll('[0-9]+').map(x, int(x)).sum() &lt; 100</code>	Verify that the numbers in a string sum to less than 100

See the [Kubernetes regex library](#) godoc for more information.

## Kubernetes URL library

To make it easier and safer to process URLs, the following functions have been added:

- `isURL(string)` checks if a string is a valid URL according to the [Go's net/url](#) package. The string must be an absolute URL.
- `url(string)` URL converts a string to a URL or results in an error if the string is not a valid URL.

Once parsed via the `url` function, the resulting URL object has `getScheme`, `getHost`, `getHostname`, `getPort`, `getEscapedPath` and `getQuery` accessors.

Examples:

CEL Expression	Purpose
<code>url('https://example.com:80/').getHost()</code>	Get the 'example.com:80' host part of the URL.
<code>url('https://example.com/path with spaces/').getEscapedPath()</code>	Returns '/path%20with%20spaces/'

See the [Kubernetes URL library](#) godoc for more information.

## Kubernetes authorizer library

For CEL expressions in the API where a variable of type `Authorizer` is available, the authorizer may be used to perform authorization checks for the principal (authenticated user) of the request.

API resource checks are performed as follows:

1. Specify the group and resource to check:

```
Authorizer.group(string).resource(string) ResourceCheck
```

2. Optionally call any combination of the following builder functions to further narrow the authorization check. Note that these functions return the receiver type and can be chained:

- `ResourceCheck.subresource(string) ResourceCheck`
- `ResourceCheck.namespace(string) ResourceCheck`
- `ResourceCheck.name(string) ResourceCheck`

3. Call `ResourceCheck.check(verb string) Decision` to perform the authorization check.

4. Call `allowed() bool` or `reason() string` to inspect the result of the authorization check.

Non-resource authorization performed are used as follows:

1. specify only a path: `Authorizer.path(string) PathCheck`
2. Call `PathCheck.check(httpVerb string) Decision` to perform the authorization check.
3. Call `allowed() bool` or `reason() string` to inspect the result of the authorization check.

To perform an authorization check for a service account:

- `Authorizer.serviceAccount(namespace string, name string)`  
`Authorizer`

---

### CEL Expression

```
authorizer.group('').resource('pods').namespace('default').check('create').a
```

---

## CEL Expression

---

```
authorizer.path('/healthz').check('get').allowed()
```

---

```
authorizer.serviceAccount('default',  
'myserviceaccount').resource('deployments').check('delete').allowed()
```

### ⓘ FEATURE STATE: Kubernetes v1.31 [alpha]

With the alpha `AuthorizeWithSelectors` feature enabled, field and label selectors can be added to authorization checks.

---

## CEL Expression

---

```
authorizer.group('').resource('pods').fieldSelector('spec.nodeName=mynode').
```

---

```
authorizer.group('').resource('pods').labelSelector('example.com/  
mylabel=myvalue').check('list').allowed()
```

See the [Kubernetes Authz library](#) and [Kubernetes AuthzSelectors library](#) godoc for more information.

## Kubernetes quantity library

Kubernetes 1.28 adds support for manipulating quantity strings (ex 1.5G, 512k, 20Mi)

- `isQuantity(string)` checks if a string is a valid Quantity according to [Kubernetes' resource.Quantity](#).
- `quantity(string)` `Quantity` converts a string to a Quantity or results in an error if the string is not a valid quantity.

Once parsed via the `quantity` function, the resulting `Quantity` object has the following library of member functions:

Member Function	CEL Return Value	Description
isInteger()	bool	returns true if and only if asInteger is
		safe to call without an error
asInteger()	int	returns a representation of the current value as an int64 if possible or results in an error if conversion would result in
		overflow or loss of precision.
asApproximateFloat()	float	returns a float64 representation of the quantity which may lose precision. If the value of the quantity is outside the
		range of a float64 +Inf/-Inf will be returned.
sign()	int	Returns 1 if the quantity is positive, -1 if it is negative. 0 if it is zero
add(<Quantity>)	Quantity	Returns sum of two quantities
add(<int>)	Quantity	Returns sum of quantity and an integer
sub(<Quantity>)	Quantity	Returns difference between two quantities
sub(<int>)	Quantity	Returns difference between a quantity and an integer
isLessThan(<Quantity>)	bool	Returns true if and only if the receiver is less than the operand
isGreaterThan(<Quantity>)	bool	Returns true if and only if the receiver is greater than the operand
compareTo(<Quantity>)	int	Compares receiver to operand and returns 0 if they are equal, 1 if the receiver is greater, or -1 if the receiver is less than the operand

## Examples:

CEL Expression	P
----------------	---

quantity("50k").add(quantity("20k"))	A q
quantity("50k").sub(20000)	S ai fr q
quantity("50k").add(20).sub(quantity("100k")).sub(-50000)	C a s in ai q
quantity("200M").compareTo(quantity("0.2G"))	C tv q
quantity("150Mi").isGreaterThan(quantity("100Mi"))	T q gi tr re
quantity("50M").isLessThan(quantity("100M"))	T q le tr re

## Type checking

CEL is a [gradually typed language](#).

Some Kubernetes API fields contain fully type checked CEL expressions. For example, [CustomResourceDefinitions Validation Rules](#) are fully type checked.

Some Kubernetes API fields contain partially type checked CEL expressions. A partially type checked expression is an expressions where some of the variables are statically typed but others are dynamically typed. For example, in the CEL expressions of [ValidatingAdmissionPolicies](#) the `request` variable is typed, but the `object` variable is dynamically typed. As a result, an expression containing `request.namex` would fail type checking because the `namex` field is not defined. However, `object.namex` would pass type checking even when the `namex` field is not defined for the resource kinds that `object` refers to, because `object` is dynamically typed.

The `has()` macro in CEL may be used in CEL expressions to check if a field of a dynamically typed variable is accessible before attempting to access the field's value. For example:

```
has(object.namex) ? object.namex == 'special' : request.name == 'spec
```

# Type system integration

OpenAPIv3 type	CEL type
'object' with Properties	object / "message type" ( type(<object>) evaluates to selfType<uniqueNumber>.path.to.object.from.self
'object' with AdditionalProperties	map
'object' with x-kubernetes-embedded-type	object / "message type", 'apiVersion', 'kind', 'metadata.name' and 'metadata.generateName' are implicitly included in schema
'object' with x-kubernetes-preserve-unknown-fields	object / "message type", unknown fields are NOT accessible in CEL expression
x-kubernetes-int-or-string	union of int or string, self.intOrString < 100    self.intOrString == '50%' evaluates to true for both 50 and "50%"
'array'	list
'array' with x-kubernetes-list-type=map	list with map based Equality & unique key guarantees
'array' with x-kubernetes-list-type=set	list with set based Equality & unique entry guarantees
'boolean'	boolean
'number' (all formats)	double
'integer' (all formats)	int (64)
<i>no equivalent</i>	uint (64)
'null'	null_type
'string'	string
'string' with format=byte (base64 encoded)	bytes
'string' with format=date	timestamp (google.protobuf.Timestamp)
'string' with format=datetime	timestamp (google.protobuf.Timestamp)
'string' with format=duration	duration (google.protobuf.Duration)

Also see: [CEL types](#), [OpenAPI types](#), [Kubernetes Structural Schemas](#).

Equality comparison for arrays with x-kubernetes-list-type of set or

`map` ignores element order. For example `[1, 2] == [2, 1]` if the arrays represent Kubernetes `set` values.

Concatenation on arrays with `x-kubernetes-list-type` use the semantics of the list type:

- `set` :  $x + y$  performs a union where the array positions of all elements in  $x$  are preserved and non-intersecting elements in  $y$  are appended, retaining their partial order.
- `map` :  $x + y$  performs a merge where the array positions of all keys in  $x$  are preserved but the values are overwritten by values in  $y$  when the key sets of  $x$  and  $y$  intersect. Elements in  $y$  with non-intersecting keys are appended, retaining their partial order.

## Escaping

Only Kubernetes resource property names of the form `[a-zA-Z_.-/][a-zA-Z0-9_.-/*]` are accessible from CEL. Accessible property names are escaped according to the following rules when accessed in the expression:

escape sequence	property name equivalent
<code>__underscores__</code>	<code>_</code>
<code>__dot__</code>	<code>.</code>
<code>__dash__</code>	<code>-</code>
<code>__slash__</code>	<code>/</code>
<code>__{keyword}__</code>	<a href="#">CEL RESERVED keyword</a>

When you escape any of CEL's **RESERVED** keywords you need to match the exact property name use the underscore escaping (for example, `int` in the word `sprint` would not be escaped and nor would it need to be).

Examples on escaping:

property name	rule with escaped property name
<code>namespace</code>	<code>self.__namespace__ &gt; 0</code>
<code>x-prop</code>	<code>self.x__dash__prop &gt; 0</code>
<code>redact__d</code>	<code>self.redact__underscores__d &gt; 0</code>
<code>string</code>	<code>self.startsWith('kube')</code>

## Resource constraints

CEL is non-Turing complete and offers a variety of production safety controls to limit execution time. CEL's *resource constraint* features provide feedback to developers about expression complexity and help protect the API server from excessive resource consumption during evaluation. CEL's resource constraint features are used to prevent CEL evaluation from consuming excessive API server resources.

A key element of the resource constraint features is a *cost unit* that CEL defines as a way of tracking CPU utilization. Cost units are independent of system load and hardware. Cost units are also deterministic; for any given CEL expression and input data, evaluation of the expression by the CEL interpreter will always result in the same cost.

Many of CEL's core operations have fixed costs. The simplest operations, such as comparisons (e.g. `<`) have a cost of 1. Some have a higher fixed cost, for example list literal declarations have a fixed base cost of 40 cost units.

Calls to functions implemented in native code approximate cost based on the time complexity of the operation. For example: operations that use regular expressions, such as `match` and `find`, are estimated using an approximated cost of `length(regexString)*length(inputString)`. The approximated cost reflects the worst case time complexity of Go's RE2 implementation.

## Runtime cost budget

All CEL expressions evaluated by Kubernetes are constrained by a runtime cost budget. The runtime cost budget is an estimate of actual CPU utilization computed by incrementing a cost unit counter while interpreting a CEL expression. If the CEL interpreter executes too many instructions, the runtime cost budget will be exceeded, execution of the expressions will be halted, and an error will result.

Some Kubernetes resources define an additional runtime cost budget that bounds the execution of multiple expressions. If the sum total of the cost of expressions exceed the budget, execution of the expressions will be halted, and an error will result. For example the validation of a custom resource has a *per-validation* runtime cost budget for all [Validation Rules](#) evaluated to validate the custom resource.

## Estimated cost limits

For some Kubernetes resources, the API server may also check if worst case estimated running time of CEL expressions would be prohibitively expensive to execute. If so, the API server prevent the CEL expression from being written to API resources by rejecting create or update operations containing the CEL expression to the API resources. This feature offers a stronger assurance that CEL expressions written to the API resource will evaluate at runtime without exceeding the runtime cost budget.

# 2.5 - Kubernetes Deprecation Policy

This document details the deprecation policy for various facets of the system.

Kubernetes is a large system with many components and many contributors. As with any such software, the feature set naturally evolves over time, and sometimes a feature may need to be removed. This could include an API, a flag, or even an entire feature. To avoid breaking existing users, Kubernetes follows a deprecation policy for aspects of the system that are slated to be removed.

## Deprecating parts of the API

Since Kubernetes is an API-driven system, the API has evolved over time to reflect the evolving understanding of the problem space. The Kubernetes API is actually a set of APIs, called "API groups", and each API group is independently versioned. [API versions](#) fall into 3 main tracks, each of which has different policies for deprecation:

Example	Track
v1	GA (generally available, stable)
v1beta1	Beta (pre-release)
v1alpha1	Alpha (experimental)

A given release of Kubernetes can support any number of API groups and any number of versions of each.

The following rules govern the deprecation of elements of the API. This includes:

- REST resources (aka API objects)
- Fields of REST resources
- Annotations on REST resources, including "beta" annotations but not including "alpha" annotations.
- Enumerated or constant values
- Component config structures

These rules are enforced between official releases, not between arbitrary commits to master or release branches.

### Rule #1: API elements may only be removed by incrementing the version of the API group.

Once an API element has been added to an API group at a particular version, it can not be removed from that version or have its behavior significantly changed, regardless of track.

#### Note:

For historical reasons, there are 2 "monolithic" API groups - "core" (no group name) and "extensions". Resources will incrementally be moved from these legacy API groups into more domain-specific API groups.

**Rule #2: API objects must be able to round-trip between API versions in a given release without information loss, with the exception of whole REST resources that do not exist in some versions.**

For example, an object can be written as v1 and then read back as v2 and converted to v1, and the resulting v1 resource will be identical to the original. The representation in v2 might be different from v1, but the system knows how to convert between them in both directions.

Additionally, any new field added in v2 must be able to round-trip to v1 and back, which means v1 might have to add an equivalent field or represent it as an annotation.

**Rule #3: An API version in a given track may not be deprecated in favor of a less stable API version.**

- GA API versions can replace beta and alpha API versions.
- Beta API versions can replace earlier beta and alpha API versions, but *may not* replace GA API versions.
- Alpha API versions can replace earlier alpha API versions, but *may not* replace GA or beta API versions.

**Rule #4a: API lifetime is determined by the API stability level**

- GA API versions may be marked as deprecated, but must not be removed within a major version of Kubernetes
- Beta API versions are deprecated no more than 9 months or 3 minor releases after introduction (whichever is longer), and are no longer served 9 months or 3 minor releases after deprecation (whichever is longer)
- Alpha API versions may be removed in any release without prior deprecation notice

This ensures beta API support covers the [maximum supported version skew of 2 releases](#), and that APIs don't stagnate on unstable beta versions, accumulating production usage that will be disrupted when support for the beta API ends.

**Note:**

There are no current plans for a major version revision of Kubernetes that removes GA APIs.

**Note:**

Until [#52185](#) is resolved, no API versions that have been persisted to storage may be removed. Serving REST endpoints for those versions may be disabled (subject to the deprecation timelines in this document), but the API server must remain capable of decoding/ converting previously persisted data from storage.

**Rule #4b: The "preferred" API version and the "storage version" for a given group may not advance until after a release has been made that supports both the new version and the previous version**

Users must be able to upgrade to a new release of Kubernetes and then roll back to a previous release, without converting anything to the new API version or suffering breakages (unless they explicitly used features only available in the newer version). This is particularly evident in the stored representation of objects.

All of this is best illustrated by examples. Imagine a Kubernetes release, version X, which introduces a new API group. A new Kubernetes release is made every approximately 4 months (3 per year). The following table

describes which API versions are supported in a series of subsequent releases.

Release	API Versions	Preferred/ Storage Version	Notes
X	v1alpha1	v1alpha1	
X+1	v1alpha2	v1alpha2	<ul style="list-style-type: none"><li>• v1alpha1 is removed, "action required" relnote</li></ul>
X+2	v1beta1	v1beta1	<ul style="list-style-type: none"><li>• v1alpha2 is removed, "action required" relnote</li></ul>
X+3	v1beta2, v1beta1 (deprecated)	v1beta1	<ul style="list-style-type: none"><li>• v1beta1 is deprecated, "action required" relnote</li></ul>
X+4	v1beta2, v1beta1 (deprecated)	v1beta2	
X+5	v1, v1beta1 (deprecated), v1beta2 (deprecated)	v1beta2	<ul style="list-style-type: none"><li>• v1beta2 is deprecated, "action required" relnote</li></ul>
X+6	v1, v1beta2 (deprecated)	v1	<ul style="list-style-type: none"><li>• v1beta1 is removed, "action required" relnote</li></ul>
X+7	v1, v1beta2 (deprecated)	v1	
X+8	v2alpha1, v1	v1	<ul style="list-style-type: none"><li>• v1beta2 is removed, "action required" relnote</li></ul>
X+9	v2alpha2, v1	v1	<ul style="list-style-type: none"><li>• v2alpha1 is removed, "action required" relnote</li></ul>
X+10	v2beta1, v1	v1	<ul style="list-style-type: none"><li>• v2alpha2 is removed, "action required" relnote</li></ul>
X+11	v2beta2, v2beta1 (deprecated), v1	v1	<ul style="list-style-type: none"><li>• v2beta1 is deprecated, "action required" relnote</li></ul>

Release	API Versions	Preferred/ Storage Version	Notes
X+12	v2, v2beta2 (deprecated), v2beta1 (deprecated), v1 (deprecated)	v1	<ul style="list-style-type: none"> <li>v2beta2 is deprecated, "action required" relnote</li> <li>v1 is deprecated in favor of v2, but will not be removed</li> </ul>
X+13	v2, v2beta1 (deprecated), v2beta2 (deprecated), v1 (deprecated)	v2	
X+14	v2, v2beta2 (deprecated), v1 (deprecated)	v2	<ul style="list-style-type: none"> <li>v2beta1 is removed, "action required" relnote</li> </ul>
X+15	v2, v1 (deprecated)	v2	<ul style="list-style-type: none"> <li>v2beta2 is removed, "action required" relnote</li> </ul>

## REST resources (aka API objects)

Consider a hypothetical REST resource named Widget, which was present in API v1 in the above timeline, and which needs to be deprecated. We document and [announce](#) the deprecation in sync with release X+1. The Widget resource still exists in API version v1 (deprecated) but not in v2alpha1. The Widget resource continues to exist and function in releases up to and including X+8. Only in release X+9, when API v1 has aged out, does the Widget resource cease to exist, and the behavior get removed.

Starting in Kubernetes v1.19, making an API request to a deprecated REST API endpoint:

1. Returns a `Warning` header (as defined in [RFC7234, Section 5.5](#)) in the API response.
2. Adds a `"k8s.io/deprecated": "true"` annotation to the [audit event](#) recorded for the request.
3. Sets an `apiserver_requested_deprecated_apis` gauge metric to `1` in the `kube-apiserver` process. The metric has labels for `group`, `version`, `resource`, `subresource` that can be joined to the `apiserver_request_total` metric, and a `removed_release` label that indicates the Kubernetes release in which the API will no longer be served. The following Prometheus query returns information about requests made to deprecated APIs which will be removed in v1.22:

```
apiserver_requested_deprecated_apis{removed_release="1.22"} * or
```

## Fields of REST resources

As with whole REST resources, an individual field which was present in API v1 must exist and function until API v1 is removed. Unlike whole resources, the v2 APIs may choose a different representation for the

field, as long as it can be round-tripped. For example a v1 field named "magnitude" which was deprecated might be named "deprecatedMagnitude" in API v2. When v1 is eventually removed, the deprecated field can be removed from v2.

## Enumerated or constant values

As with whole REST resources and fields thereof, a constant value which was supported in API v1 must exist and function until API v1 is removed.

## Component config structures

Component configs are versioned and managed similar to REST resources.

## Future work

Over time, Kubernetes will introduce more fine-grained API versions, at which point these rules will be adjusted as needed.

# Deprecating a flag or CLI

The Kubernetes system is comprised of several different programs cooperating. Sometimes, a Kubernetes release might remove flags or CLI commands (collectively "CLI elements") in these programs. The individual programs naturally sort into two main groups - user-facing and admin-facing programs, which vary slightly in their deprecation policies. Unless a flag is explicitly prefixed or documented as "alpha" or "beta", it is considered GA.

CLI elements are effectively part of the API to the system, but since they are not versioned in the same way as the REST API, the rules for deprecation are as follows:

**Rule #5a: CLI elements of user-facing components (e.g. kubectl) must function after their announced deprecation for no less than:**

- **GA: 12 months or 2 releases (whichever is longer)**
- **Beta: 3 months or 1 release (whichever is longer)**
- **Alpha: 0 releases**

**Rule #5b: CLI elements of admin-facing components (e.g. kubelet) must function after their announced deprecation for no less than:**

- **GA: 6 months or 1 release (whichever is longer)**
- **Beta: 3 months or 1 release (whichever is longer)**
- **Alpha: 0 releases**

**Rule #5c: Command line interface (CLI) elements cannot be deprecated in favor of less stable CLI elements**

Similar to the Rule #3 for APIs, if an element of a command line interface is being replaced with an alternative implementation, such as by renaming an existing element, or by switching to use configuration sourced from a file instead of a command line argument, that recommended alternative must be of the same or higher stability level.

**Rule #6: Deprecated CLI elements must emit warnings (optionally disable) when used.**

# Deprecating a feature or behavior

Occasionally a Kubernetes release needs to deprecate some feature or behavior of the system that is not controlled by the API or CLI. In this case, the rules for deprecation are as follows:

## **Rule #7: Deprecated behaviors must function for no less than 1 year after their announced deprecation.**

If the feature or behavior is being replaced with an alternative implementation that requires work to adopt the change, there should be an effort to simplify the transition whenever possible. If an alternative implementation is under Kubernetes organization control, the following rules apply:

## **Rule #8: The feature or behavior must not be deprecated in favor of an alternative implementation that is less stable**

For example, a generally available feature cannot be deprecated in favor of a Beta replacement. The Kubernetes project does, however, encourage users to adopt and transition to alternative implementations even before they reach the same maturity level. This is particularly important for exploring new use cases of a feature or getting an early feedback on the replacement.

Alternative implementations may sometimes be external tools or products, for example a feature may move from the kubelet to container runtime that is not under Kubernetes project control. In such cases, the rule cannot be applied, but there must be an effort to ensure that there is a transition path that does not compromise on components' maturity levels. In the example with container runtimes, the effort may involve trying to ensure that popular container runtimes have versions that offer the same level of stability while implementing that replacement behavior.

Deprecation rules for features and behaviors do not imply that all changes to the system are governed by this policy. These rules apply only to significant, user-visible behaviors which impact the correctness of applications running on Kubernetes or that impact the administration of Kubernetes clusters, and which are being removed entirely.

An exception to the above rule is *feature gates*. Feature gates are key=value pairs that allow for users to enable/disable experimental features.

Feature gates are intended to cover the development life cycle of a feature - they are not intended to be long-term APIs. As such, they are expected to be deprecated and removed after a feature becomes GA or is dropped.

As a feature moves through the stages, the associated feature gate evolves. The feature life cycle matched to its corresponding feature gate is:

- Alpha: the feature gate is disabled by default and can be enabled by the user.
- Beta: the feature gate is enabled by default and can be disabled by the user.
- GA: the feature gate is deprecated (see "[Deprecation](#)") and becomes non-operational.
- GA, deprecation window complete: the feature gate is removed and calls to it are no longer accepted.

## Deprecation

Features can be removed at any point in the life cycle prior to GA. When features are removed prior to GA, their associated feature gates are also deprecated.

When an invocation tries to disable a non-operational feature gate, the call fails in order to avoid unsupported scenarios that might otherwise run silently.

In some cases, removing pre-GA features requires considerable time. Feature gates can remain operational until their associated feature is fully removed, at which point the feature gate itself can be deprecated.

When removing a feature gate for a GA feature also requires considerable time, calls to feature gates may remain operational if the feature gate has no effect on the feature, and if the feature gate causes no errors.

Features intended to be disabled by users should include a mechanism for disabling the feature in the associated feature gate.

Versioning for feature gates is different from the previously discussed components, therefore the rules for deprecation are as follows:

**Rule #9: Feature gates must be deprecated when the corresponding feature they control transitions a lifecycle stage as follows. Feature gates must function for no less than:**

- Beta feature to GA: 6 months or 2 releases (whichever is longer)
- Beta feature to EOL: 3 months or 1 release (whichever is longer)
- Alpha feature to EOL: 0 releases

**Rule #10: Deprecated feature gates must respond with a warning when used. When a feature gate is deprecated it must be documented in both in the release notes and the corresponding CLI help. Both warnings and documentation must indicate whether a feature gate is non-operational.**

## Deprecating a metric

Each component of the Kubernetes control-plane exposes metrics (usually the `/metrics` endpoint), which are typically ingested by cluster administrators. Not all metrics are the same: some metrics are commonly used as SLIs or used to determine SLOs, these tend to have greater import. Other metrics are more experimental in nature or are used primarily in the Kubernetes development process.

Accordingly, metrics fall under three stability classes ( `ALPHA` , `BETA` `STABLE` ); this impacts removal of a metric during a Kubernetes release. These classes are determined by the perceived importance of the metric. The rules for deprecating and removing a metric are as follows:

**Rule #11a: Metrics, for the corresponding stability class, must function for no less than:**

- **STABLE: 4 releases or 12 months (whichever is longer)**
- **BETA: 2 releases or 8 months (whichever is longer)**
- **ALPHA: 0 releases**

**Rule #11b: Metrics, after their *announced deprecation*, must function for no less than:**

- **STABLE: 3 releases or 9 months (whichever is longer)**
- **BETA: 1 releases or 4 months (whichever is longer)**
- **ALPHA: 0 releases**

Deprecated metrics will have their description text prefixed with a deprecation notice string '(Deprecated from x.y)' and a warning log will be emitted during metric registration. Like their stable undeprecated counterparts, deprecated metrics will be automatically registered to the metrics endpoint and therefore visible.

On a subsequent release (when the metric's `deprecatedVersion` is equal to `current_kubernetes_version - 3`), a deprecated metric will become a *hidden* metric. **Unlike** their deprecated counterparts, hidden metrics will *no longer* be automatically registered to the metrics endpoint (hence hidden). However, they can be explicitly enabled through a command line flag on the binary ( `--show-hidden-metrics-for-version=` ). This provides cluster admins an escape hatch to properly migrate off of a deprecated metric, if they were not able to react to the earlier deprecation warnings. Hidden metrics should be deleted after one release.

## Exceptions

No policy can cover every possible situation. This policy is a living document, and will evolve over time. In practice, there will be situations that do not fit neatly into this policy, or for which this policy becomes a serious impediment. Such situations should be discussed with SIGs and project leaders to find the best solutions for those specific cases, always bearing in mind that Kubernetes is committed to being a stable system that, as much as possible, never breaks users. Exceptions will always be announced in all relevant release notes.

## 2.6 - Deprecated API Migration Guide

As the Kubernetes API evolves, APIs are periodically reorganized or upgraded. When APIs evolve, the old API is deprecated and eventually removed. This page contains information you need to know when migrating from deprecated API versions to newer and more stable API versions.

### Removed APIs by release

#### v1.32

The **v1.32** release will stop serving the following deprecated API versions:

##### Flow control resources

The **flowcontrol.apiserver.k8s.io/v1beta3** API version of FlowSchema and PriorityLevelConfiguration will no longer be served in v1.32.

- Migrate manifests and API clients to use the **flowcontrol.apiserver.k8s.io/v1** API version, available since v1.29.
- All existing persisted objects are accessible via the new API
- Notable changes in **flowcontrol.apiserver.k8s.io/v1**:
  - The PriorityLevelConfiguration `spec.limited.nominalConcurrencyShares` field only defaults to 30 when unspecified, and an explicit value of 0 is not changed to 30.

#### v1.29

The **v1.29** release stopped serving the following deprecated API versions:

##### Flow control resources

The **flowcontrol.apiserver.k8s.io/v1beta2** API version of FlowSchema and PriorityLevelConfiguration is no longer served as of v1.29.

- Migrate manifests and API clients to use the **flowcontrol.apiserver.k8s.io/v1** API version, available since v1.29, or the **flowcontrol.apiserver.k8s.io/v1beta3** API version, available since v1.26.
- All existing persisted objects are accessible via the new API
- Notable changes in **flowcontrol.apiserver.k8s.io/v1**:
  - The PriorityLevelConfiguration `spec.limited.assuredConcurrencyShares` field is renamed to `spec.limited.nominalConcurrencyShares` and only defaults to 30 when unspecified, and an explicit value of 0 is not changed to 30.
- Notable changes in **flowcontrol.apiserver.k8s.io/v1beta3**:
  - The PriorityLevelConfiguration `spec.limited.assuredConcurrencyShares` field is renamed to `spec.limited.nominalConcurrencyShares`

#### v1.27

The **v1.27** release stopped serving the following deprecated API versions:

## CSIStorageCapacity

The **storage.k8s.io/v1beta1** API version of CSIStorageCapacity is no longer served as of v1.27.

- Migrate manifests and API clients to use the **storage.k8s.io/v1** API version, available since v1.24.
- All existing persisted objects are accessible via the new API
- No notable changes

## v1.26

The **v1.26** release stopped serving the following deprecated API versions:

### Flow control resources

The **flowcontrol.apiserver.k8s.io/v1beta1** API version of FlowSchema and PriorityLevelConfiguration is no longer served as of v1.26.

- Migrate manifests and API clients to use the **flowcontrol.apiserver.k8s.io/v1beta2** API version.
- All existing persisted objects are accessible via the new API
- No notable changes

### HorizontalPodAutoscaler

The **autoscaling/v2beta2** API version of HorizontalPodAutoscaler is no longer served as of v1.26.

- Migrate manifests and API clients to use the **autoscaling/v2** API version, available since v1.23.
- All existing persisted objects are accessible via the new API
- Notable changes:
  - `targetAverageUtilization` is replaced with `target.averageUtilization` and `target.type: Utilization`. See [Autoscaling on multiple metrics and custom metrics](#).

## v1.25

The **v1.25** release stopped serving the following deprecated API versions:

### CronJob

The **batch/v1beta1** API version of CronJob is no longer served as of v1.25.

- Migrate manifests and API clients to use the **batch/v1** API version, available since v1.21.
- All existing persisted objects are accessible via the new API
- No notable changes

### EndpointSlice

The **discovery.k8s.io/v1beta1** API version of EndpointSlice is no longer served as of v1.25.

- Migrate manifests and API clients to use the **discovery.k8s.io/v1** API version, available since v1.21.
- All existing persisted objects are accessible via the new API
- Notable changes in **discovery.k8s.io/v1**:
  - use per Endpoint `nodeName` field instead of deprecated

- topology["kubernetes.io/hostname"] field
  - use per Endpoint zone field instead of deprecated topology["topology.kubernetes.io/zone"] field
  - topology is replaced with the deprecatedTopology field which is not writable in v1

## Event

The **events.k8s.io/v1beta1** API version of Event is no longer served as of v1.25.

- Migrate manifests and API clients to use the **events.k8s.io/v1** API version, available since v1.19.
- All existing persisted objects are accessible via the new API
- Notable changes in **events.k8s.io/v1**:
  - type is limited to Normal and Warning
  - involvedObject is renamed to regarding
  - action, reason, reportingController, and reportingInstance are required when creating new **events.k8s.io/v1** Events
  - use eventTime instead of the deprecated firstTimestamp field (which is renamed to deprecatedFirstTimestamp and not permitted in new **events.k8s.io/v1** Events)
  - use series.lastObservedTime instead of the deprecated lastTimestamp field (which is renamed to deprecatedLastTimestamp and not permitted in new **events.k8s.io/v1** Events)
  - use series.count instead of the deprecated count field (which is renamed to deprecatedCount and not permitted in new **events.k8s.io/v1** Events)
  - use reportingController instead of the deprecated source.component field (which is renamed to deprecatedSource.component and not permitted in new **events.k8s.io/v1** Events)
  - use reportingInstance instead of the deprecated source.host field (which is renamed to deprecatedSource.host and not permitted in new **events.k8s.io/v1** Events)

## HorizontalPodAutoscaler

The **autoscaling/v2beta1** API version of HorizontalPodAutoscaler is no longer served as of v1.25.

- Migrate manifests and API clients to use the **autoscaling/v2** API version, available since v1.23.
- All existing persisted objects are accessible via the new API
- Notable changes:
  - targetAverageUtilization is replaced with target.averageUtilization and target.type: Utilization. See [Autoscaling on multiple metrics and custom metrics](#).

## PodDisruptionBudget

The **policy/v1beta1** API version of PodDisruptionBudget is no longer served as of v1.25.

- Migrate manifests and API clients to use the **policy/v1** API version, available since v1.21.
- All existing persisted objects are accessible via the new API
- Notable changes in **policy/v1**:

- an empty `spec.selector` ( `{}` ) written to a `policy/v1` PodDisruptionBudget selects all pods in the namespace (in `policy/v1beta1` an empty `spec.selector` selected no pods). An unset `spec.selector` selects no pods in either API version.

## PodSecurityPolicy

PodSecurityPolicy in the **policy/v1beta1** API version is no longer served as of v1.25, and the PodSecurityPolicy admission controller will be removed.

Migrate to [Pod Security Admission](#) or a [3rd party admission webhook](#). For a migration guide, see [Migrate from PodSecurityPolicy to the Built-In PodSecurity Admission Controller](#). For more information on the deprecation, see [PodSecurityPolicy Deprecation: Past, Present, and Future](#).

## RuntimeClass

RuntimeClass in the **node.k8s.io/v1beta1** API version is no longer served as of v1.25.

- Migrate manifests and API clients to use the **node.k8s.io/v1** API version, available since v1.20.
- All existing persisted objects are accessible via the new API
- No notable changes

## v1.22

The **v1.22** release stopped serving the following deprecated API versions:

### Webhook resources

The **admissionregistration.k8s.io/v1beta1** API version of MutatingWebhookConfiguration and ValidatingWebhookConfiguration is no longer served as of v1.22.

- Migrate manifests and API clients to use the **admissionregistration.k8s.io/v1** API version, available since v1.16.
- All existing persisted objects are accessible via the new APIs
- Notable changes:
  - `webhooks[*].failurePolicy` default changed from `Ignore` to `Fail` for v1
  - `webhooks[*].matchPolicy` default changed from `Exact` to `Equivalent` for v1
  - `webhooks[*].timeoutSeconds` default changed from `30s` to `10s` for v1
  - `webhooks[*].sideEffects` default value is removed, and the field made required, and only `None` and `NoneOnDryRun` are permitted for v1
  - `webhooks[*].admissionReviewVersions` default value is removed and the field made required for v1 (supported versions for AdmissionReview are `v1` and `v1beta1` )
  - `webhooks[*].name` must be unique in the list for objects created via `admissionregistration.k8s.io/v1`

### CustomResourceDefinition

The **apiextensions.k8s.io/v1beta1** API version of CustomResourceDefinition is no longer served as of v1.22.

- Migrate manifests and API clients to use the **apiextensions.k8s.io/v1** API version, available since v1.16.
- All existing persisted objects are accessible via the new API
- Notable changes:
  - `spec.scope` is no longer defaulted to `Namespaced` and must be explicitly specified
  - `spec.version` is removed in v1; use `spec.versions` instead
  - `spec.validation` is removed in v1; use `spec.versions[*].schema` instead
  - `spec.subresources` is removed in v1; use `spec.versions[*].subresources` instead
  - `spec.additionalPrinterColumns` is removed in v1; use `spec.versions[*].additionalPrinterColumns` instead
  - `spec.conversion.webhookClientConfig` is moved to `spec.conversion.webhook.clientConfig` in v1
  - `spec.conversion.conversionReviewVersions` is moved to `spec.conversion.webhook.conversionReviewVersions` in v1
  - `spec.versions[*].schema.openAPIV3Schema` is now required when creating v1 CustomResourceDefinition objects, and must be a [structural schema](#)
  - `spec.preserveUnknownFields: true` is disallowed when creating v1 CustomResourceDefinition objects; it must be specified within schema definitions as `x-kubernetes-preserve-unknown-fields: true`
  - In `additionalPrinterColumns` items, the `JSONPath` field was renamed to `jsonPath` in v1 (fixes [#66531](#))

## APIService

The **apiregistration.k8s.io/v1beta1** API version of APIService is no longer served as of v1.22.

- Migrate manifests and API clients to use the **apiregistration.k8s.io/v1** API version, available since v1.10.
- All existing persisted objects are accessible via the new API
- No notable changes

## TokenReview

The **authentication.k8s.io/v1beta1** API version of TokenReview is no longer served as of v1.22.

- Migrate manifests and API clients to use the **authentication.k8s.io/v1** API version, available since v1.6.
- No notable changes

## SubjectAccessReview resources

The **authorization.k8s.io/v1beta1** API version of LocalSubjectAccessReview, SelfSubjectAccessReview, SubjectAccessReview, and SelfSubjectRulesReview is no longer served as of v1.22.

- Migrate manifests and API clients to use the **authorization.k8s.io/v1** API version, available since v1.6.
- Notable changes:
  - `spec.group` was renamed to `spec.groups` in v1 (fixes [#32709](#))

## CertificateSigningRequest

The **certificates.k8s.io/v1beta1** API version of CertificateSigningRequest is no longer served as of v1.22.

- Migrate manifests and API clients to use the **certificates.k8s.io/v1** API version, available since v1.19.
- All existing persisted objects are accessible via the new API
- Notable changes in `certificates.k8s.io/v1` :
  - For API clients requesting certificates:
    - `spec.signerName` is now required (see [known Kubernetes signers](#)), and requests for `kubernetes.io/legacy-unknown` are not allowed to be created via the `certificates.k8s.io/v1` API
    - `spec.usages` is now required, may not contain duplicate values, and must only contain known usages
  - For API clients approving or signing certificates:
    - `status.conditions` may not contain duplicate types
    - `status.conditions[*].status` is now required
    - `status.certificate` must be PEM-encoded, and contain only `CERTIFICATE` blocks

## Lease

The **coordination.k8s.io/v1beta1** API version of Lease is no longer served as of v1.22.

- Migrate manifests and API clients to use the **coordination.k8s.io/v1** API version, available since v1.14.
- All existing persisted objects are accessible via the new API
- No notable changes

## Ingress

The **extensions/v1beta1** and **networking.k8s.io/v1beta1** API versions of Ingress is no longer served as of v1.22.

- Migrate manifests and API clients to use the **networking.k8s.io/v1** API version, available since v1.19.
- All existing persisted objects are accessible via the new API
- Notable changes:
  - `spec.backend` is renamed to `spec.defaultBackend`
  - The `backend serviceName` field is renamed to `service.name`
  - Numeric `backend servicePort` fields are renamed to `service.port.number`
  - String `backend servicePort` fields are renamed to `service.port.name`
  - `pathType` is now required for each specified path. Options are `Prefix`, `Exact`, and `ImplementationSpecific`. To match the undefined `v1beta1` behavior, use `ImplementationSpecific`.

## IngressClass

The **networking.k8s.io/v1beta1** API version of IngressClass is no longer served as of v1.22.

- Migrate manifests and API clients to use the **networking.k8s.io/v1** API version, available since v1.19.
- All existing persisted objects are accessible via the new API
- No notable changes

## RBAC resources

The **rbac.authorization.k8s.io/v1beta1** API version of ClusterRole, ClusterRoleBinding, Role, and RoleBinding is no longer served as of v1.22.

- Migrate manifests and API clients to use the **rbac.authorization.k8s.io/v1** API version, available since v1.8.
- All existing persisted objects are accessible via the new APIs
- No notable changes

## PriorityClass

The **scheduling.k8s.io/v1beta1** API version of PriorityClass is no longer served as of v1.22.

- Migrate manifests and API clients to use the **scheduling.k8s.io/v1** API version, available since v1.14.
- All existing persisted objects are accessible via the new API
- No notable changes

## Storage resources

The **storage.k8s.io/v1beta1** API version of CSIDriver, CSINode, StorageClass, and VolumeAttachment is no longer served as of v1.22.

- Migrate manifests and API clients to use the **storage.k8s.io/v1** API version
  - CSIDriver is available in **storage.k8s.io/v1** since v1.19.
  - CSINode is available in **storage.k8s.io/v1** since v1.17
  - StorageClass is available in **storage.k8s.io/v1** since v1.6
  - VolumeAttachment is available in **storage.k8s.io/v1** v1.13
- All existing persisted objects are accessible via the new APIs
- No notable changes

## v1.16

The **v1.16** release stopped serving the following deprecated API versions:

### NetworkPolicy

The **extensions/v1beta1** API version of NetworkPolicy is no longer served as of v1.16.

- Migrate manifests and API clients to use the **networking.k8s.io/v1** API version, available since v1.8.
- All existing persisted objects are accessible via the new API

### DaemonSet

The **extensions/v1beta1** and **apps/v1beta2** API versions of DaemonSet are no longer served as of v1.16.

- Migrate manifests and API clients to use the **apps/v1** API version, available since v1.9.
- All existing persisted objects are accessible via the new API
- Notable changes:
  - `spec.templateGeneration` is removed
  - `spec.selector` is now required and immutable after creation; use the existing template labels as the selector for seamless upgrades

- `spec.updateStrategy.type` now defaults to `RollingUpdate` (the default in `extensions/v1beta1` was `onDelete` )

## Deployment

The **extensions/v1beta1**, **apps/v1beta1**, and **apps/v1beta2** API versions of Deployment are no longer served as of v1.16.

- Migrate manifests and API clients to use the **apps/v1** API version, available since v1.9.
- All existing persisted objects are accessible via the new API
- Notable changes:
  - `spec.rollbackTo` is removed
  - `spec.selector` is now required and immutable after creation; use the existing template labels as the selector for seamless upgrades
  - `spec.progressDeadlineSeconds` now defaults to 600 seconds (the default in `extensions/v1beta1` was no deadline)
  - `spec.revisionHistoryLimit` now defaults to 10 (the default in `apps/v1beta1` was 2, the default in `extensions/v1beta1` was to retain all)
  - `maxSurge` and `maxUnavailable` now default to 25% (the default in `extensions/v1beta1` was 1)

## StatefulSet

The **apps/v1beta1** and **apps/v1beta2** API versions of StatefulSet are no longer served as of v1.16.

- Migrate manifests and API clients to use the **apps/v1** API version, available since v1.9.
- All existing persisted objects are accessible via the new API
- Notable changes:
  - `spec.selector` is now required and immutable after creation; use the existing template labels as the selector for seamless upgrades
  - `spec.updateStrategy.type` now defaults to `RollingUpdate` (the default in `apps/v1beta1` was `onDelete` )

## ReplicaSet

The **extensions/v1beta1**, **apps/v1beta1**, and **apps/v1beta2** API versions of ReplicaSet are no longer served as of v1.16.

- Migrate manifests and API clients to use the **apps/v1** API version, available since v1.9.
- All existing persisted objects are accessible via the new API
- Notable changes:
  - `spec.selector` is now required and immutable after creation; use the existing template labels as the selector for seamless upgrades

## PodSecurityPolicy

The **extensions/v1beta1** API version of PodSecurityPolicy is no longer served as of v1.16.

- Migrate manifests and API client to use the **policy/v1beta1** API version, available since v1.10.
- Note that the **policy/v1beta1** API version of PodSecurityPolicy will

be removed in v1.25.

## What to do

### Test with deprecated APIs disabled

You can test your clusters by starting an API server with specific API versions disabled to simulate upcoming removals. Add the following flag to the API server startup arguments:

```
--runtime-config=<group>/<version>=false
```

For example:

```
--runtime-config=admissionregistration.k8s.io/v1beta1=false,apiextensions.k8s.io/v1beta1,...
```

### Locate use of deprecated APIs

Use [client warnings, metrics, and audit information available in 1.19+](#) to locate use of deprecated APIs.

### Migrate to non-deprecated APIs

- Update custom integrations and controllers to call the non-deprecated APIs
- Change YAML files to reference the non-deprecated APIs

You can use the `kubectl convert` command to automatically convert an existing object:

```
kubectl convert -f <file> --output-version <group>/<version> .
```

For example, to convert an older Deployment to `apps/v1`, you can run:

```
kubectl convert -f ./my-deployment.yaml --output-version apps/v1
```

This conversion may use non-ideal default values. To learn more about a specific resource, check the Kubernetes [API reference](#).

#### Note:

The `kubectl convert` tool is not installed by default, although in fact it once was part of `kubectl` itself. For more details, you can read the [deprecation and removal issue](#) for the built-in subcommand.

To learn how to set up `kubectl convert` on your computer, visit the page that is right for your operating system: [Linux](#), [macOS](#), or [Windows](#).

## 2.7 - Kubernetes API health endpoints

The Kubernetes API server provides API endpoints to indicate the current status of the API server. This page describes these API endpoints and explains how you can use them.

### API endpoints for health

The Kubernetes API server provides 3 API endpoints (`healthz`, `livez` and `readyz`) to indicate the current status of the API server. The `healthz` endpoint is deprecated (since Kubernetes v1.16), and you should use the more specific `livez` and `readyz` endpoints instead. The `livez` endpoint can be used with the `--livez-grace-period` [flag](#) to specify the startup duration. For a graceful shutdown you can specify the `--shutdown-delay-duration` [flag](#) with the `/readyz` endpoint. Machines that check the `healthz` / `livez` / `readyz` of the API server should rely on the HTTP status code. A status code `200` indicates the API server is `healthy` / `live` / `ready`, depending on the called endpoint. The more verbose options shown below are intended to be used by human operators to debug their cluster or understand the state of the API server.

The following examples will show how you can interact with the health API endpoints.

For all endpoints, you can use the `verbose` parameter to print out the checks and their status. This can be useful for a human operator to debug the current status of the API server, it is not intended to be consumed by a machine:

```
curl -k https://localhost:6443/livez?verbose
```

or from a remote host with authentication:

```
kubectl get --raw='/readyz?verbose'
```

The output will look like this:

```
[+]ping ok
[+]log ok
[+]etcd ok
[+]poststarthook/start-kube-apiserver-admission-initializer ok
[+]poststarthook/generic-apiserver-start-informers ok
[+]poststarthook/start-apisextensions-informers ok
[+]poststarthook/start-apisextensions-controllers ok
[+]poststarthook/crd-informer-synced ok
[+]poststarthook/bootstrap-controller ok
[+]poststarthook/rbac/bootstrap-roles ok
[+]poststarthook/scheduling/bootstrap-system-priority-classes ok
[+]poststarthook/start-cluster-authentication-info-controller ok
[+]poststarthook/start-kube-aggregator-informers ok
[+]poststarthook/apiservice-registration-controller ok
[+]poststarthook/apiservice-status-available-controller ok
[+]poststarthook/kube-apiserver-autoregistration ok
[+]autoregister-completion ok
[+]poststarthook/apiservice-openapi-controller ok
healthz check passed
```

The Kubernetes API server also supports to exclude specific checks. The query parameters can also be combined like in this example:

```
curl -k 'https://localhost:6443/readyz?verbose&exclude=etcd'
```

The output shows that the `etcd` check is excluded:

```
[+]ping ok
[+]log ok
[+]etcd excluded: ok
[+]poststarthook/start-kube-apiserver-admission-initializer ok
[+]poststarthook/generic-apiserver-start-informers ok
[+]poststarthook/start-apisextensions-informers ok
[+]poststarthook/start-apisextensions-controllers ok
[+]poststarthook/crd-informer-synced ok
[+]poststarthook/bootstrap-controller ok
[+]poststarthook/rbac/bootstrap-roles ok
[+]poststarthook/scheduling/bootstrap-system-priority-classes ok
[+]poststarthook/start-cluster-authentication-info-controller ok
[+]poststarthook/start-kube-aggregator-informers ok
[+]poststarthook/apiservice-registration-controller ok
[+]poststarthook/apiservice-status-available-controller ok
[+]poststarthook/kube-apiserver-autoregistration ok
[+]autoregister-completion ok
[+]poststarthook/apiservice-openapi-controller ok
[+]shutdown ok
healthz check passed
```

## Individual health checks

ⓘ **FEATURE STATE:** Kubernetes v1.31 [alpha]

Each individual health check exposes an HTTP endpoint and can be checked individually. The schema for the individual health checks is `/livez/<healthcheck-name>` or `/readyz/<healthcheck-name>`, where `livez` and `readyz` can be used to indicate if you want to check the liveness or the readiness of the API server, respectively. The `<healthcheck-name>` path can be discovered using the `verbose` flag from above and take the path between `[+]` and `ok`. These individual health checks should not be consumed by machines but can be helpful for a human operator to debug a system:

```
curl -k https://localhost:6443/livez/etcd
```

# 3 - API Access Control

For an introduction to how Kubernetes implements and controls API access, read [Controlling Access to the Kubernetes API](#).

Reference documentation:

- [Authenticating](#)
  - [Authenticating with Bootstrap Tokens](#)
- [Admission Controllers](#)
  - [Dynamic Admission Control](#)
- [Authorization](#)
  - [Role Based Access Control](#)
  - [Attribute Based Access Control](#)
  - [Node Authorization](#)
  - [Webhook Authorization](#)
- [Certificate Signing Requests](#)
  - including [CSR approval](#) and [certificate signing](#)
- Service accounts
  - [Developer guide](#)
  - [Administration](#)
- [Kubelet Authentication & Authorization](#)
  - including kubelet [TLS bootstrapping](#)

## 3.1 - Authenticating

This page provides an overview of authentication.

### Users in Kubernetes

All Kubernetes clusters have two categories of users: service accounts managed by Kubernetes, and normal users.

It is assumed that a cluster-independent service manages normal users in the following ways:

- an administrator distributing private keys
- a user store like Keystone or Google Accounts
- a file with a list of usernames and passwords

In this regard, *Kubernetes does not have objects which represent normal user accounts*. Normal users cannot be added to a cluster through an API call.

Even though a normal user cannot be added via an API call, any user that presents a valid certificate signed by the cluster's certificate authority (CA) is considered authenticated. In this configuration, Kubernetes determines the username from the common name field in the 'subject' of the cert (e.g., "/CN=bob"). From there, the role based access control (RBAC) sub-system would determine whether the user is authorized to perform a specific operation on a resource. For more details, refer to the normal users topic in [certificate request](#) for more details about this.

In contrast, service accounts are users managed by the Kubernetes API. They are bound to specific namespaces, and created automatically by the API server or manually through API calls. Service accounts are tied to a set of credentials stored as `Secrets`, which are mounted into pods allowing in-cluster processes to talk to the Kubernetes API.

API requests are tied to either a normal user or a service account, or are treated as [anonymous requests](#). This means every process inside or outside the cluster, from a human user typing `kubectl` on a workstation, to `kubelets` on nodes, to members of the control plane, must authenticate when making requests to the API server, or be treated as an anonymous user.

### Authentication strategies

Kubernetes uses client certificates, bearer tokens, or an authenticating proxy to authenticate API requests through authentication plugins. As HTTP requests are made to the API server, plugins attempt to associate the following attributes with the request:

- Username: a string which identifies the end user. Common values might be `kube-admin` or `jane@example.com`.
- UID: a string which identifies the end user and attempts to be more consistent and unique than username.
- Groups: a set of strings, each of which indicates the user's membership in a named logical collection of users. Common values might be `system:masters` or `devops-team`.
- Extra fields: a map of strings to list of strings which holds additional information authorizers may find useful.

All values are opaque to the authentication system and only hold

significance when interpreted by an [authorizer](#).

You can enable multiple authentication methods at once. You should usually use at least two methods:

- service account tokens for service accounts
- at least one other method for user authentication.

When multiple authenticator modules are enabled, the first module to successfully authenticate the request short-circuits evaluation. The API server does not guarantee the order authenticators run in.

The `system:authenticated` group is included in the list of groups for all authenticated users.

Integrations with other authentication protocols (LDAP, SAML, Kerberos, alternate x509 schemes, etc) can be accomplished using an [authenticating proxy](#) or the [authentication webhook](#).

## X509 client certificates

Client certificate authentication is enabled by passing the `--client-ca-file=SOMEFILE` option to API server. The referenced file must contain one or more certificate authorities to use to validate client certificates presented to the API server. If a client certificate is presented and verified, the common name of the subject is used as the user name for the request. As of Kubernetes 1.4, client certificates can also indicate a user's group memberships using the certificate's organization fields. To include multiple group memberships for a user, include multiple organization fields in the certificate.

For example, using the `openssl` command line tool to generate a certificate signing request:

```
openssl req -new -key jbeda.pem -out jbeda-csr.pem -subj "/CN=jbeda/C
```

This would create a CSR for the username "jbeda", belonging to two groups, "app1" and "app2".

See [Managing Certificates](#) for how to generate a client cert.

## Static token file

The API server reads bearer tokens from a file when given the `--token-auth-file=SOMEFILE` option on the command line. Currently, tokens last indefinitely, and the token list cannot be changed without restarting the API server.

The token file is a csv file with a minimum of 3 columns: token, user name, user uid, followed by optional group names.

### Note:

If you have more than one group, the column must be double quoted e.g.

```
token,user,uid,"group1,group2,group3"
```

## Putting a bearer token in a request

When using bearer token authentication from an http client, the API server expects an `Authorization` header with a value of `Bearer <token>`. The bearer token must be a character sequence that can be put in an HTTP header value using no more than the encoding and quoting facilities of HTTP. For example: if the bearer token is `31ada4fd-adec-460c-809a-9e56ceb75269` then it would appear in an HTTP header as shown below.

```
Authorization: Bearer 31ada4fd-adec-460c-809a-9e56ceb75269
```

## Bootstrap tokens

### ⓘ FEATURE STATE: Kubernetes v1.18 [stable]

To allow for streamlined bootstrapping for new clusters, Kubernetes includes a dynamically-managed Bearer token type called a *Bootstrap Token*. These tokens are stored as Secrets in the `kube-system` namespace, where they can be dynamically managed and created. Controller Manager contains a TokenCleaner controller that deletes bootstrap tokens as they expire.

The tokens are of the form `[a-z0-9]{6}.[a-z0-9]{16}`. The first component is a Token ID and the second component is the Token Secret. You specify the token in an HTTP header as follows:

```
Authorization: Bearer 781292.db7bc3a58fc5f07e
```

You must enable the Bootstrap Token Authenticator with the `--enable-bootstrap-token-auth` flag on the API Server. You must enable the TokenCleaner controller via the `--controllers` flag on the Controller Manager. This is done with something like `--controllers=*,tokencleaner`. `kubeadm` will do this for you if you are using it to bootstrap a cluster.

The authenticator authenticates as `system:bootstrap:<Token ID>`. It is included in the `system:bootstrappers` group. The naming and groups are intentionally limited to discourage users from using these tokens past bootstrapping. The user names and group can be used (and are used by `kubeadm`) to craft the appropriate authorization policies to support bootstrapping a cluster.

Please see [Bootstrap Tokens](#) for in depth documentation on the Bootstrap Token authenticator and controllers along with how to manage these tokens with `kubeadm`.

## Service account tokens

A service account is an automatically enabled authenticator that uses signed bearer tokens to verify requests. The plugin takes two optional flags:

- `--service-account-key-file` File containing PEM-encoded x509 RSA or ECDSA private or public keys, used to verify ServiceAccount tokens. The specified file can contain multiple keys, and the flag can be specified multiple times with different files. If unspecified, `--tls-private-key-file` is used.
- `--service-account-lookup` If enabled, tokens which are deleted from the API will be revoked.

Service accounts are usually created automatically by the API server and 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. Accounts may be explicitly associated with pods using the `serviceAccountName` field of a `PodSpec`.

**Note:**

`serviceAccountName` is usually omitted because this is done automatically.

```
apiVersion: apps/v1 # this apiVersion is relevant as of Kubernetes 1.
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
spec:
  replicas: 3
  template:
    metadata:
      # ...
    spec:
      serviceAccountName: bob-the-bot
      containers:
        - name: nginx
          image: nginx:1.14.2
```

Service account bearer tokens are perfectly valid to use outside the cluster and can be used to create identities for long standing jobs that wish to talk to the Kubernetes API. To manually create a service account, use the `kubectl create serviceaccount (NAME)` command. This creates a service account in the current namespace.

```
kubectl create serviceaccount jenkins
```

```
serviceaccount/jenkins created
```

Create an associated token:

```
kubectl create token jenkins
```

```
eyJhbGciOiJSUzI1NiIsImtp...
```

The created token is a signed JSON Web Token (JWT).

The signed JWT can be used as a bearer token to authenticate as the given service account. See [above](#) for how the token is included in a request. Normally these tokens are mounted into pods for in-cluster access to the API server, but can be used from outside the cluster as well.

Service accounts authenticate with the username `system:serviceaccount:(NAMESPACE):(SERVICEACCOUNT)`, and are assigned to the groups `system:serviceaccounts` and `system:serviceaccounts:(NAMESPACE)`.

## Warning:

Because service account tokens can also be stored in Secret API objects, any user with write access to Secrets can request a token, and any user with read access to those Secrets can authenticate as the service account. Be cautious when granting permissions to service accounts and read or write capabilities for Secrets.

## OpenID Connect Tokens

[OpenID Connect](#) is a flavor of OAuth2 supported by some OAuth2 providers, notably Microsoft Entra ID, Salesforce, and Google. The protocol's main extension of OAuth2 is an additional field returned with the access token called an [ID Token](#). This token is a JSON Web Token (JWT) with well known fields, such as a user's email, signed by the server.

To identify the user, the authenticator uses the `id_token` (not the `access_token`) from the OAuth2 [token response](#) as a bearer token. See [above](#) for how the token is included in a request.

```
sequenceDiagram participant user as User participant idp as Identity Provider participant kube as kubectl participant api as API Server user ->> idp: 1. Log in to IdP activate idp idp -->> user: 2. Provide access_token, id_token, and refresh_token deactivate idp activate user user ->> kube: 3. Call kubectl with --token being the id_token OR add tokens to .kube/config deactivate user activate kube kube ->> api: 4. Authorization: Bearer... deactivate kube activate api api ->> api: 5. Is JWT signature valid? api ->> api: 6. Has the JWT expired? (iat+exp) api ->> api: 7. User authorized? api -->> kube: 8. Authorized: Perform action and return result deactivate api activate kube kube --x user: 9. Return result deactivate kube
```

1. Log in to your identity provider
2. Your identity provider will provide you with an `access_token` , `id_token` and a `refresh_token`
3. When using `kubectl` , use your `id_token` with the `--token` flag or add it directly to your `kubeconfig`
4. `kubectl` sends your `id_token` in a header called `Authorization` to the API server
5. The API server will make sure the JWT signature is valid
6. Check to make sure the `id_token` hasn't expired

Perform claim and/or user validation if CEL expressions are configured with `AuthenticationConfiguration` .

7. Make sure the user is authorized
8. Once authorized the API server returns a response to `kubectl`
9. `kubectl` provides feedback to the user

Since all of the data needed to validate who you are is in the `id_token` , Kubernetes doesn't need to "phone home" to the identity provider. In a model where every request is stateless this provides a very scalable solution for authentication. It does offer a few challenges:

1. Kubernetes has no "web interface" to trigger the authentication process. There is no browser or interface to collect credentials

which is why you need to authenticate to your identity provider first.

2. The `id_token` can't be revoked, it's like a certificate so it should be short-lived (only a few minutes) so it can be very annoying to have to get a new token every few minutes.
3. To authenticate to the Kubernetes dashboard, you must use the `kubectl proxy` command or a reverse proxy that injects the `id_token`.

## Configuring the API Server

### Using flags

To enable the plugin, configure the following flags on the API server:

Parameter	Description	Example	Required
<code>--oidc-issuer-url</code>	URL of the provider that allows the API server to discover public signing keys. Only URLs that use the <code>https://</code> scheme are accepted. This is typically the provider's discovery URL, changed to have an empty path.	If the issuer's OIDC discovery URL is <code>https://accounts.provider.example/.well-known/openid-configuration</code> , the value should be <code>https://accounts.provider.example</code>	Yes
<code>--oidc-client-id</code>	A client id that all tokens must be issued for.	<code>kubernetes</code>	Yes
<code>--oidc-username-claim</code>	JWT claim to use as the user name. By default <code>sub</code> , which is expected to be a unique identifier of the end user. Admins can choose other claims, such as <code>email</code> or <code>name</code> , depending on their provider. However, claims other than <code>email</code> will be prefixed with the issuer URL to prevent naming clashes with other plugins.	<code>sub</code>	No

Parameter	Description	Example	Required
--oidc-username-prefix	Prefix prepended to username claims to prevent clashes with existing names (such as <code>system:users</code> ). For example, the value <code>oidc:</code> will create usernames like <code>oidc:jane.doe</code> . If this flag isn't provided and <code>--oidc-username-claim</code> is a value other than <code>email</code> the prefix defaults to <code>(Issuer URL) # where (Issuer URL)</code> is the value of <code>--oidc-issuer-url</code> . The value <code>-</code> can be used to disable all prefixing.	<code>oidc:</code>	No
--oidc-groups-claim	JWT claim to use as the user's group. If the claim is present it must be an array of strings.	<code>groups</code>	No
--oidc-groups-prefix	Prefix prepended to group claims to prevent clashes with existing names (such as <code>system:groups</code> ). For example, the value <code>oidc:</code> will create group names like <code>oidc:engineering</code> and <code>oidc:infra</code> .	<code>oidc:</code>	No
--oidc-required-claim	A key=value pair that describes a required claim in the ID Token. If set, the claim is verified to be present in the ID Token with a matching value. Repeat this flag to specify multiple claims.	<code>claim=value</code>	No

Parameter	Description	Example	Required
--oidc-ca-file	The path to the certificate for the CA that signed your identity provider's web certificate. Defaults to the host's root CAs.	/etc/kubernetes/ssl/kc-ca.pem	No
--oidc-signing-algs	The signing algorithms accepted. Default is "RS256".	RS512	No

Authentication configuration from a file

ⓘ **FEATURE STATE:** Kubernetes v1.30 [beta]

JWT Authenticator is an authenticator to authenticate Kubernetes users using JWT compliant tokens. The authenticator will attempt to parse a raw ID token, verify it's been signed by the configured issuer. The public key to verify the signature is discovered from the issuer's public endpoint using OIDC discovery.

The minimum valid JWT payload must contain the following claims:

```
{
  "iss": "https://example.com",           // must match the issuer.url
  "aud": ["my-app"],                     // at least one of the entries in i
  "exp": 1234567890,                   // token expiration as Unix time (t
  "<username-claim>": "user"          // this is the username claim config
}
```

The configuration file approach allows you to configure multiple JWT authenticators, each with a unique `issuer.url` and `issuer.discoveryURL`. The configuration file even allows you to specify [CEL](#) expressions to map claims to user attributes, and to validate claims and user information. The API server also automatically reloads the authenticators when the configuration file is modified. You can use `apiserver_authentication_config_controller_automatic_reload_last_timestamp` metric to monitor the last time the configuration was reloaded by the API server.

You must specify the path to the authentication configuration using the `--authentication-config` flag on the API server. If you want to use command line flags instead of the configuration file, those will continue to work as-is. To access the new capabilities like configuring multiple authenticators, setting multiple audiences for an issuer, switch to using the configuration file.

For Kubernetes v1.31, the structured authentication configuration file format is beta-level, and the mechanism for using that configuration is also beta. Provided you didn't specifically disable the `StructuredAuthenticationConfiguration` [feature gate](#) for your cluster, you can turn on structured authentication by specifying the `--authentication-config` command line argument to the `kube-apiserver`. An example of the structured authentication configuration file is shown below.

**Note:**

If you specify `--authentication-config` along with any of the `--oidc-*` command line arguments, this is a misconfiguration. In this situation, the API server reports an error and then immediately exits. If you want to switch to using structured authentication configuration, you have to remove the `--oidc-*` command line arguments, and use the configuration file instead.

```
---  
#  
# CAUTION: this is an example configuration.  
#           Do not use this for your own cluster!  
#  
apiVersion: apiserver.config.k8s.io/v1beta1  
kind: AuthenticationConfiguration  
# List of authenticators to authenticate Kubernetes users using JWT c  
# the maximum number of allowed authenticators is 64.  
jwt:  
- issuer:  
  # url must be unique across all authenticators.  
  # url must not conflict with issuer configured in --service-accou  
  url: https://example.com # Same as --oidc-issuer-url.  
  # discoveryURL, if specified, overrides the URL used to fetch dis  
  # information instead of using "{url}/.well-known/openid-configur  
  # The exact value specified is used, so "./.well-known/openid-conf  
  # must be included in discoveryURL if needed.  
  #  
  # The "issuer" field in the fetched discovery information must ma  
  # in the AuthenticationConfiguration and will be used to validate  
  # This is for scenarios where the well-known and jwks endpoints a  
  # location than the issuer (such as locally in the cluster).  
  # discoveryURL must be different from url if specified and must b  
  discoveryURL: https://discovery.example.com/.well-known/openid-co  
  # PEM encoded CA certificates used to validate the connection whe  
  # discovery information. If not set, the system verifier will be  
  # Same value as the content of the file referenced by the --oidc-  
  certificateAuthority: <PEM encoded CA certificates>  
  # audiences is the set of acceptable audiences the JWT must be is  
  # At least one of the entries must match the "aud" claim in prese  
  audiences:  
  - my-app # Same as --oidc-client-id.  
  - my-other-app  
  # this is required to be set to "MatchAny" when multiple audience  
  audienceMatchPolicy: MatchAny  
  # rules applied to validate token claims to authenticate users.  
  claimValidationRules:  
    # Same as --oidc-required-claim key=value.  
    - claim: hd  
      requiredValue: example.com  
      # Instead of claim and requiredValue, you can use expression to v  
      # expression is a CEL expression that evaluates to a boolean.  
      # all the expressions must evaluate to true for validation to suc  
      - expression: 'claims.hd == "example.com"'  
        # Message customizes the error message seen in the API server Log  
        message: the hd claim must be set to example.com  
      - expression: 'claims.exp - claims.nbf <= 86400'  
        message: total token lifetime must not exceed 24 hours  
  claimMappings:  
    # username represents an option for the username attribute.  
    # This is the only required attribute.  
    username:  
      # Same as --oidc-username-claim. Mutually exclusive with usern  
      claim: "sub"  
      # Same as --oidc-username-prefix. Mutually exclusive with usern  
      # if username.claim is set, username.prefix is required.  
      # Explicitly set it to "" if no prefix is desired.  
      prefix: ""  
      # Mutually exclusive with username.claim and username.prefix.  
      # expression is a CEL expression that evaluates to a string.  
      #  
      # 1. If username.expression uses 'claims.email', then 'claims.  
      #     username.expression or extra[*].valueExpression or claimv  
      #     An example claim validation rule expression that matches
```

```

#      applied when username.claim is set to 'email' is 'claims'.
# 2. If the username asserted based on username.expression is
#      request will fail.
expression: 'claims.username + ":external-user"
# groups represents an option for the groups attribute.
groups:
# Same as --oidc-groups-claim. Mutually exclusive with groups.e
claim: "sub"
# Same as --oidc-groups-prefix. Mutually exclusive with groups.
# if groups.claim is set, groups.prefix is required.
# Explicitly set it to "" if no prefix is desired.
prefix: ""
# Mutually exclusive with groups.claim and groups.prefix.
# expression is a CEL expression that evaluates to a string or
expression: 'claims.roles.split(",")'
# uid represents an option for the uid attribute.
uid:
# Mutually exclusive with uid.expression.
claim: 'sub'
# Mutually exclusive with uid.claim
# expression is a CEL expression that evaluates to a string.
expression: 'claims.sub'
# extra attributes to be added to the UserInfo object. Keys must
extra:
- key: 'example.com/tenant'
# valueExpression is a CEL expression that evaluates to a string
valueExpression: 'claims.tenant'
# validation rules applied to the final user object.
userValidationRules:
# expression is a CEL expression that evaluates to a boolean.
# all the expressions must evaluate to true for the user to be va
- expression: "!user.username.startsWith('system:')"
# Message customizes the error message seen in the API server log
message: 'username cannot used reserved system: prefix'
- expression: "user.groups.all(group, !group.startsWith('system:'))"
message: 'groups cannot used reserved system: prefix'

```

- Claim validation rule expression

`jwt.claimValidationRules[i].expression` represents the expression which will be evaluated by CEL. CEL expressions have access to the contents of the token payload, organized into `claims` CEL variable. `claims` is a map of claim names (as strings) to claim values (of any type).

- User validation rule expression

`jwt.userValidationRules[i].expression` represents the expression which will be evaluated by CEL. CEL expressions have access to the contents of `userInfo`, organized into `user` CEL variable. Refer to the [UserInfo](#) API documentation for the schema of `user`.

- Claim mapping expression

`jwt.claimMappings.username.expression`,  
`jwt.claimMappings.groups.expression`,  
`jwt.claimMappings.uid.expression`  
`jwt.claimMappings.extra[i].valueExpression` represents the expression which will be evaluated by CEL. CEL expressions have access to the contents of the token payload, organized into `claims` CEL variable. `claims` is a map of claim names (as strings) to claim values (of any type).

To learn more, see the [Documentation on CEL](#)

Here are examples of the `AuthenticationConfiguration` with different token payloads.

Valid token	<a href="#">Fails claim validation</a>	<a href="#">Fails user validation</a>
<pre>apiVersion: apiserver.config.k8s.io/v1beta1 kind: AuthenticationConfiguration jwt: - issuer:   url: https://example.com   audiences:   - my-app   claimMappings:     username:       expression: 'claims.username + ":external-user"'     groups:       expression: 'claims.roles.split(",")'     uid:       expression: 'claims.sub'     extra:     - key: 'example.com/tenant'       valueExpression: 'claims.tenant'   userValidationRules:   - expression: "user.username.startsWith('system:')" # this     message: 'username cannot used reserved system: prefix'</pre>		

TOKEN=eyJhbGciOiJSUzI1NiIsImtpZCI6ImY3dF9tOEROWmFTQk1oWGw5Q;

where the token payload is:

```
{
  "aud": "kubernetes",
  "exp": 1703232949,
  "iat": 1701107233,
  "iss": "https://example.com",
  "jti": "7c337942807e73caa2c30c868ac0ce910bce02ddcbfebe8",
  "nbf": 1701107233,
  "roles": "user,admin",
  "sub": "auth",
  "tenant": "72f988bf-86f1-41af-91ab-2d7cd011db4a",
  "username": "foo"
}
```

The token with the above `AuthenticationConfiguration` will produce the following `UserInfo` object and successfully authenticate the user.

```
{
  "username": "foo:external-user",
  "uid": "auth",
  "groups": [
    "user",
    "admin"
  ],
  "extra": {
    "example.com/tenant": "72f988bf-86f1-41af-91ab-2d7cd011db4a"
  }
}
```

## Limitations

1. Distributed claims do not work via [CEL](#) expressions.
2. Egress selector configuration is not supported for calls to

```
issuer.url and issuer.discoveryURL .
```

Kubernetes does not provide an OpenID Connect Identity Provider. You can use an existing public OpenID Connect Identity Provider (such as Google, or [others](#)). Or, you can run your own Identity Provider, such as [dex](#), [Keycloak](#), CloudFoundry [UAA](#), or Tremolo Security's [OpenUnison](#).

For an identity provider to work with Kubernetes it must:

### 1. Support [OpenID connect discovery](#)

The public key to verify the signature is discovered from the issuer's public endpoint using OIDC discovery. If you're using the authentication configuration file, the identity provider doesn't need to publicly expose the discovery endpoint. You can host the discovery endpoint at a different location than the issuer (such as locally in the cluster) and specify the `issuer.discoveryURL` in the configuration file.

### 2. Run in TLS with non-obsolete ciphers

### 3. Have a CA signed certificate (even if the CA is not a commercial CA or is self signed)

A note about requirement #3 above, requiring a CA signed certificate. If you deploy your own identity provider (as opposed to one of the cloud providers like Google or Microsoft) you MUST have your identity provider's web server certificate signed by a certificate with the `ca` flag set to `TRUE`, even if it is self signed. This is due to GoLang's TLS client implementation being very strict to the standards around certificate validation. If you don't have a CA handy, you can use the [gencert script](#) from the Dex team to create a simple CA and a signed certificate and key pair. Or you can use [this similar script](#) that generates SHA256 certs with a longer life and larger key size.

Refer to setup instructions for specific systems:

- [UAA](#)
- [Dex](#)
- [OpenUnison](#)

## Using kubectl

### Option 1 - OIDC Authenticator

The first option is to use the `kubectl oidc` authenticator, which sets the `id_token` as a bearer token for all requests and refreshes the token once it expires. After you've logged into your provider, use `kubectl` to add your `id_token`, `refresh_token`, `client_id`, and `client_secret` to configure the plugin.

Providers that don't return an `id_token` as part of their refresh token response aren't supported by this plugin and should use "Option 2" below.

```
kubectl config set-credentials USER_NAME \
  --auth-provider=oidc \
  --auth-provider-arg=idp-issuer-url=( issuer url ) \
  --auth-provider-arg=client-id=( your client id ) \
  --auth-provider-arg=client-secret=( your client secret ) \
  --auth-provider-arg=refresh-token=( your refresh token ) \
  --auth-provider-arg=idp-certificate-authority=( path to your ca cert ) \
  --auth-provider-arg=id-token=( your id_token )
```

As an example, running the below command after authenticating to your identity provider:

```
kubectl config set-credentials mmosley \
  --auth-provider=oidc \
  --auth-provider-arg=idp-issuer-url=https://oidcidp.tremolo.la
  --auth-provider-arg=client-id=kubernetes \
  --auth-provider-arg=client-secret=1db158f6-177d-4d9c-8a8b-d36
  --auth-provider-arg=refresh-token=q1bKLF0yUiosTfawzA93TzZIDzH
  --auth-provider-arg=idp-certificate-authority=/root/ca.pem \
  --auth-provider-arg=id-token=eyJraWQiOijDTj1vaWRjaWRwLnRyZW1v
```

Which would produce the below configuration:

```
users:
- name: mmosley
  user:
    auth-provider:
      config:
        client-id: kubernetes
        client-secret: 1db158f6-177d-4d9c-8a8b-d36869918ec5
        id-token: eyJraWQiOijDTj1vaWRjaWRwLnRyZW1vbG8ubGFuLCBPVT1EZW1
        idp-certificate-authority: /root/ca.pem
        idp-issuer-url: https://oidcidp.tremolo.lan:8443/auth/idp/Oidc
        refresh-token: q1bKLF0yUiosTfawzA93TzZIDzH2TNa2SMm0zEiPKTUwME
      name: oidc
```

Once your `id_token` expires, `kubectl` will attempt to refresh your `id_token` using your `refresh_token` and `client_secret` storing the new values for the `refresh_token` and `id_token` in your `.kube/config`.

### Option 2 - Use the `--token` Option

The `kubectl` command lets you pass in a token using the `--token` option. Copy and paste the `id_token` into this option:

```
kubectl --token=eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL21sYi50cmVt
```

## Webhook Token Authentication

Webhook authentication is a hook for verifying bearer tokens.

- `--authentication-token-webhook-config-file` a configuration file describing how to access the remote webhook service.
- `--authentication-token-webhook-cache-ttl` how long to cache authentication decisions. Defaults to two minutes.
- `--authentication-token-webhook-version` determines whether to use `authentication.k8s.io/v1beta1` OR `authentication.k8s.io/v1` `TokenReview` objects to send/receive information from the webhook. Defaults to `v1beta1`.

The configuration file uses the [kubeconfig](#) file format. Within the file, `clusters` refers to the remote service and `users` refers to the API server webhook. An example would be:

```
# Kubernetes API version
apiVersion: v1
# kind of the API object
kind: Config
# clusters refers to the remote service.
clusters:
  - name: name-of-remote-authn-service
    cluster:
      certificate-authority: /path/to/ca.pem          # CA for verifyi
      server: https://authn.example.com/authenticate # URL of remote

# users refers to the API server's webhook configuration.
users:
  - name: name-of-api-server
    user:
      client-certificate: /path/to/cert.pem # cert for the webhook pl
      client-key: /path/to/key.pem          # key matching the cert

# kubeconfig files require a context. Provide one for the API server.
current-context: webhook
contexts:
  - context:
      cluster: name-of-remote-authn-service
      user: name-of-api-server
      name: webhook
```

When a client attempts to authenticate with the API server using a bearer token as discussed [above](#), the authentication webhook POSTs a JSON-serialized `TokenReview` object containing the token to the remote service.

Note that webhook API objects are subject to the same [versioning compatibility rules](#) as other Kubernetes API objects. Implementers should check the `apiVersion` field of the request to ensure correct deserialization, and **must** respond with a `TokenReview` object of the same version as the request.

[authentication.k8s.io/v1](#)

[authentication.k8s.io/v1beta1](#)

**Note:**

The Kubernetes API server defaults to sending `authentication.k8s.io/v1beta1` token reviews for backwards compatibility. To opt into receiving `authentication.k8s.io/v1` token reviews, the API server must be started with `--authentication-token-webhook-version=v1`.

```
{  
  "apiVersion": "authentication.k8s.io/v1",  
  "kind": "TokenReview",  
  "spec": {  
    # Opaque bearer token sent to the API server  
    "token": "014fbff9a07c...",  
  
    # Optional list of the audience identifiers for the server the token was presented to.  
    # Audience-aware token authenticators (for example, OIDC token authenticators)  
    # should verify the token was intended for at least one of the audiences in this list  
    # and return the intersection of this list and the valid audiences for the token.  
    # This ensures the token is valid to authenticate to the server it was presented to.  
    # If no audiences are provided, the token should be validated to authenticate to any server.  
    "audiences": ["https://myserver.example.com", "https://myserver.internal.example.com"]  
  }  
}
```

The remote service is expected to fill the `status` field of the request to indicate the success of the login. The response body's `spec` field is ignored and may be omitted. The remote service must return a response using the same `TokenReview` API version that it received. A successful validation of the bearer token would return:

[authentication.k8s.io/v1](#)[authentication.k8s.io/v1beta1](#)

```
{  
  "apiVersion": "authentication.k8s.io/v1",  
  "kind": "TokenReview",  
  "status": {  
    "authenticated": true,  
    "user": {  
      # Required  
      "username": "janedoe@example.com",  
      # Optional  
      "uid": "42",  
      # Optional group memberships  
      "groups": ["developers", "qa"],  
      # Optional additional information provided by the authenticator.  
      # This should not contain confidential data, as it can be recorded in logs  
      # or API objects, and is made available to admission webhooks.  
      "extra": {  
        "extrafield1": [  
          "extravalue1",  
          "extravalue2"  
        ]  
      }  
    },  
    # Optional list audience-aware token authenticators can return,  
    # containing the audiences from the `spec.audiences` list for which the provided  
    # token is valid.  
    # If this is omitted, the token is considered to be valid to authenticate to the server.  
    "audiences": ["https://myserver.example.com"]  
  }  
}
```

An unsuccessful request would return:

[authentication.k8s.io/v1](https://kubernetes.io/docs/reference/config/api/v1/authentication.k8s.io/v1)    [authentication.k8s.io/v1beta1](https://kubernetes.io/docs/reference/config/api/v1beta1/authentication.k8s.io/v1beta1)

```
{  
  "apiVersion": "authentication.k8s.io/v1",  
  "kind": "TokenReview",  
  "status": {  
    "authenticated": false,  
    # Optionally include details about why authentication failed.  
    # If no error is provided, the API will return a generic Unauthorized message.  
    # The error field is ignored when authenticated=true.  
    "error": "Credentials are expired"  
  }  
}
```

## Authenticating Proxy

The API server can be configured to identify users from request header values, such as `X-Remote-User`. It is designed for use in combination with an authenticating proxy, which sets the request header value.

- `--requestheader-username-headers` Required, case-insensitive. Header names to check, in order, for the user identity. The first header containing a value is used as the username.
- `--requestheader-group-headers` 1.6+. Optional, case-insensitive. "X-Remote-Group" is suggested. Header names to check, in order, for the user's groups. All values in all specified headers are used as group names.
- `--requestheader-extra-headers-prefix` 1.6+. Optional, case-insensitive. "X-Remote-Extra-" is suggested. Header prefixes to look for to determine extra information about the user (typically used by the configured authorization plugin). Any headers beginning with any of the specified prefixes have the prefix removed. The remainder of the header name is lowercased and [percent-decoded](#) and becomes the extra key, and the header value is the extra value.

### Note:

Prior to 1.11.3 (and 1.10.7, 1.9.11), the extra key could only contain characters which were [legal in HTTP header labels](#).

For example, with this configuration:

```
--requestheader-username-headers=X-Remote-User  
--requestheader-group-headers=X-Remote-Group  
--requestheader-extra-headers-prefix=X-Remote-Extra-
```

this request:

```
GET / HTTP/1.1  
X-Remote-User: fido  
X-Remote-Group: dogs  
X-Remote-Group: dachshunds  
X-Remote-Extra-Acme.com%2Fproject: some-project  
X-Remote-Extra-Scopes: openid  
X-Remote-Extra-Scopes: profile
```

would result in this user info:

```
name: fido
groups:
- dogs
- dachshunds
extra:
  acme.com/project:
  - some-project
  scopes:
  - openid
  - profile
```

In order to prevent header spoofing, the authenticating proxy is required to present a valid client certificate to the API server for validation against the specified CA before the request headers are checked. **WARNING:** do **not** reuse a CA that is used in a different context unless you understand the risks and the mechanisms to protect the CA's usage.

- `--requestheader-client-ca-file` Required. PEM-encoded certificate bundle. A valid client certificate must be presented and validated against the certificate authorities in the specified file before the request headers are checked for user names.
- `--requestheader-allowed-names` Optional. List of Common Name values (CNs). If set, a valid client certificate with a CN in the specified list must be presented before the request headers are checked for user names. If empty, any CN is allowed.

## Anonymous requests

When enabled, requests that are not rejected by other configured authentication methods are treated as anonymous requests, and given a username of `system:anonymous` and a group of `system:unauthenticated`.

For example, on a server with token authentication configured, and anonymous access enabled, a request providing an invalid bearer token would receive a `401 Unauthorized` error. A request providing no bearer token would be treated as an anonymous request.

In 1.5.1-1.5.x, anonymous access is disabled by default, and can be enabled by passing the `--anonymous-auth=true` option to the API server.

In 1.6+, anonymous access is enabled by default if an authorization mode other than `AlwaysAllow` is used, and can be disabled by passing the `--anonymous-auth=false` option to the API server. Starting in 1.6, the ABAC and RBAC authorizers require explicit authorization of the `system:anonymous` user or the `system:unauthenticated` group, so legacy policy rules that grant access to the `*` user or `*` group do not include anonymous users.

## Anonymous Authenticator Configuration

### ⓘ FEATURE STATE: Kubernetes v1.31 [alpha]

The `AuthenticationConfiguration` can be used to configure the anonymous authenticator. To enable configuring anonymous auth via the config file you need enable the `AnonymousAuthConfigurableEndpoints` feature gate. When this feature gate is enabled you cannot set the `--`

anonymous-auth flag.

The main advantage of configuring anonymous authenticator using the authentication configuration file is that in addition to enabling and disabling anonymous authentication you can also configure which endpoints support anonymous authentication.

A sample authentication configuration file is below:

```
---  
#  
# CAUTION: this is an example configuration.  
#           Do not use this for your own cluster!  
#  
apiVersion: apiserver.config.k8s.io/v1beta1  
kind: AuthenticationConfiguration  
anonymous:  
  enabled: true  
  conditions:  
    - path: /livez  
    - path: /readyz  
    - path: /healthz
```

In the configuration above only the `/livez`, `/readyz` and `/healthz` endpoints are reachable by anonymous requests. Any other endpoints will not be reachable even if it is allowed by RBAC configuration.

## User impersonation

A user can act as another user through impersonation headers. These let requests manually override the user info a request authenticates as. For example, an admin could use this feature to debug an authorization policy by temporarily impersonating another user and seeing if a request was denied.

Impersonation requests first authenticate as the requesting user, then switch to the impersonated user info.

- A user makes an API call with their credentials *and* impersonation headers.
- API server authenticates the user.
- API server ensures the authenticated users have impersonation privileges.
- Request user info is replaced with impersonation values.
- Request is evaluated, authorization acts on impersonated user info.

The following HTTP headers can be used to performing an impersonation request:

- `Impersonate-User` : The username to act as.
- `Impersonate-Group` : A group name to act as. Can be provided multiple times to set multiple groups. Optional. Requires "`Impersonate-User`".
- `Impersonate-Extra-( extra name )` : A dynamic header used to associate extra fields with the user. Optional. Requires "`Impersonate-User`". In order to be preserved consistently, `( extra name )` must be lower-case, and any characters which aren't [legal in HTTP header labels](#) MUST be utf8 and [percent-encoded](#).
- `Impersonate-Uid` : A unique identifier that represents the user being impersonated. Optional. Requires "`Impersonate-User`". Kubernetes

does not impose any format requirements on this string.

**Note:**

Prior to 1.11.3 (and 1.10.7, 1.9.11), ( `extra name` ) could only contain characters which were [legal in HTTP header labels](#).

**Note:**

`Impersonate-Uid` is only available in versions 1.22.0 and higher.

An example of the impersonation headers used when impersonating a user with groups:

```
Impersonate-User: jane.doe@example.com
Impersonate-Group: developers
Impersonate-Group: admins
```

An example of the impersonation headers used when impersonating a user with a UID and extra fields:

```
Impersonate-User: jane.doe@example.com
Impersonate-Extra-dn: cn=jane,ou=engineers,dc=example,dc=com
Impersonate-Extra-acme.com%2Fproject: some-project
Impersonate-Extra-scopes: view
Impersonate-Extra-scopes: development
Impersonate-Uid: 06f6ce97-e2c5-4ab8-7ba5-7654dd08d52b
```

When using `kubectl` set the `--as` flag to configure the `Impersonate-User` header, set the `--as-group` flag to configure the `Impersonate-Group` header.

```
kubectl drain mynode
```

```
Error from server (Forbidden): User "clark" cannot get nodes at the c
```

Set the `--as` and `--as-group` flag:

```
kubectl drain mynode --as=superman --as-group=system:masters
```

```
node/mynode cordoned
node/mynode drained
```

**Note:**

`kubectl` cannot impersonate extra fields or UIDs.

To impersonate a user, group, user identifier (UID) or extra fields, the impersonating user must have the ability to perform the "impersonate" verb on the kind of attribute being impersonated ("user", "group", "uid", etc.). For clusters that enable the RBAC authorization plugin, the following

ClusterRole encompasses the rules needed to set user and group impersonation headers:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: impersonator
rules:
- apiGroups: [""]
  resources: ["users", "groups", "serviceaccounts"]
  verbs: ["impersonate"]
```

For impersonation, extra fields and impersonated UIDs are both under the "authentication.k8s.io" apiGroup . Extra fields are evaluated as sub-resources of the resource "userextras". To allow a user to use impersonation headers for the extra field "scopes" and for UIDs, a user should be granted the following role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: scopes-and-uid-impersonator
rules:
# Can set "Impersonate-Extra-scopes" header and the "Impersonate-Uid"
- apiGroups: ["authentication.k8s.io"]
  resources: ["userextras/scopes", "uids"]
  verbs: ["impersonate"]
```

The values of impersonation headers can also be restricted by limiting the set of `resourceNames` a resource can take.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: limited-impersonator
rules:
# Can impersonate the user "jane.doe@example.com"
- apiGroups: [""]
  resources: ["users"]
  verbs: ["impersonate"]
  resourceNames: ["jane.doe@example.com"]

# Can impersonate the groups "developers" and "admins"
- apiGroups: [""]
  resources: ["groups"]
  verbs: ["impersonate"]
  resourceNames: ["developers", "admins"]

# Can impersonate the extras field "scopes" with the values "view" and "development"
- apiGroups: ["authentication.k8s.io"]
  resources: ["userextras/scopes"]
  verbs: ["impersonate"]
  resourceNames: ["view", "development"]

# Can impersonate the uid "06f6ce97-e2c5-4ab8-7ba5-7654dd08d52b"
- apiGroups: ["authentication.k8s.io"]
  resources: ["uids"]
  verbs: ["impersonate"]
  resourceNames: ["06f6ce97-e2c5-4ab8-7ba5-7654dd08d52b"]
```

**Note:**

Impersonating a user or group allows you to perform any action as if you were that user or group; for that reason, impersonation is not namespace scoped. If you want to allow impersonation using Kubernetes RBAC, this requires using a [ClusterRole](#) and a [ClusterRoleBinding](#), not a [Role](#) and [RoleBinding](#).

## client-go credential plugins

### ⓘ FEATURE STATE: Kubernetes v1.22 [stable]

`k8s.io/client-go` and tools using it such as `kubectl` and `kubelet` are able to execute an external command to receive user credentials.

This feature is intended for client side integrations with authentication protocols not natively supported by `k8s.io/client-go` (LDAP, Kerberos, OAuth2, SAML, etc.). The plugin implements the protocol specific logic, then returns opaque credentials to use. Almost all credential plugin use cases require a server side component with support for the [webhook token authenticator](#) to interpret the credential format produced by the client plugin.

**Note:**

Earlier versions of `kubectl` included built-in support for authenticating to AKS and GKE, but this is no longer present.

### Example use case

In a hypothetical use case, an organization would run an external service that exchanges LDAP credentials for user specific, signed tokens. The service would also be capable of responding to [webhook token authenticator](#) requests to validate the tokens. Users would be required to install a credential plugin on their workstation.

To authenticate against the API:

- The user issues a `kubectl` command.
- Credential plugin prompts the user for LDAP credentials, exchanges credentials with external service for a token.
- Credential plugin returns token to client-go, which uses it as a bearer token against the API server.
- API server uses the [webhook token authenticator](#) to submit a `TokenReview` to the external service.
- External service verifies the signature on the token and returns the user's username and groups.

### Configuration

Credential plugins are configured through [kubectl config files](#) as part of the user fields.

[client.authentication.k8s.io/v1](#)

[client.authentication.k8s.io/v1beta1](#)

```
apiVersion: v1
kind: Config
users:
- name: my-user
  user:
    exec:
      # Command to execute. Required.
      command: "example-client-go-exec-plugin"

      # API version to use when decoding the ExecCredentials resource. Required.
      #
      # The API version returned by the plugin MUST match the version listed here.
      #
      # To integrate with tools that support multiple versions (such as client.authentication.k8s.io/v1), set an environment variable, pass an argument to the tool that indicates which version to use, or read the version from the ExecCredential object in the KUBERNETES_EXEC_INFO environment variable.
      apiVersion: "client.authentication.k8s.io/v1"

      # Environment variables to set when executing the plugin. Optional.
      env:
        - name: "FOO"
          value: "bar"

      # Arguments to pass when executing the plugin. Optional.
      args:
        - "arg1"
        - "arg2"

      # Text shown to the user when the executable doesn't seem to be present. Optional.
      installHint: |
        example-client-go-exec-plugin is required to authenticate
        to the current cluster. It can be installed:

        On macOS: brew install example-client-go-exec-plugin

        On Ubuntu: apt-get install example-client-go-exec-plugin

        On Fedora: dnf install example-client-go-exec-plugin

        ...

      # Whether or not to provide cluster information, which could potentially contain very large CA data, to this exec plugin as a part of the KUBERNETES_EXEC_INFO environment variable.
      provideClusterInfo: true

      # The contract between the exec plugin and the standard input I/O stream. If this contract cannot be satisfied, this plugin will not be run and an error will be returned. Valid values are "Never" (this exec plugin never uses standard input), "IfAvailable" (this exec plugin wants to use standard input if it is available) or "Always" (this exec plugin requires standard input to function). Required.
      interactiveMode: Never

clusters:
- name: my-cluster
  cluster:
    server: "https://172.17.4.100:6443"
    certificate-authority: "/etc/kubernetes/ca.pem"
    extensions:
      - name: client.authentication.k8s.io/exec # reserved extension name for per cluster exec
        extension:
          arbitrary: config
          this: can be provided via the KUBERNETES_EXEC_INFO environment variable upon
          you: ["can", "put", "anything", "here"]
contexts:
- name: my-cluster
```

```
context:  
  cluster: my-cluster  
  user: my-user  
current-context: my-cluster
```

Relative command paths are interpreted as relative to the directory of the config file. If KUBECONFIG is set to `/home/jane/kubeconfig` and the exec command is `./bin/example-client-go-exec-plugin`, the binary `/home/jane/bin/example-client-go-exec-plugin` is executed.

```
- name: my-user  
  user:  
    exec:  
      # Path relative to the directory of the kubeconfig  
      command: "./bin/example-client-go-exec-plugin"  
      apiVersion: "client.authentication.k8s.io/v1"  
      interactiveMode: Never
```

## Input and output formats

The executed command prints an `ExecCredential` object to `stdout`. `k8s.io/client-go` authenticates against the Kubernetes API using the returned credentials in the `status`. The executed command is passed an `ExecCredential` object as input via the `KUBERNETES_EXEC_INFO` environment variable. This input contains helpful information like the expected API version of the returned `ExecCredential` object and whether or not the plugin can use `stdin` to interact with the user.

When run from an interactive session (i.e., a terminal), `stdin` can be exposed directly to the plugin. Plugins should use the `spec.interactive` field of the input `ExecCredential` object from the `KUBERNETES_EXEC_INFO` environment variable in order to determine if `stdin` has been provided. A plugin's `stdin` requirements (i.e., whether `stdin` is optional, strictly required, or never used in order for the plugin to run successfully) is declared via the `user.exec.interactiveMode` field in the [kubeconfig](#) (see table below for valid values). The `user.exec.interactiveMode` field is optional in `client.authentication.k8s.io/v1beta1` and required in `client.authentication.k8s.io/v1`.

interactiveMode	Value	Meaning
Never		This exec plugin never needs to use standard input, and therefore the exec plugin will be run regardless of whether standard input is available for user input.
IfAvailable		This exec plugin would like to use standard input if it is available, but can still operate if standard input is not available. Therefore, the exec plugin will be run regardless of whether <code>stdin</code> is available for user input. If standard input is available for user input, then it will be provided to this exec plugin.
Always		This exec plugin requires standard input in order to run, and therefore the exec plugin will only be run if standard input is available for user input. If standard input is not available for user input, then the exec plugin will not be run and an error will be returned by the exec plugin runner.

To use bearer token credentials, the plugin returns a token in the status of the [ExecCredential](#)

[client.authentication.k8s.io/v1](#)[client.authentication.k8s.io/v1beta1](#)

```
{  
  "apiVersion": "client.authentication.k8s.io/v1",  
  "kind": "ExecCredential",  
  "status": {  
    "token": "my-bearer-token"  
  }  
}
```

Alternatively, a PEM-encoded client certificate and key can be returned to use TLS client auth. If the plugin returns a different certificate and key on a subsequent call, `k8s.io/client-go` will close existing connections with the server to force a new TLS handshake.

If specified, `clientKeyData` and `clientCertificateData` must both must be present.

`clientCertificateData` may contain additional intermediate certificates to send to the server.

[client.authentication.k8s.io/v1](#)[client.authentication.k8s.io/v1beta1](#)

```
{  
  "apiVersion": "client.authentication.k8s.io/v1",  
  "kind": "ExecCredential",  
  "status": {  
    "clientCertificateData": "-----BEGIN CERTIFICATE-----\n...END CERTIFICATE-----",  
    "clientKeyData": "-----BEGIN RSA PRIVATE KEY-----\n...END RSA PRIVATE KEY-----"  
  }  
}
```

Optionally, the response can include the expiry of the credential formatted as a [RFC 3339](#) timestamp.

Presence or absence of an expiry has the following impact:

- If an expiry is included, the bearer token and TLS credentials are cached until the expiry time is reached, or if the server responds with a 401 HTTP status code, or when the process exits.
- If an expiry is omitted, the bearer token and TLS credentials are cached until the server responds with a 401 HTTP status code or until the process exits.

[client.authentication.k8s.io/v1](#)[client.authentication.k8s.io/v1beta1](#)

```
{  
  "apiVersion": "client.authentication.k8s.io/v1",  
  "kind": "ExecCredential",  
  "status": {  
    "token": "my-bearer-token",  
    "expirationTimestamp": "2018-03-05T17:30:20-08:00"  
  }  
}
```

To enable the exec plugin to obtain cluster-specific information, set `provideClusterInfo` on the `user.exec` field in the [kubeconfig](#). The plugin

will then be supplied this cluster-specific information in the `KUBERNETES_EXEC_INFO` environment variable. Information from this environment variable can be used to perform cluster-specific credential acquisition logic. The following `ExecCredential` manifest describes a cluster information sample.

[client.authentication.k8s.io/v1](#)

[client.authentication.k8s.io/v1beta1](#)

```
{  
  "apiVersion": "client.authentication.k8s.io/v1",  
  "kind": "ExecCredential",  
  "spec": {  
    "cluster": {  
      "server": "https://172.17.4.100:6443",  
      "certificate-authority-data": "LS0t...=",  
      "config": {  
        "arbitrary": "config",  
        "this": "can be provided via the KUBERNETES_EXEC_INFO environment variable",  
        "you": ["can", "put", "anything", "here"]  
      }  
    },  
    "interactive": true  
  }  
}
```

## API access to authentication information for a client

ⓘ **FEATURE STATE:** Kubernetes v1.28 [stable]

If your cluster has the API enabled, you can use the `SelfSubjectReview` API to find out how your Kubernetes cluster maps your authentication information to identify you as a client. This works whether you are authenticating as a user (typically representing a real person) or as a `ServiceAccount`.

`SelfSubjectReview` objects do not have any configurable fields. On receiving a request, the Kubernetes API server fills the status with the user attributes and returns it to the user.

Request example (the body would be a `SelfSubjectReview`):

```
POST /apis/authentication.k8s.io/v1/selfsubjectreviews
```

```
{  
  "apiVersion": "authentication.k8s.io/v1",  
  "kind": "SelfSubjectReview"  
}
```

Response example:

```
{  
  "apiVersion": "authentication.k8s.io/v1",  
  "kind": "SelfSubjectReview",  
  "status": {  
    "userInfo": {  
      "name": "jane.doe",  
      "uid": "b6c7cf4-f166-11ec-8ea0-0242ac120002",  
      "groups": [  
        "viewers",  
        "editors",  
        "system:authenticated"  
      ],  
      "extra": {  
        "provider_id": ["token.company.example"]  
      }  
    }  
  }  
}
```

For convenience, the `kubectl auth whoami` command is present. Executing this command will produce the following output (yet different user attributes will be shown):

- Simple output example

ATTRIBUTE	VALUE
Username	jane.doe
Groups	[system:authenticated]

- Complex example including extra attributes

ATTRIBUTE	VALUE
Username	jane.doe
UID	b79dbf30-0c6a-11ed-861d-0242ac120002
Groups	[students teachers system:authenticated]
Extra: skills	[reading learning]
Extra: subjects	[math sports]

By providing the `output` flag, it is also possible to print the JSON or YAML representation of the result:

[JSON](#) [YAML](#)

```
{  
  "apiVersion": "authentication.k8s.io/v1",  
  "kind": "SelfSubjectReview",  
  "status": {  
    "userInfo": {  
      "username": "jane.doe",  
      "uid": "b79dbf30-0c6a-11ed-861d-0242ac120002",  
      "groups": [  
        "students",  
        "teachers",  
        "system:authenticated"  
      ],  
      "extra": {  
        "skills": [  
          "reading",  
          "learning"  
        ],  
        "subjects": [  
          "math",  
          "sports"  
        ]  
      }  
    }  
  }  
}
```

This feature is extremely useful when a complicated authentication flow is used in a Kubernetes cluster, for example, if you use [webhook token authentication](#) or [authenticating proxy](#).

**Note:**

The Kubernetes API server fills the `userInfo` after all authentication mechanisms are applied, including [impersonation](#). If you, or an authentication proxy, make a `SelfSubjectReview` using impersonation, you see the user details and properties for the user that was impersonated.

By default, all authenticated users can create `SelfSubjectReview` objects when the `APISelfSubjectReview` feature is enabled. It is allowed by the `system:basic-user` cluster role.

**Note:**

You can only make `SelfSubjectReview` requests if:

- the `APISelfSubjectReview` [feature gate](#) is enabled for your cluster (not needed for Kubernetes 1.31, but older Kubernetes versions might not offer this feature gate, or might default it to be off)
- (if you are running a version of Kubernetes older than v1.28) the API server for your cluster has the `authentication.k8s.io/v1alpha1` or `authentication.k8s.io/v1beta1` API group enabled.

## What's next

- Read the [client authentication reference \(v1beta1\)](#)
- Read the [client authentication reference \(v1\)](#)

## 3.2 - Authenticating with Bootstrap Tokens

ⓘ **FEATURE STATE:** Kubernetes v1.18 [stable]

Bootstrap tokens are a simple bearer token that is meant to be used when creating new clusters or joining new nodes to an existing cluster. It was built to support [kubeadm](#), but can be used in other contexts for users that wish to start clusters without `kubeadm`. It is also built to work, via RBAC policy, with the [kubelet TLS Bootstrapping](#) system.

### Bootstrap Tokens Overview

Bootstrap Tokens are defined with a specific type (`bootstrap.kubernetes.io/token`) of secrets that lives in the `kube-system` namespace. These Secrets are then read by the Bootstrap Authenticator in the API Server. Expired tokens are removed with the TokenCleaner controller in the Controller Manager. The tokens are also used to create a signature for a specific ConfigMap used in a "discovery" process through a BootstrapSigner controller.

### Token Format

Bootstrap Tokens take the form of `abcdef.0123456789abcdef`. More formally, they must match the regular expression `[a-z0-9]{6}\.[a-z0-9]{16}`.

The first part of the token is the "Token ID" and is considered public information. It is used when referring to a token without leaking the secret part used for authentication. The second part is the "Token Secret" and should only be shared with trusted parties.

### Enabling Bootstrap Token Authentication

The Bootstrap Token authenticator can be enabled using the following flag on the API server:

```
--enable-bootstrap-token-auth
```

When enabled, bootstrapping tokens can be used as bearer token credentials to authenticate requests against the API server.

```
Authorization: Bearer 07401b.f395accd246ae52d
```

Tokens authenticate as the username `system:bootstrap:<token id>` and are members of the group `system:bootstrappers`. Additional groups may be specified in the token's Secret.

Expired tokens can be deleted automatically by enabling the `tokencleaner` controller on the controller manager.

```
--controllers=*,tokencleaner
```

## Bootstrap Token Secret Format

Each valid token is backed by a secret in the `kube-system` namespace. You can find the full design doc [here](#).

Here is what the secret looks like.

```
apiVersion: v1
kind: Secret
metadata:
  # Name MUST be of form "bootstrap-token-<token id>"
  name: bootstrap-token-07401b
  namespace: kube-system

  # Type MUST be 'bootstrap.kubernetes.io/token'
  type: bootstrap.kubernetes.io/token
  stringData:
    # Human readable description. Optional.
    description: "The default bootstrap token generated by 'kubeadm ini

    # Token ID and secret. Required.
    token-id: 07401b
    token-secret: f395accd246ae52d

    # Expiration. Optional.
    expiration: 2017-03-10T03:22:11Z

    # Allowed usages.
    usage-bootstrap-authentication: "true"
    usage-bootstrap-signing: "true"

  # Extra groups to authenticate the token as. Must start with "system:
  auth-extra-groups: system:bootstrappers:worker,system:bootstrappers
```

The type of the secret must be `bootstrap.kubernetes.io/token` and the name must be `bootstrap-token-<token id>`. It must also exist in the `kube-system` namespace.

The `usage-bootstrap-*` members indicate what this secret is intended to be used for. A value must be set to `true` to be enabled.

- `usage-bootstrap-authentication` indicates that the token can be used to authenticate to the API server as a bearer token.
- `usage-bootstrap-signing` indicates that the token may be used to sign the `cluster-info` ConfigMap as described below.

The `expiration` field controls the expiry of the token. Expired tokens are rejected when used for authentication and ignored during ConfigMap signing. The expiry value is encoded as an absolute UTC time using RFC3339. Enable the `tokencleaner` controller to automatically delete expired tokens.

## Token Management with kubeadm

You can use the `kubeadm` tool to manage tokens on a running cluster. See

the [kubeadm token docs](#) for details.

## ConfigMap Signing

In addition to authentication, the tokens can be used to sign a ConfigMap. This is used early in a cluster bootstrap process before the client trusts the API server. The signed ConfigMap can be authenticated by the shared token.

Enable ConfigMap signing by enabling the `bootstrapsigner` controller on the Controller Manager.

```
--controllers=*,bootstrapsigner
```

The ConfigMap that is signed is `cluster-info` in the `kube-public` namespace. The typical flow is that a client reads this ConfigMap while unauthenticated and ignoring TLS errors. It then validates the payload of the ConfigMap by looking at a signature embedded in the ConfigMap.

The ConfigMap may look like this:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-info
  namespace: kube-public
data:
  jws-kubeconfig-07401b: eyJhbGciOiJIUzI1NiIsImtpZCI6IjA3NDAxYiJ9..ty
  kubeconfig: |
    apiVersion: v1
    clusters:
    - cluster:
        certificate-authority-data: <really long certificate data>
        server: https://10.138.0.2:6443
        name: ""
    contexts: []
    current-context: ""
    kind: Config
    preferences: {}
    users: []
```

The `kubeconfig` member of the ConfigMap is a config file with only the cluster information filled out. The key thing being communicated here is the `certificate-authority-data`. This may be expanded in the future.

The signature is a JWS signature using the "detached" mode. To validate the signature, the user should encode the `kubeconfig` payload according to JWS rules (base64 encoded while discarding any trailing `=`). That encoded payload is then used to form a whole JWS by inserting it between the 2 dots. You can verify the JWS using the `HS256` scheme (HMAC-SHA256) with the full token (e.g. `07401b.f395accd246ae52d`) as the shared secret. Users *must* verify that HS256 is used.

**Warning:**

Any party with a bootstrapping token can create a valid signature for that token. When using ConfigMap signing it's discouraged to share the same token with many clients, since a compromised client can potentially man-in-the middle another client relying on the signature to bootstrap TLS trust.

Consult the [kubeadm implementation details](#) section for more information.

## 3.3 - Authorization

Details of Kubernetes authorization mechanisms and supported authorization modes.

Kubernetes authorization takes place following [authentication](#). Usually, a client making a request must be authenticated (logged in) before its request can be allowed; however, Kubernetes also allows anonymous requests in some circumstances.

For an overview of how authorization fits into the wider context of API access control, read [Controlling Access to the Kubernetes API](#).

### Authorization verdicts

Kubernetes authorization of API requests takes place within the API server. The API server evaluates all of the request attributes against all policies, potentially also consulting external services, and then allows or denies the request.

All parts of an API request must be allowed by some authorization mechanism in order to proceed. In other words: access is denied by default.

**Note:**

Access controls and policies that depend on specific fields of specific kinds of objects are handled by [admission controllers](#).

Kubernetes admission control happens after authorization has completed (and, therefore, only when the authorization decision was to allow the request).

When multiple [authorization modules](#) are configured, each is checked in sequence. If any authorizer *approves* or *denies* a request, that decision is immediately returned and no other authorizer is consulted. If all modules have *no opinion* on the request, then the request is denied. An overall deny verdict means that the API server rejects the request and responds with an HTTP 403 (Forbidden) status.

### Request attributes used in authorization

Kubernetes reviews only the following API request attributes:

- **user** - The `user` string provided during authentication.
- **group** - The list of group names to which the authenticated user belongs.
- **extra** - A map of arbitrary string keys to string values, provided by the authentication layer.
- **API** - Indicates whether the request is for an API resource.
- **Request path** - Path to miscellaneous non-resource endpoints like `/api` or `/healthz`.
- **API request verb** - API verbs like `get`, `list`, `create`, `update`, `patch`, `watch`, `delete`, and `deletecollection` are used for resource requests. To determine the request verb for a resource API endpoint, see [request verbs and authorization](#).

- **HTTP request verb** - Lowercased HTTP methods like `get`, `post`, `put`, and `delete` are used for non-resource requests.
- **Resource** - The ID or name of the resource that is being accessed (for resource requests only) -- For resource requests using `get`, `update`, `patch`, and `delete` verbs, you must provide the resource name.
- **Subresource** - The subresource that is being accessed (for resource requests only).
- **Namespace** - The namespace of the object that is being accessed (for namespaced resource requests only).
- **API group** - The API Group being accessed (for resource requests only). An empty string designates the core API group.

## Request verbs and authorization

### Non-resource requests

Requests to endpoints other than `/api/v1/...` or `/apis/<group>/<version>/...` are considered *non-resource requests*, and use the lower-cased HTTP method of the request as the verb. For example, making a `GET` request using HTTP to endpoints such as `/api` or `/healthz` would use `get` as the verb.

### Resource requests

To determine the request verb for a resource API endpoint, Kubernetes maps the HTTP verb used and considers whether or not the request acts on an individual resource or on a collection of resources:

HTTP verb	request verb
POST	<b>create</b>
GET , HEAD	<b>get</b> (for individual resources), <b>list</b> (for collections, including full object content), <b>watch</b> (for watching an individual resource or collection of resources)
PUT	<b>update</b>
PATCH	<b>patch</b>
DELETE	<b>delete</b> (for individual resources), <b>deletecollection</b> (for collections)

#### Caution:

+The **get**, **list** and **watch** verbs can all return the full details of a resource. In terms of access to the returned data they are equivalent. For example, **list** on **secrets** will reveal the **data** attributes of any returned resources.

Kubernetes sometimes checks authorization for additional permissions using specialized verbs. For example:

- Special cases of [authentication](#)
  - **impersonate** verb on `users`, `groups`, and `serviceaccounts` in the core API group, and the `userextras` in the `authentication.k8s.io` API group.

- [Authorization of CertificateSigningRequests](#)
  - **approve** verb for CertificateSigningRequests, and **update** for revisions to existing approvals
- [RBAC](#)
  - **bind** and **escalate** verbs on `roles` and `clusterroles` resources in the `rbac.authorization.k8s.io` API group.

## Authorization context

Kubernetes expects attributes that are common to REST API requests. This means that Kubernetes authorization works with existing organization-wide or cloud-provider-wide access control systems which may handle other APIs besides the Kubernetes API.

## Authorization modes

The Kubernetes API server may authorize a request using one of several authorization modes:

### **AlwaysAllow**

This mode allows all requests, which brings [security risks](#). Use this authorization mode only if you do not require authorization for your API requests (for example, for testing).

### **AlwaysDeny**

This mode blocks all requests. Use this authorization mode only for testing.

### [ABAC \(attribute-based access control\)](#)

Kubernetes ABAC mode defines an access control paradigm whereby access rights are granted to users through the use of policies which combine attributes together. The policies can use any type of attributes (user attributes, resource attributes, object, environment attributes, etc).

### [RBAC \(role-based access control\)](#)

Kubernetes RBAC is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise. In this context, access is the ability of an individual user to perform a specific task, such as view, create, or modify a file. In this mode, Kubernetes uses the `rbac.authorization.k8s.io` API group to drive authorization decisions, allowing you to dynamically configure permission policies through the Kubernetes API.

### **Node**

A special-purpose authorization mode that grants permissions to kubelets based on the pods they are scheduled to run. To learn more about the Node authorization mode, see [Node Authorization](#).

### **Webhook**

Kubernetes [webhook mode](#) for authorization makes a synchronous HTTP callout, blocking the request until the remote HTTP service responds to the query. You can write your own software to handle the callout, or use solutions from the ecosystem.

**Warning:**

Enabling the `AlwaysAllow` mode bypasses authorization; do not use this on a cluster where you do not trust **all** potential API clients, including the workloads that you run.

Authorization mechanisms typically return either a *deny* or *no opinion* result; see [authorization verdicts](#) for more on this. Activating the `AlwaysAllow` means that if all other authorizers return “no opinion”, the request is allowed. For example, `--authorization-mode=AlwaysAllow,RBAC` has the same effect as `--authorization-mode=AlwaysAllow` because Kubernetes RBAC does not provide negative (deny) access rules.

You should not use the `AlwaysAllow` mode on a Kubernetes cluster where the API server is reachable from the public internet.

## Authorization mode configuration

You can configure the Kubernetes API server's authorizer chain using either [command line arguments](#) only or, as a beta feature, using a [configuration file](#).

You have to pick one of the two configuration approaches; setting both `--authorization-config` path and configuring an authorization webhook using the `--authorization-mode` and `--authorization-webhook-*` command line arguments is not allowed. If you try this, the API server reports an error message during startup, then exits immediately.

## Command line authorization mode configuration

ⓘ **FEATURE STATE:** Kubernetes v1.8 [stable]

You can use the following modes:

- `--authorization-mode=ABAC` (Attribute-based access control mode)
- `--authorization-mode=RBAC` (Role-based access control mode)
- `--authorization-mode=Node` (Node authorizer)
- `--authorization-mode=Webhook` (Webhook authorization mode)
- `--authorization-mode=AlwaysAllow` (always allows requests; carries [security risks](#))
- `--authorization-mode=AlwaysDeny` (always denies requests)

You can choose more than one authorization mode; for example: `--authorization-mode=Node,Webhook`

Kubernetes checks authorization modules based on the order that you specify them on the API server's command line, so an earlier module has higher priority to allow or deny a request.

You cannot combine the `--authorization-mode` command line argument with the `--authorization-config` command line argument used for [configuring authorization using a local file](#).

For more information on command line arguments to the API server, read the [kube-apiserver reference](#).

## Configuring the API Server using an authorization config file

ⓘ **FEATURE STATE:** Kubernetes v1.30 [beta]

As a beta feature, Kubernetes lets you configure authorization chains that can include multiple webhooks. The authorization items in that chain can have well-defined parameters that validate requests in a particular order, offering you fine-grained control, such as explicit Deny on failures.

The configuration file approach even allows you to specify [CEL](#) rules to pre-filter requests before they are dispatched to webhooks, helping you to prevent unnecessary invocations. The API server also automatically reloads the authorizer chain when the configuration file is modified.

You specify the path to the authorization configuration using the `--authorization-config` command line argument.

If you want to use command line arguments instead of a configuration file, that's also a valid and supported approach. Some authorization capabilities (for example: multiple webhooks, webhook failure policy, and pre-filter rules) are only available if you use an authorization configuration file.

## Example configuration

```
---  
#  
# DO NOT USE THE CONFIG AS IS. THIS IS AN EXAMPLE.  
#  
apiVersion: apiserver.config.k8s.io/v1beta1  
kind: AuthorizationConfiguration  
authorizers:  
  - type: Webhook  
    # Name used to describe the authorizer  
    # This is explicitly used in monitoring machinery for metrics  
    # Note:  
    #   - Validation for this field is similar to how K8s Labels are  
    # Required, with no default  
    name: webhook  
    webhook:  
      # The duration to cache 'authorized' responses from the webhook  
      # authorizer.  
      # Same as setting `--authorization-webhook-cache-authorized-ttl`  
      # Default: 5m0s  
      authorizedTTL: 30s  
      # The duration to cache 'unauthorized' responses from the webhook  
      # authorizer.  
      # Same as setting `--authorization-webhook-cache-unauthorized-ttl`  
      # Default: 30s  
      unauthorizedTTL: 30s  
      # Timeout for the webhook request  
      # Maximum allowed is 30s.  
      # Required, with no default.  
      timeout: 3s  
      # The API version of the authorization.k8s.io SubjectAccessReview  
      # send to and expect from the webhook.  
      # Same as setting `--authorization-webhook-version` flag  
      # Required, with no default  
      # Valid values: v1beta1, v1  
      subjectAccessReviewVersion: v1  
      # MatchConditionSubjectAccessReviewVersion specifies the SubjectAccessReview  
      # version the CEL expressions are evaluated against  
      # Valid values: v1  
      # Required, no default value  
      matchConditionSubjectAccessReviewVersion: v1  
      # Controls the authorization decision when a webhook request fails  
      # complete or returns a malformed response or errors evaluating  
      # matchConditions.  
      # Valid values:  
      #   - NoOpinion: continue to subsequent authorizers to see if one  
      #     them allows the request  
      #   - Deny: reject the request without consulting subsequent authorizers  
      # Required, with no default.  
      failurePolicy: Deny  
      connectionInfo:  
        # Controls how the webhook should communicate with the server  
        # Valid values:  
        # - KubeConfig: use the file specified in kubeConfigFile to connect  
        #   to the server.  
        # - InClusterConfig: use the in-cluster configuration to call  
        #   the SubjectAccessReview API hosted by kube-apiserver. This mode  
        #   is only allowed for kube-apiserver.  
        type: KubeConfig  
        # Path to KubeConfigFile for connection info  
        # Required, if connectionInfo.Type is KubeConfig  
        kubeConfigFile: /kube-system-authz-webhook.yaml  
        # matchConditions is a list of conditions that must be met for  
        # webhook. An empty list of matchConditions matches all requests.  
        # There are a maximum of 64 match conditions allowed.  
        #  
        # The exact matching logic is (in order):
```

```
# 1. If at least one matchCondition evaluates to FALSE, the
# 2. If ALL matchConditions evaluate to TRUE, then the webhook
# 3. If at least one matchCondition evaluates to an error (
#     - If failurePolicy=Deny, then the webhook rejects the
#     - If failurePolicy=NoOpinion, then the error is ignored
matchConditions:
  # expression represents the expression which will be evaluated
  # CEL expressions have access to the contents of the SubjectAccessReview
  # If version specified by subjectAccessReviewVersion in the request
  # the contents would be converted to the v1 version before evaluation
  #
  # Documentation on CEL: https://kubernetes.io/docs/reference/using-api/celexpressions
  #
  # only send resource requests to the webhook
  - expression: has(request.resourceAttributes)
  # only intercept requests to kube-system
  - expression: request.resourceAttributes.namespace == 'kube-system'
  # don't intercept requests from kube-system service accounts
  - expression: !(system:serviceaccounts:kube-system' in request
  - type: Node
    name: node
  - type: RBAC
    name: rbac
  - type: Webhook
    name: in-cluster-authorizer
    webhook:
      authorizedTTL: 5m
      unauthorizedTTL: 30s
      timeout: 3s
      subjectAccessReviewVersion: v1
      failurePolicy: NoOpinion
      connectionInfo:
        type: InClusterConfig
```

When configuring the authorizer chain using a configuration file, make sure all the control plane nodes have the same file contents. Take a note of the API server configuration when upgrading / downgrading your clusters. For example, if upgrading from Kubernetes 1.30 to Kubernetes 1.31, you would need to make sure the config file is in a format that Kubernetes 1.31 can understand, before you upgrade the cluster. If you downgrade to 1.30, you would need to set the configuration appropriately.

## Authorization configuration and reloads

Kubernetes reloads the authorization configuration file when the API server observes a change to the file, and also on a 60 second schedule if no change events were observed.

### Note:

You must ensure that all non-webhook authorizer types remain unchanged in the file on reload.

A reload **must not** add or remove Node or RBAC authorizers (they can be reordered, but cannot be added or removed).

## Privilege escalation via workload creation or edits

Users who can create/edit pods in a namespace, either directly or

through an object that enables indirect [workload management](#), may be able to escalate their privileges in that namespace. The potential routes to privilege escalation include Kubernetes [API extensions](#) and their associated controllers.

**Caution:**

As a cluster administrator, use caution when granting access to create or edit workloads. Some details of how these can be misused are documented in [escalation paths](#).

## Escalation paths

There are different ways that an attacker or untrustworthy user could gain additional privilege within a namespace, if you allow them to run arbitrary Pods in that namespace:

- Mounting arbitrary Secrets in that namespace
  - Can be used to access confidential information meant for other workloads
  - Can be used to obtain a more privileged ServiceAccount's service account token
- Using arbitrary ServiceAccounts in that namespace
  - Can perform Kubernetes API actions as another workload (impersonation)
  - Can perform any privileged actions that ServiceAccount has
- Mounting or using ConfigMaps meant for other workloads in that namespace
  - Can be used to obtain information meant for other workloads, such as database host names.
- Mounting volumes meant for other workloads in that namespace
  - Can be used to obtain information meant for other workloads, and change it.

**Caution:**

As a system administrator, you should be cautious when deploying CustomResourceDefinitions that let users make changes to the above areas. These may open privilege escalation paths. Consider the consequences of this kind of change when deciding on your authorization controls.

## Checking API access

`kubectl` provides the `auth can-i` subcommand for quickly querying the API authorization layer. The command uses the `SelfSubjectAccessReview` API to determine if the current user can perform a given action, and works regardless of the authorization mode used.

```
kubectl auth can-i create deployments --namespace dev
```

The output is similar to this:

```
yes
```

```
kubectl auth can-i create deployments --namespace prod
```

The output is similar to this:

```
no
```

Administrators can combine this with [user impersonation](#) to determine what action other users can perform.

```
kubectl auth can-i list secrets --namespace dev --as dave
```

The output is similar to this:

```
no
```

Similarly, to check whether a ServiceAccount named `dev-sa` in Namespace `dev` can list Pods in the Namespace `target`:

```
kubectl auth can-i list pods \
  --namespace target \
  --as system:serviceaccount:dev:dev-sa
```

The output is similar to this:

```
yes
```

`SelfSubjectAccessReview` is part of the `authorization.k8s.io` API group, which exposes the API server authorization to external services. Other resources in this group include:

### **SubjectAccessReview**

Access review for any user, not only the current one. Useful for delegating authorization decisions to the API server. For example, the kubelet and extension API servers use this to determine user access to their own APIs.

### **LocalSubjectAccessReview**

Like `SubjectAccessReview` but restricted to a specific namespace.

### **SelfSubjectRulesReview**

A review which returns the set of actions a user can perform within a namespace. Useful for users to quickly summarize their own access, or for UIs to hide/show actions.

These APIs can be queried by creating normal Kubernetes resources, where the response `status` field of the returned object is the result of the query. For example:

```
kubectl create -f - -o yaml << EOF
apiVersion: authorization.k8s.io/v1
kind: SelfSubjectAccessReview
spec:
  resourceAttributes:
    group: apps
    resource: deployments
    verb: create
    namespace: dev
EOF
```

The generated SelfSubjectAccessReview is similar to:

```
apiVersion: authorization.k8s.io/v1
kind: SelfSubjectAccessReview
metadata:
  creationTimestamp: null
spec:
  resourceAttributes:
    group: apps
    resource: deployments
    namespace: dev
    verb: create
status:
  allowed: true
  denied: false
```

## What's next

- To learn more about Authentication, see [Authentication](#).
- For an overview, read [Controlling Access to the Kubernetes API](#).
- To learn more about Admission Control, see [Using Admission Controllers](#).
- Read more about [Common Expression Language in Kubernetes](#).

## 3.4 - Using RBAC Authorization

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within your organization.

RBAC authorization uses the `rbac.authorization.k8s.io` API group to drive authorization decisions, allowing you to dynamically configure policies through the Kubernetes API.

To enable RBAC, start the API server with the `--authorization-mode` flag set to a comma-separated list that includes `RBAC`; for example:

```
kube-apiserver --authorization-mode=Example,RBAC --other-options --mo
```

## API objects

The RBAC API declares four kinds of Kubernetes object: *Role*, *ClusterRole*, *RoleBinding* and *ClusterRoleBinding*. You can describe or amend the RBAC objects using tools such as `kubectl`, just like any other Kubernetes object.

### Caution:

These objects, by design, impose access restrictions. If you are making changes to a cluster as you learn, see [privilege escalation prevention and bootstrapping](#) to understand how those restrictions can prevent you making some changes.

## Role and ClusterRole

An RBAC *Role* or *ClusterRole* contains rules that represent a set of permissions. Permissions are purely additive (there are no "deny" rules).

A Role always sets permissions within a particular namespace; when you create a Role, you have to specify the namespace it belongs in.

ClusterRole, by contrast, is a non-namespaced resource. The resources have different names (Role and ClusterRole) because a Kubernetes object always has to be either namespaced or not namespaced; it can't be both.

ClusterRoles have several uses. You can use a ClusterRole to:

1. define permissions on namespaced resources and be granted access within individual namespace(s)
2. define permissions on namespaced resources and be granted access across all namespaces
3. define permissions on cluster-scoped resources

If you want to define a role within a namespace, use a Role; if you want to define a role cluster-wide, use a ClusterRole.

## Role example

Here's an example Role in the "default" namespace that can be used to grant read access to pods:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

## ClusterRole example

A ClusterRole can be used to grant the same permissions as a Role. Because ClusterRoles are cluster-scoped, you can also use them to grant access to:

- cluster-scoped resources (like `nodes`)
- non-resource endpoints (like `/healthz`)
- namespaced resources (like Pods), across all namespaces

For example: you can use a ClusterRole to allow a particular user to run `kubectl get pods --all-namespaces`

Here is an example of a ClusterRole that can be used to grant read access to `secrets` in any particular namespace, or across all namespaces (depending on how it is [bound](#)):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  #
  # at the HTTP Level, the name of the resource for accessing Secret
  # objects is "secrets"
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

The name of a Role or a ClusterRole object must be a valid [path segment name](#).

## RoleBinding and ClusterRoleBinding

A role binding grants the permissions defined in a role to a user or set of users. It holds a list of *subjects* (users, groups, or service accounts), and a reference to the role being granted. A RoleBinding grants permissions within a specific namespace whereas a ClusterRoleBinding grants that access cluster-wide.

A RoleBinding may reference any Role in the same namespace. Alternatively, a RoleBinding can reference a ClusterRole and bind that ClusterRole to the namespace of the RoleBinding. If you want to bind a ClusterRole to all the namespaces in your cluster, you use a ClusterRoleBinding.

The name of a RoleBinding or ClusterRoleBinding object must be a valid [path segment name](#).

## RoleBinding examples

Here is an example of a RoleBinding that grants the "pod-reader" Role to the user "jane" within the "default" namespace. This allows "jane" to read pods in the "default" namespace.

```
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "jane" to read pods in the "default" namespace
# You need to already have a Role named "pod-reader" in that namespace
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
# You can specify more than one "subject"
- kind: User
  name: jane # "name" is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
# "roleRef" specifies the binding to a Role / ClusterRole
  kind: Role #this must be Role or ClusterRole
  name: pod-reader # this must match the name of the Role or ClusterRole
  apiGroup: rbac.authorization.k8s.io
```

A RoleBinding can also reference a ClusterRole to grant the permissions defined in that ClusterRole to resources inside the RoleBinding's namespace. This kind of reference lets you define a set of common roles across your cluster, then reuse them within multiple namespaces.

For instance, even though the following RoleBinding refers to a ClusterRole, "dave" (the subject, case sensitive) will only be able to read Secrets in the "development" namespace, because the RoleBinding's namespace (in its metadata) is "development".

```
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "dave" to read secrets in the "development" namespace
# You need to already have a ClusterRole named "secret-reader".
kind: RoleBinding
metadata:
  name: read-secrets
  #
# The namespace of the RoleBinding determines where the permissions are granted
# This only grants permissions within the "development" namespace.
  namespace: development
subjects:
- kind: User
  name: dave # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

## ClusterRoleBinding example

To grant permissions across a whole cluster, you can use a ClusterRoleBinding. The following ClusterRoleBinding allows any user in the group "manager" to read secrets in any namespace.

```
apiVersion: rbac.authorization.k8s.io/v1
# This cluster role binding allows anyone in the "manager" group to r
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

After you create a binding, you cannot change the Role or ClusterRole that it refers to. If you try to change a binding's `roleRef`, you get a validation error. If you do want to change the `roleRef` for a binding, you need to remove the binding object and create a replacement.

There are two reasons for this restriction:

1. Making `roleRef` immutable allows granting someone `update` permission on an existing binding object, so that they can manage the list of subjects, without being able to change the role that is granted to those subjects.
2. A binding to a different role is a fundamentally different binding. Requiring a binding to be deleted/recreated in order to change the `roleRef` ensures the full list of subjects in the binding is intended to be granted the new role (as opposed to enabling or accidentally modifying only the `roleRef` without verifying all of the existing subjects should be given the new role's permissions).

The `kubectl auth reconcile` command-line utility creates or updates a manifest file containing RBAC objects, and handles deleting and recreating binding objects if required to change the role they refer to. See [command usage and examples](#) for more information.

## Referring to resources

In the Kubernetes API, most resources are represented and accessed using a string representation of their object name, such as `pods` for a Pod. RBAC refers to resources using exactly the same name that appears in the URL for the relevant API endpoint. Some Kubernetes APIs involve a *subresource*, such as the logs for a Pod. A request for a Pod's logs looks like:

```
GET /api/v1/namespaces/{namespace}/pods/{name}/log
```

In this case, `pods` is the namespaced resource for Pod resources, and `log` is a subresource of `pods`. To represent this in an RBAC role, use a slash ( / ) to delimit the resource and subresource. To allow a subject to read `pods` and also access the `log` subresource for each of those Pods, you write:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-and-pod-logs-reader
rules:
- apiGroups: [""]
  resources: ["pods", "pods/log"]
  verbs: ["get", "list"]
```

You can also refer to resources by name for certain requests through the `resourceNames` list. When specified, requests can be restricted to individual instances of a resource. Here is an example that restricts its subject to only `get` or `update` a `ConfigMap` named `my-configmap`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: configmap-updater
rules:
- apiGroups: [""]
  #
  # at the HTTP Level, the name of the resource for accessing ConfigMap
  # objects is "configmaps"
  resources: ["configmaps"]
  resourceNames: ["my-configmap"]
  verbs: ["update", "get"]
```

**Note:**

You cannot restrict `create` or `deletecollection` requests by their resource name. For `create`, this limitation is because the name of the new object may not be known at authorization time. If you restrict `list` or `watch` by resourceName, clients must include a `metadata.name` field selector in their `list` or `watch` request that matches the specified resourceName in order to be authorized. For example, `kubectl get configmaps --field-selector=metadata.name=my-configmap`

Rather than referring to individual `resources`, `apiGroups`, and `verbs`, you can use the wildcard `*` symbol to refer to all such objects. For `nonResourceURLs`, you can use the wildcard `*` as a suffix glob match. For `resourceNames`, an empty set means that everything is allowed. Here is an example that allows access to perform any current and future action on all current and future resources in the `example.com` API group. This is similar to the built-in `cluster-admin` role.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: example.com-superuser # DO NOT USE THIS ROLE, IT IS JUST AN EXAMPLE
rules:
- apiGroups: ["example.com"]
  resources: ["*"]
  verbs: ["*"]
```

**Caution:**

Using wildcards in resource and verb entries could result in overly permissive access being granted to sensitive resources. For instance, if a new resource type is added, or a new subresource is added, or a new custom verb is checked, the wildcard entry automatically grants access, which may be undesirable. The [principle of least privilege](#) should be employed, using specific resources and verbs to ensure only the permissions required for the workload to function correctly are applied.

## Aggregated ClusterRoles

You can *aggregate* several ClusterRoles into one combined ClusterRole. A controller, running as part of the cluster control plane, watches for ClusterRole objects with an `aggregationRule` set. The `aggregationRule` defines a label `selector` that the controller uses to match other ClusterRole objects that should be combined into the `rules` field of this one.

**Caution:**

The control plane overwrites any values that you manually specify in the `rules` field of an aggregate ClusterRole. If you want to change or add rules, do so in the `ClusterRole` objects that are selected by the `aggregationRule`.

Here is an example aggregated ClusterRole:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: monitoring
aggregationRule:
  clusterRoleSelectors:
  - matchLabels:
    rbac.example.com/aggregate-to-monitoring: "true"
rules: [] # The control plane automatically fills in the rules
```

If you create a new ClusterRole that matches the label selector of an existing aggregated ClusterRole, that change triggers adding the new rules into the aggregated ClusterRole. Here is an example that adds rules to the "monitoring" ClusterRole, by creating another ClusterRole labeled `rbac.example.com/aggregate-to-monitoring: true`.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: monitoring-endpoints
  labels:
    rbac.example.com/aggregate-to-monitoring: "true"
  # When you create the "monitoring-endpoints" ClusterRole,
  # the rules below will be added to the "monitoring" ClusterRole.
rules:
- apiGroups: [""]
  resources: ["services", "endpointslices", "pods"]
  verbs: ["get", "list", "watch"]
```

The [default user-facing roles](#) use ClusterRole aggregation. This lets you, as a cluster administrator, include rules for custom resources, such as those served by CustomResourceDefinitions or aggregated API servers, to extend the default roles.

For example: the following ClusterRoles let the "admin" and "edit" default roles manage the custom resource named CronTab, whereas the "view" role can perform only read actions on CronTab resources. You can assume that CronTab objects are named "crontabs" in URLs as seen by the API server.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: aggregate-cron-tabs-edit
  labels:
    # Add these permissions to the "admin" and "edit" default roles.
    rbac.authorization.k8s.io/aggregate-to-admin: "true"
    rbac.authorization.k8s.io/aggregate-to-edit: "true"
rules:
- apiGroups: ["stable.example.com"]
  resources: ["crontabs"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: aggregate-cron-tabs-view
  labels:
    # Add these permissions to the "view" default role.
    rbac.authorization.k8s.io/aggregate-to-view: "true"
rules:
- apiGroups: ["stable.example.com"]
  resources: ["crontabs"]
  verbs: ["get", "list", "watch"]
```

## Role examples

The following examples are excerpts from Role or ClusterRole objects, showing only the `rules` section.

Allow reading "pods" resources in the core API Group:

```
rules:
- apiGroups: [""]
  #
  # at the HTTP Level, the name of the resource for accessing Pod
  # objects is "pods"
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```

Allow reading/writing Deployments (at the HTTP level: objects with "deployments" in the resource part of their URL) in the "apps" API groups:

```
rules:
- apiGroups: ["apps"]
  #
  # at the HTTP Level, the name of the resource for accessing Deployments
  # objects is "deployments"
  resources: ["deployments"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

Allow reading Pods in the core API group, as well as reading or writing Job resources in the "batch" API group:

```
rules:
- apiGroups: [""]
  #
  # at the HTTP Level, the name of the resource for accessing Pod
  # objects is "pods"
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["batch"]
  #
  # at the HTTP Level, the name of the resource for accessing Job
  # objects is "jobs"
  resources: ["jobs"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

Allow reading a ConfigMap named "my-config" (must be bound with a RoleBinding to limit to a single ConfigMap in a single namespace):

```
rules:
- apiGroups: [""]
  #
  # at the HTTP Level, the name of the resource for accessing ConfigMaps
  # objects is "configmaps"
  resources: ["configmaps"]
  resourceNames: ["my-config"]
  verbs: ["get"]
```

Allow reading the resource "nodes" in the core group (because a Node is cluster-scoped, this must be in a ClusterRole bound with a ClusterRoleBinding to be effective):

```
rules:
- apiGroups: [""]
  #
  # at the HTTP Level, the name of the resource for accessing Node
  # objects is "nodes"
  resources: ["nodes"]
  verbs: ["get", "list", "watch"]
```

Allow GET and POST requests to the non-resource endpoint /healthz and all subpaths (must be in a ClusterRole bound with a ClusterRoleBinding to be effective):

```
rules:
- nonResourceURLs: ["/healthz", "/healthz/*"] # '*' in a nonResourceURL
  verbs: ["get", "post"]
```

## Referring to subjects

A RoleBinding or ClusterRoleBinding binds a role to subjects. Subjects can be groups, users or [ServiceAccounts](#).

Kubernetes represents usernames as strings. These can be: plain names, such as "alice"; email-style names, like "bob@example.com"; or numeric user IDs represented as a string. It is up to you as a cluster administrator to configure the [authentication modules](#) so that authentication produces usernames in the format you want.

### Caution:

The prefix `system:` is reserved for Kubernetes system use, so you should ensure that you don't have users or groups with names that start with `system:` by accident. Other than this special prefix, the RBAC authorization system does not require any format for usernames.

In Kubernetes, Authenticator modules provide group information. Groups, like users, are represented as strings, and that string has no format requirements, other than that the prefix `system:` is reserved.

[ServiceAccounts](#) have names prefixed with `system:serviceaccount:` , and belong to groups that have names prefixed with `system:serviceaccounts:` .

### Note:

- `system:serviceaccount:` (singular) is the prefix for service account usernames.
- `system:serviceaccounts:` (plural) is the prefix for service account groups.

## RoleBinding examples

The following examples are `RoleBinding` excerpts that only show the `subjects` section.

For a user named `alice@example.com` :

```
subjects:
- kind: User
  name: "alice@example.com"
  apiGroup: rbac.authorization.k8s.io
```

For a group named `frontend-admins` :

```
subjects:
- kind: Group
  name: "frontend-admins"
  apiGroup: rbac.authorization.k8s.io
```

For the default service account in the "kube-system" namespace:

```
subjects:
- kind: ServiceAccount
  name: default
  namespace: kube-system
```

For all service accounts in the "qa" namespace:

```
subjects:
- kind: Group
  name: system:serviceaccounts:qa
  apiGroup: rbac.authorization.k8s.io
```

For all service accounts in any namespace:

```
subjects:
- kind: Group
  name: system:serviceaccounts
  apiGroup: rbac.authorization.k8s.io
```

For all authenticated users:

```
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
```

For all unauthenticated users:

```
subjects:
- kind: Group
  name: system:unauthenticated
  apiGroup: rbac.authorization.k8s.io
```

For all users:

```
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
- kind: Group
  name: system:unauthenticated
  apiGroup: rbac.authorization.k8s.io
```

## Default roles and role bindings

API servers create a set of default ClusterRole and ClusterRoleBinding objects. Many of these are `system:` prefixed, which indicates that the resource is directly managed by the cluster control plane. All of the

default ClusterRoles and ClusterRoleBindings are labeled with `kubernetes.io/bootstrapping=rbac-defaults` .

**Caution:**

Take care when modifying ClusterRoles and ClusterRoleBindings with names that have a `system:` prefix. Modifications to these resources can result in non-functional clusters.

## Auto-reconciliation

At each start-up, the API server updates default cluster roles with any missing permissions, and updates default cluster role bindings with any missing subjects. This allows the cluster to repair accidental modifications, and helps to keep roles and role bindings up-to-date as permissions and subjects change in new Kubernetes releases.

To opt out of this reconciliation, set the `rbac.authorization.kubernetes.io/autoupdate` annotation on a default cluster role or default cluster RoleBinding to `false` . Be aware that missing default permissions and subjects can result in non-functional clusters.

Auto-reconciliation is enabled by default if the RBAC authorizer is active.

## API discovery roles

Default cluster role bindings authorize unauthenticated and authenticated users to read API information that is deemed safe to be publicly accessible (including CustomResourceDefinitions). To disable anonymous unauthenticated access, add `--anonymous-auth=false` flag to the API server configuration.

To view the configuration of these roles via `kubectl` run:

```
kubectl get clusterroles system:discovery -o yaml
```

**Note:**

If you edit that ClusterRole, your changes will be overwritten on API server restart via [auto-reconciliation](#). To avoid that overwriting, either do not manually edit the role, or disable auto-reconciliation.

Default ClusterRole	Default ClusterRoleBinding	Description
<code>system:basic-user</code>	<code>system:authenticated</code> group	Allows a user read-only access to basic information about themselves. Prior to v1.14, this role was also bound to <code>system:unauthenticated</code> by default.

Default ClusterRole	Default ClusterRoleBinding	Description
<b>system:discovery</b>	<b>system:authenticated</b> group	Allows read-only access to API discovery endpoints needed to discover and negotiate an API level. Prior to v1.14, this role was also bound to <code>system:unauthenticated</code> by default.
<b>system:public-info-viewer</b>	<b>system:authenticated</b> and <b>system:unauthenticated</b> groups	Allows read-only access to non-sensitive information about the cluster. Introduced in Kubernetes v1.14.

## User-facing roles

Some of the default ClusterRoles are not `system:` prefixed. These are intended to be user-facing roles. They include super-user roles (`cluster-admin`), roles intended to be granted cluster-wide using ClusterRoleBindings, and roles intended to be granted within particular namespaces using RoleBindings (`admin`, `edit`, `view`).

User-facing ClusterRoles use [ClusterRole aggregation](#) to allow admins to include rules for custom resources on these ClusterRoles. To add rules to the `admin`, `edit`, or `view` roles, create a ClusterRole with one or more of the following labels:

```
metadata:  
  labels:  
    rbac.authorization.k8s.io/aggregate-to-admin: "true"  
    rbac.authorization.k8s.io/aggregate-to-edit: "true"  
    rbac.authorization.k8s.io/aggregate-to-view: "true"
```

Default ClusterRole	Default ClusterRoleBinding	Description
<b>cluster-admin</b>	<b>system:masters</b> group	Allows super-user access to perform any action on any resource. When used in a <b>ClusterRoleBinding</b> , it gives full control over every resource in the cluster and in all namespaces. When used in a <b>RoleBinding</b> , it gives full control over every resource in the role binding's namespace, including the namespace itself.

Default ClusterRole	Default ClusterRoleBinding	Description
<b>admin</b>	None	<p>Allows admin access, intended to be granted within a namespace using a <b>RoleBinding</b>.</p> <p>If used in a <b>RoleBinding</b>, allows read/write access to most resources in a namespace, including the ability to create roles and role bindings within the namespace. This role does not allow write access to resource quota or to the namespace itself. This role also does not allow write access to EndpointSlices (or Endpoints) in clusters created using Kubernetes v1.22+. More information is available in the <a href="#">"Write Access for EndpointSlices and Endpoints" section</a>.</p>
<b>edit</b>	None	<p>Allows read/write access to most objects in a namespace.</p> <p>This role does not allow viewing or modifying roles or role bindings.</p> <p>However, this role allows accessing Secrets and running Pods as any ServiceAccount in the namespace, so it can be used to gain the API access levels of any ServiceAccount in the namespace.</p> <p>This role also does not allow write access to EndpointSlices (or Endpoints) in clusters created using Kubernetes v1.22+. More information is available in the <a href="#">"Write Access for EndpointSlices and Endpoints" section</a>.</p>
<b>view</b>	None	<p>Allows read-only access to see most objects in a namespace. It does not allow viewing roles or role bindings.</p> <p>This role does not allow viewing Secrets, since reading the contents of Secrets enables access to ServiceAccount credentials in the namespace, which would allow API access as any ServiceAccount in the namespace (a form of privilege escalation).</p>

## Core component roles

Default ClusterRole	Default ClusterRoleBinding	Description
<b>system:kube-scheduler</b>	<b>system:kube-scheduler</b> user	Allows access to the resources required by the <u>scheduler</u> component.
<b>system:volume-scheduler</b>	<b>system:kube-scheduler</b> user	Allows access to the volume resources required by the kube-scheduler component.

Default ClusterRole	Default ClusterRoleBinding	Description
<b>system:kube-controller-manager</b>	<b>system:kube-controller-manager</b> user	Allows access to the resources required by the controller manager component. The permissions required by individual controllers are detailed in the <a href="#">controller roles</a> .
<b>system:node</b>	None	<p>Allows access to resources required by the kubelet, <b>including read access to all secrets, and write access to all pod status objects</b>.</p> <p>You should use the <a href="#">Node authorizer</a> and <a href="#">NodeRestriction admission plugin</a> instead of the system:node role, and allow granting API access to kubelets based on the Pods scheduled to run on them.</p> <p>The system:node role only exists for compatibility with Kubernetes clusters upgraded from versions prior to v1.8.</p>
<b>system:node-proxier</b>	<b>system:kube-proxy</b> user	Allows access to the resources required by the kube-proxy component.

## Other component roles

Default ClusterRole	Default ClusterRoleBinding	Description
<b>system:auth-delegator</b>	None	Allows delegated authentication and authorization checks. This is commonly used by add-on API servers for unified authentication and authorization.
<b>system:heapster</b>	None	Role for the <a href="#">Heapster</a> component (deprecated).
<b>system:kube-aggregator</b>	None	Role for the <a href="#">kube-aggregator</a> component.
<b>system:kube-dns</b>	<b>kube-dns</b> service account in the <b>kube-system</b> namespace	Role for the <a href="#">kube-dns</a> component.
<b>system:kubelet-api-admin</b>	None	Allows full access to the kubelet API.
<b>system:node-bootstrapper</b>	None	Allows access to the resources required to perform <a href="#">kubelet TLS bootstrapping</a> .
<b>system:node-problem-detector</b>	None	Role for the <a href="#">node-problem-detector</a> component.
<b>system:persistent-volume-provisioner</b>	None	Allows access to the resources required by most <a href="#">dynamic volume provisioners</a> .

Default ClusterRole	Default ClusterRoleBinding	Description
<b>system:monitoring</b>	<b>system:monitoring</b> group	Allows read access to control-plane monitoring endpoints (i.e. <code>/liveness</code> and <code>/readiness</code> endpoints ( <code>/healthz</code> , <code>/livez</code> , <code>/readyz</code> ), the individual health-check endpoints ( <code>/healthz/*</code> , <code>/livez/*</code> , <code>/readyz/*</code> ), and <code>/metrics</code> ). Note that individual health check endpoints and the metric endpoint may expose sensitive information.

## Roles for built-in controllers

The Kubernetes controller manager runs controllers that are built in to the Kubernetes control plane. When invoked with `--use-service-account-credentials`, `kube-controller-manager` starts each controller using a separate service account. Corresponding roles exist for each built-in controller, prefixed with `system:controller:`. If the controller manager is not started with `--use-service-account-credentials`, it runs all control loops using its own credential, which must be granted all the relevant roles. These roles include:

- `system:controller:attachdetach-controller`
- `system:controller:certificate-controller`
- `system:controller:clusterrole-aggregation-controller`
- `system:controller:cronjob-controller`
- `system:controller:daemon-set-controller`
- `system:controller:deployment-controller`
- `system:controller:disruption-controller`
- `system:controller:endpoint-controller`
- `system:controller:expand-controller`
- `system:controller:generic-garbage-collector`
- `system:controller:horizontal-pod-autoscaler`
- `system:controller:job-controller`
- `system:controller:namespace-controller`
- `system:controller:node-controller`
- `system:controller:persistent-volume-binder`
- `system:controller:pod-garbage-collector`
- `system:controller:pv-protection-controller`
- `system:controller:pvc-protection-controller`
- `system:controller:replicaset-controller`
- `system:controller:replication-controller`
- `system:controller:resourcequota-controller`
- `system:controller:root-ca-cert-publisher`
- `system:controller:route-controller`
- `system:controller:service-account-controller`
- `system:controller:service-controller`
- `system:controller:statefulset-controller`
- `system:controller:ttl-controller`

## Privilege escalation prevention and bootstrapping

The RBAC API prevents users from escalating privileges by editing roles or role bindings. Because this is enforced at the API level, it applies even when the RBAC authorizer is not in use.

## Restrictions on role creation or update

You can only create/update a role if at least one of the following things is true:

1. You already have all the permissions contained in the role, at the same scope as the object being modified (cluster-wide for a ClusterRole, within the same namespace or cluster-wide for a Role).
2. You are granted explicit permission to perform the `escalate` verb on the `roles` or `clusterroles` resource in the `rbac.authorization.k8s.io` API group.

For example, if `user-1` does not have the ability to list Secrets cluster-wide, they cannot create a ClusterRole containing that permission. To allow a user to create/update roles:

1. Grant them a role that allows them to create/update Role or ClusterRole objects, as desired.
2. Grant them permission to include specific permissions in the roles they create/update:
  - o implicitly, by giving them those permissions (if they attempt to create or modify a Role or ClusterRole with permissions they themselves have not been granted, the API request will be forbidden)
  - o or explicitly allow specifying any permission in a Role or ClusterRole by giving them permission to perform the `escalate` verb on `roles` or `clusterroles` resources in the `rbac.authorization.k8s.io` API group

## Restrictions on role binding creation or update

You can only create/update a role binding if you already have all the permissions contained in the referenced role (at the same scope as the role binding) *or* if you have been authorized to perform the `bind` verb on the referenced role. For example, if `user-1` does not have the ability to list Secrets cluster-wide, they cannot create a ClusterRoleBinding to a role that grants that permission. To allow a user to create/update role bindings:

1. Grant them a role that allows them to create/update RoleBinding or ClusterRoleBinding objects, as desired.
2. Grant them permissions needed to bind a particular role:
  - o implicitly, by giving them the permissions contained in the role.
  - o explicitly, by giving them permission to perform the `bind` verb on the particular Role (or ClusterRole).

For example, this ClusterRole and RoleBinding would allow `user-1` to grant other users the `admin`, `edit`, and `view` roles in the namespace `user-1-namespace`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: role-grantor
rules:
- apiGroups: ["rbac.authorization.k8s.io"]
  resources: ["rolebindings"]
  verbs: ["create"]
- apiGroups: ["rbac.authorization.k8s.io"]
  resources: ["clusterroles"]
  verbs: ["bind"]
  # omit resourceNames to allow binding any ClusterRole
  resourceNames: ["admin", "edit", "view"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: role-grantor-binding
  namespace: user-1-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: role-grantor
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: user-1
```

When bootstrapping the first roles and role bindings, it is necessary for the initial user to grant permissions they do not yet have. To bootstrap initial roles and role bindings:

- Use a credential with the "system:masters" group, which is bound to the "cluster-admin" super-user role by the default bindings.

## Command-line utilities

### kubectl create role

Creates a Role object defining permissions within a single namespace.

Examples:

- Create a Role named "pod-reader" that allows users to perform `get`, `watch` and `list` on pods:

```
kubectl create role pod-reader --verb=get --verb=list --verb=wat
```

- Create a Role named "pod-reader" with `resourceNames` specified:

```
kubectl create role pod-reader --verb=get --resource=pods --resou
```

- Create a Role named "foo" with `apiGroups` specified:

```
kubectl create role foo --verb=get,list,watch --resource=replica
```

- Create a Role named "foo" with subresource permissions:

```
kubectl create role foo --verb=get,list,watch --resource=pods,po
```

- Create a Role named "my-component-lease-holder" with permissions to get/update a resource with a specific name:

```
kubectl create role my-component-lease-holder --verb=get,list,wa
```

## kubectl create clusterrole

Creates a ClusterRole. Examples:

- Create a ClusterRole named "pod-reader" that allows user to perform `get` , `watch` and `list` on pods:

```
kubectl create clusterrole pod-reader --verb=get,list,watch --re
```

- Create a ClusterRole named "pod-reader" with `resourceNames` specified:

```
kubectl create clusterrole pod-reader --verb=get --resource=pods
```

- Create a ClusterRole named "foo" with `apiGroups` specified:

```
kubectl create clusterrole foo --verb=get,list,watch --resource=
```

- Create a ClusterRole named "foo" with subresource permissions:

```
kubectl create clusterrole foo --verb=get,list,watch --resource=
```

- Create a ClusterRole named "foo" with `nonResourceURL` specified:

```
kubectl create clusterrole "foo" --verb=get --non-resource-url=
```

- Create a ClusterRole named "monitoring" with an `aggregationRule` specified:

```
kubectl create clusterrole monitoring --aggregation-rule="rbac.e
```

## kubectl create rolebinding

Grants a Role or ClusterRole within a specific namespace. Examples:

- Within the namespace "acme", grant the permissions in the "admin" ClusterRole to a user named "bob":

```
kubectl create rolebinding bob-admin-binding --clusterrole=adm
```

- Within the namespace "acme", grant the permissions in the "view" ClusterRole to the service account in the namespace "acme" named "myapp":

```
kubectl create rolebinding myapp-view-binding --clusterrole=view
```

- Within the namespace "acme", grant the permissions in the "view" ClusterRole to a service account in the namespace "myappnamespace" named "myapp":

```
kubectl create rolebinding myappnamespace-myapp-view-binding --c
```

## kubectl create clusterrolebinding

Grants a ClusterRole across the entire cluster (all namespaces).

Examples:

- Across the entire cluster, grant the permissions in the "cluster-admin" ClusterRole to a user named "root":

```
kubectl create clusterrolebinding root-cluster-admin-binding --c
```

- Across the entire cluster, grant the permissions in the "system:node-proxier" ClusterRole to a user named "system:kube-proxy":

```
kubectl create clusterrolebinding kube-proxy-binding --clusterro
```

- Across the entire cluster, grant the permissions in the "view" ClusterRole to a service account named "myapp" in the namespace "acme":

```
kubectl create clusterrolebinding myapp-view-binding --clusterro
```

## kubectl auth reconcile

Creates or updates `rbac.authorization.k8s.io/v1` API objects from a manifest file.

Missing objects are created, and the containing namespace is created for namespaced objects, if required.

Existing roles are updated to include the permissions in the input objects, and remove extra permissions if `--remove-extra-permissions` is specified.

Existing bindings are updated to include the subjects in the input objects, and remove extra subjects if `--remove-extra-subjects` is specified.

Examples:

- Test applying a manifest file of RBAC objects, displaying changes that would be made:

```
kubectl auth reconcile -f my-rbac-rules.yaml --dry-run=client
```

- Apply a manifest file of RBAC objects, preserving any extra permissions (in roles) and any extra subjects (in bindings):

```
kubectl auth reconcile -f my-rbac-rules.yaml
```

- Apply a manifest file of RBAC objects, removing any extra permissions (in roles) and any extra subjects (in bindings):

```
kubectl auth reconcile -f my-rbac-rules.yaml --remove-extra-subj
```

## ServiceAccount permissions

Default RBAC policies grant scoped permissions to control-plane components, nodes, and controllers, but grant *no permissions* to service accounts outside the `kube-system` namespace (beyond the permissions given by [API discovery roles](#)).

This allows you to grant particular roles to particular ServiceAccounts as needed. Fine-grained role bindings provide greater security, but require more effort to administrate. Broader grants can give unnecessary (and potentially escalating) API access to ServiceAccounts, but are easier to administrate.

In order from most secure to least secure, the approaches are:

1. Grant a role to an application-specific service account (best practice)

This requires the application to specify a `serviceAccountName` in its pod spec, and for the service account to be created (via the API, application manifest, `kubectl create serviceaccount`, etc.).

For example, grant read-only permission within "my-namespace" to the "my-sa" service account:

```
kubectl create rolebinding my-sa-view \
  --clusterrole=view \
  --serviceaccount=my-namespace:my-sa \
  --namespace=my-namespace
```

2. Grant a role to the "default" service account in a namespace

If an application does not specify a `serviceAccountName`, it uses the "default" service account.

**Note:**

Permissions given to the "default" service account are available to any pod in the namespace that does not specify a `serviceAccountName`.

For example, grant read-only permission within "my-namespace" to the "default" service account:

```
kubectl create rolebinding default-view \
--clusterrole=view \
--serviceaccount=my-namespace:default \
--namespace=my-namespace
```

Many [add-ons](#) run as the "default" service account in the `kube-system` namespace. To allow those add-ons to run with super-user access, grant cluster-admin permissions to the "default" service account in the `kube-system` namespace.

**Caution:**

Enabling this means the `kube-system` namespace contains Secrets that grant super-user access to your cluster's API.

```
kubectl create clusterrolebinding add-on-cluster-admin \
--clusterrole=cluster-admin \
--serviceaccount=kube-system:default
```

### 3. Grant a role to all service accounts in a namespace

If you want all applications in a namespace to have a role, no matter what service account they use, you can grant a role to the service account group for that namespace.

For example, grant read-only permission within "my-namespace" to all service accounts in that namespace:

```
kubectl create rolebinding serviceaccounts-view \
--clusterrole=view \
--group=system:serviceaccounts:my-namespace \
--namespace=my-namespace
```

### 4. Grant a limited role to all service accounts cluster-wide (discouraged)

If you don't want to manage permissions per-namespace, you can grant a cluster-wide role to all service accounts.

For example, grant read-only permission across all namespaces to all service accounts in the cluster:

```
kubectl create clusterrolebinding serviceaccounts-view \
--clusterrole=view \
--group=system:serviceaccounts
```

### 5. Grant super-user access to all service accounts cluster-wide (strongly discouraged)

If you don't care about partitioning permissions at all, you can grant super-user access to all service accounts.

## Warning:

This allows any application full access to your cluster, and also grants any user with read access to Secrets (or the ability to create any pod) full access to your cluster.

```
kubectl create clusterrolebinding serviceaccounts-cluster-admin  
  --clusterrole=cluster-admin \  
  --group=system:serviceaccounts
```

## Write access for EndpointSlices and Endpoints

Kubernetes clusters created before Kubernetes v1.22 include write access to EndpointSlices (and Endpoints) in the aggregated "edit" and "admin" roles. As a mitigation for [CVE-2021-25740](#), this access is not part of the aggregated roles in clusters that you create using Kubernetes v1.22 or later.

Existing clusters that have been upgraded to Kubernetes v1.22 will not be subject to this change. The [CVE announcement](#) includes guidance for restricting this access in existing clusters.

If you want new clusters to retain this level of access in the aggregated roles, you can create the following ClusterRole:

[access/endpoints-aggregated.yaml](#) 

```
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRole  
metadata:  
  annotations:  
    kubernetes.io/description: |-  
      Add endpoints write permissions to the edit and admin roles. This  
      was removed by default in 1.22 because of CVE-2021-25740. See  
      https://issues.k8s.io/103675. This can allow writers to direct L  
      oad Balancer or Ingress implementations to expose backend IPs that would not  
      be accessible, and can circumvent network policies or security  
      intended to prevent/isolate access to those backends.  
      EndpointSlices were never included in the edit or admin roles,  
      so there is nothing to restore for the EndpointSlice API.  
  labels:  
    rbac.authorization.k8s.io/aggregate-to-edit: "true"  
  name: custom:aggregate-to-edit:endpoints # you can change this if you  
  rules:  
    - apiGroups: [""]  
      resources: ["endpoints"]  
      verbs: ["create", "delete", "deletecollection", "patch", "update"]
```

## Upgrading from ABAC

Clusters that originally ran older Kubernetes versions often used permissive ABAC policies, including granting full API access to all service accounts.

Default RBAC policies grant scoped permissions to control-plane components, nodes, and controllers, but grant *no permissions* to service accounts outside the `kube-system` namespace (beyond the permissions given by [API discovery roles](#)).

While far more secure, this can be disruptive to existing workloads expecting to automatically receive API permissions. Here are two approaches for managing this transition:

## Parallel authorizers

Run both the RBAC and ABAC authorizers, and specify a policy file that contains the [legacy ABAC policy](#):

```
--authorization-mode=...,RBAC,ABAC --authorization-policy-file=mypoli
```

To explain that first command line option in detail: if earlier authorizers, such as Node, deny a request, then the RBAC authorizer attempts to authorize the API request. If RBAC also denies that API request, the ABAC authorizer is then run. This means that any request allowed by *either* the RBAC or ABAC policies is allowed.

When the `kube-apiserver` is run with a log level of 5 or higher for the RBAC component ( `--vmodule=rbac*=5` or `--v=5` ), you can see RBAC denials in the API server log (prefixed with `RBAC` ). You can use that information to determine which roles need to be granted to which users, groups, or service accounts.

Once you have [granted roles to service accounts](#) and workloads are running with no RBAC denial messages in the server logs, you can remove the ABAC authorizer.

## Permissive RBAC permissions

You can replicate a permissive ABAC policy using RBAC role bindings.

### Warning:

The following policy allows **ALL** service accounts to act as cluster administrators. Any application running in a container receives service account credentials automatically, and could perform any action against the API, including viewing secrets and modifying permissions. This is not a recommended policy.

```
kubectl create clusterrolebinding permissive-binding \
  --clusterrole=cluster-admin \
  --user=admin \
  --user=kubelet \
  --group=system:serviceaccounts
```

After you have transitioned to use RBAC, you should adjust the access controls for your cluster to ensure that these meet your information security needs.

## 3.5 - Using Node Authorization

Node authorization is a special-purpose authorization mode that specifically authorizes API requests made by kubelets.

### Overview

The Node authorizer allows a kubelet to perform API operations. This includes:

Read operations:

- services
- endpoints
- nodes
- pods
- secrets, configmaps, persistent volume claims and persistent volumes related to pods bound to the kubelet's node

#### ⓘ FEATURE STATE: Kubernetes v1.31 [alpha]

When the `AuthorizeNodeWithSelectors` feature is enabled (along with the pre-requisite `AuthorizeWithSelectors` feature), kubelets are only allowed to read their own Node objects, and are only allowed to read pods bound to their node.

Write operations:

- nodes and node status (enable the `NodeRestriction` admission plugin to limit a kubelet to modify its own node)
- pods and pod status (enable the `NodeRestriction` admission plugin to limit a kubelet to modify pods bound to itself)
- events

Auth-related operations:

- read/write access to the [CertificateSigningRequests API](#) for TLS bootstrapping
- the ability to create TokenReviews and SubjectAccessReviews for delegated authentication/authorization checks

In future releases, the node authorizer may add or remove permissions to ensure kubelets have the minimal set of permissions required to operate correctly.

In order to be authorized by the Node authorizer, kubelets must use a credential that identifies them as being in the `system:nodes` group, with a username of `system:node:<nodeName>`. This group and user name format match the identity created for each kubelet as part of [kubelet TLS bootstrapping](#).

The value of `<nodeName>` **must** match precisely the name of the node as registered by the kubelet. By default, this is the host name as provided by `hostname`, or overridden via the [kubelet option](#) `--hostname-override`. However, when using the `--cloud-provider` kubelet option, the specific hostname may be determined by the cloud provider, ignoring the local `hostname` and the `--hostname-override` option. For specifics about how the kubelet determines the hostname, see the [kubelet options reference](#).

To enable the Node authorizer, start the apiserver with `--authorization-mode=Node` .

To limit the API objects kubelets are able to write, enable the [NodeRestriction](#) admission plugin by starting the apiserver with `--enable-admission-plugins=...,NodeRestriction,...`

## Migration considerations

### Kubelets outside the `system:nodes` group

Kubelets outside the `system:nodes` group would not be authorized by the `Node` authorization mode, and would need to continue to be authorized via whatever mechanism currently authorizes them. The node admission plugin would not restrict requests from these kubelets.

### Kubelets with undifferentiated usernames

In some deployments, kubelets have credentials that place them in the `system:nodes` group, but do not identify the particular node they are associated with, because they do not have a username in the `system:node:...` format. These kubelets would not be authorized by the `Node` authorization mode, and would need to continue to be authorized via whatever mechanism currently authorizes them.

The `NodeRestriction` admission plugin would ignore requests from these kubelets, since the default node identifier implementation would not consider that a node identity.

## 3.6 - Webhook Mode

A WebHook is an HTTP callback: an HTTP POST that occurs when something happens; a simple event-notification via HTTP POST. A web application implementing WebHooks will POST a message to a URL when certain things happen.

When specified, mode `Webhook` causes Kubernetes to query an outside REST service when determining user privileges.

## Configuration File Format

Mode `Webhook` requires a file for HTTP configuration, specify by the `--authorization-webhook-config-file=SOME_FILENAME` flag.

The configuration file uses the [kubeconfig](#) file format. Within the file "users" refers to the API Server webhook and "clusters" refers to the remote service.

A configuration example which uses HTTPS client auth:

```
# Kubernetes API version
apiVersion: v1
# kind of the API object
kind: Config
# clusters refers to the remote service.
clusters:
  - name: name-of-remote-authz-service
    cluster:
      # CA for verifying the remote service.
      certificate-authority: /path/to/ca.pem
      # URL of remote service to query. Must use 'https'. May not include port.
      server: https://authz.example.com/authorize

# users refers to the API Server's webhook configuration.
users:
  - name: name-of-api-server
    user:
      client-certificate: /path/to/cert.pem # cert for the webhook plugin
      client-key: /path/to/key.pem           # key matching the cert

# kubeconfig files require a context. Provide one for the API Server.
current-context: webhook
contexts:
  - context:
      cluster: name-of-remote-authz-service
      user: name-of-api-server
      name: webhook
```

## Request Payloads

When faced with an authorization decision, the API Server POSTs a JSON-serialized `authorization.k8s.io/v1beta1 SubjectAccessReview` object describing the action. This object contains fields describing the user attempting to make the request, and either details about the resource being accessed or requests attributes.

Note that webhook API objects are subject to the same [versioning compatibility rules](#) as other Kubernetes API objects. Implementers should be aware of looser compatibility promises for beta objects and check the

"apiVersion" field of the request to ensure correct deserialization. Additionally, the API Server must enable the `authorization.k8s.io/v1beta1` API extensions group ( `--runtime-config=authorization.k8s.io/v1beta1=true` ).

An example request body:

```
{  
  "apiVersion": "authorization.k8s.io/v1beta1",  
  "kind": "SubjectAccessReview",  
  "spec": {  
    "resourceAttributes": {  
      "namespace": "kittensandponies",  
      "verb": "get",  
      "group": "unicorn.example.org",  
      "resource": "pods"  
    },  
    "user": "jane",  
    "group": [  
      "group1",  
      "group2"  
    ]  
  }  
}
```

The remote service is expected to fill the `status` field of the request and respond to either allow or disallow access. The response body's `spec` field is ignored and may be omitted. A permissive response would return:

```
{  
  "apiVersion": "authorization.k8s.io/v1beta1",  
  "kind": "SubjectAccessReview",  
  "status": {  
    "allowed": true  
  }  
}
```

For disallowing access there are two methods.

The first method is preferred in most cases, and indicates the authorization webhook does not allow, or has "no opinion" about the request, but if other authorizers are configured, they are given a chance to allow the request. If there are no other authorizers, or none of them allow the request, the request is forbidden. The webhook would return:

```
{  
  "apiVersion": "authorization.k8s.io/v1beta1",  
  "kind": "SubjectAccessReview",  
  "status": {  
    "allowed": false,  
    "reason": "user does not have read access to the namespace"  
  }  
}
```

The second method denies immediately, short-circuiting evaluation by other configured authorizers. This should only be used by webhooks that have detailed knowledge of the full authorizer configuration of the cluster. The webhook would return:

```
{  
  "apiVersion": "authorization.k8s.io/v1beta1",  
  "kind": "SubjectAccessReview",  
  "status": {  
    "allowed": false,  
    "denied": true,  
    "reason": "user does not have read access to the namespace"  
  }  
}
```

Access to non-resource paths are sent as:

```
{  
  "apiVersion": "authorization.k8s.io/v1beta1",  
  "kind": "SubjectAccessReview",  
  "spec": {  
    "nonResourceAttributes": {  
      "path": "/debug",  
      "verb": "get"  
    },  
    "user": "jane",  
    "group": [  
      "group1",  
      "group2"  
    ]  
  }  
}
```

## ⓘ FEATURE STATE: Kubernetes v1.31 [alpha]

With the `AuthorizeWithSelectors` feature enabled, field and label selectors in the request are passed to the authorization webhook. The webhook can make authorization decisions informed by the scoped field and label selectors, if it wishes.

The [SubjectAccessReview API documentation](#) gives guidelines for how these fields should be interpreted and handled by authorization webhooks, specifically using the parsed requirements rather than the raw selector strings, and how to handle unrecognized operators safely.

```
{  
  "apiVersion": "authorization.k8s.io/v1beta1",  
  "kind": "SubjectAccessReview",  
  "spec": {  
    "resourceAttributes": {  
      "verb": "list",  
      "group": "",  
      "resource": "pods",  
      "fieldSelector": {  
        "requirements": [  
          {"key": "spec.nodeName", "operator": "In", "values": ["mynode"]}],  
      },  
      "labelSelector": {  
        "requirements": [  
          {"key": "example.com/mykey", "operator": "In", "values": ["myvalue"]}],  
      },  
      "user": "jane",  
      "group": [  
        "group1",  
        "group2"  
      ]  
    }  
  }  
}
```

Non-resource paths include: `/api` , `/apis` , `/metrics` , `/logs` , `/debug` , `/healthz` , `/livez` , `/openapi/v2` , `/readyz` , and `/version`. Clients require access to `/api` , `/api/*` , `/apis` , `/apis/*` , and `/version` to discover what resources and versions are present on the server. Access to other non-resource paths can be disallowed without restricting access to the REST api.

For further information, refer to the [SubjectAccessReview API documentation](#) and [webhook.go implementation](#).

## 3.7 - Using ABAC Authorization

Attribute-based access control (ABAC) defines an access control paradigm whereby access rights are granted to users through the use of policies which combine attributes together.

### Policy File Format

To enable ABAC mode, specify `--authorization-policy-file=SOME_FILENAME` and `--authorization-mode=ABAC` on startup.

The file format is [one JSON object per line](#). There should be no enclosing list or map, only one map per line.

Each line is a "policy object", where each such object is a map with the following properties:

- Versioning properties:
  - `apiVersion`, type string; valid values are "abac.authorization.kubernetes.io/v1beta1". Allows versioning and conversion of the policy format.
  - `kind`, type string; valid values are "Policy". Allows versioning and conversion of the policy format.
- `spec` property set to a map with the following properties:
  - Subject-matching properties:
    - `user`, type string; the user-string from `--token-auth-file`. If you specify `user`, it must match the username of the authenticated user.
    - `group`, type string; if you specify `group`, it must match one of the groups of the authenticated user.  
`system:authenticated` matches all authenticated requests. `system:unauthenticated` matches all unauthenticated requests.
  - Resource-matching properties:
    - `apiGroup`, type string; an API group.
      - Ex: `apps`, `networking.k8s.io`
      - Wildcard: `*` matches all API groups.
    - `namespace`, type string; a namespace.
      - Ex: `kube-system`
      - Wildcard: `*` matches all resource requests.
    - `resource`, type string; a resource type
      - Ex: `pods`, `deployments`
      - Wildcard: `*` matches all resource requests.
  - Non-resource-matching properties:
    - `nonResourcePath`, type string; non-resource request paths.
      - Ex: `/version` or `/apis`
      - Wildcard:
        - `*` matches all non-resource requests.
        - `/foo/*` matches all subpaths of `/foo/`.

#### Note:

An unset property is the same as a property set to the zero value for

its type (e.g. empty string, 0, false). However, unset should be preferred for readability.

In the future, policies may be expressed in a JSON format, and managed via a REST interface.

## Authorization Algorithm

A request has attributes which correspond to the properties of a policy object.

When a request is received, the attributes are determined. Unknown attributes are set to the zero value of its type (e.g. empty string, 0, false).

A property set to `"*"` will match any value of the corresponding attribute.

The tuple of attributes is checked for a match against every policy in the policy file. If at least one line matches the request attributes, then the request is authorized (but may fail later validation).

To permit any authenticated user to do something, write a policy with the `group` property set to `"system:authenticated"`.

To permit any unauthenticated user to do something, write a policy with the `group` property set to `"system:unauthenticated"`.

To permit a user to do anything, write a policy with the `apiGroup`, `namespace`, `resource`, and `nonResourcePath` properties set to `"*"`.

## Kubectl

Kubectl uses the `/api` and `/apis` endpoints of apiserver to discover served resource types, and validates objects sent to the API by create/update operations using schema information located at `/openapi/v2`.

When using ABAC authorization, those special resources have to be explicitly exposed via the `nonResourcePath` property in a policy (see [examples](#) below):

- `/api`, `/api/*`, `/apis`, and `/apis/*` for API version negotiation.
- `/version` for retrieving the server version via `kubectl version`.
- `/swaggerapi/*` for create/update operations.

To inspect the HTTP calls involved in a specific kubectl operation you can turn up the verbosity:

```
kubectl --v=8 version
```

## Examples

1. Alice can do anything to all resources:

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "metadata": {"name": "allow-everything", "namespace": "system"}, "spec": {"subjects": [{"name": "alice", "groups": ["system:authenticated"]}], "resources": [{"verb": "get", "resource": "*", "apiGroup": "*"}]}
```

2. The kubelet can read any pods:

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "metadata": {"name": "allow-all"}, "spec": {"allow": true, "subjects": [{"subject": "system:serviceaccount:default"}], "resources": [{"resource": "events", "verbs": ["get", "list", "watch"]}], "nonResourceURLs": ["*"]}}
```

3. The kubelet can read and write events:

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "metadata": {"name": "allow-kubelet"}, "spec": {"allow": true, "subjects": [{"subject": "system:kubelet"}], "resources": [{"resource": "events", "verbs": ["get", "list", "watch"]}], "nonResourceURLs": ["*"]}}
```

4. Bob can just read pods in namespace "projectCaribou":

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "metadata": {"name": "allow-bob"}, "spec": {"allow": true, "subjects": [{"subject": "system:serviceaccount:projectCaribou:bob"}], "resources": [{"resource": "pods", "verbs": ["get"]}], "nonResourceURLs": ["*"]}}
```

5. Anyone can make read-only requests to all non-resource paths:

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "metadata": {"name": "allow-all"}, "spec": {"allow": true, "subjects": [{"subject": "system:serviceaccount:default"}], "resources": [{"resource": "events", "verbs": ["get", "list", "watch"]}], "nonResourceURLs": ["*"]}}
```

[Complete file example](#)

## A quick note on service accounts

Every service account has a corresponding ABAC username, and that service account's username is generated according to the naming convention:

```
system:serviceaccount:<namespace>:<serviceaccountname>
```

Creating a new namespace leads to the creation of a new service account in the following format:

```
system:serviceaccount:<namespace>:default
```

For example, if you wanted to grant the default service account (in the `kube-system` namespace) full privilege to the API using ABAC, you would add this line to your policy file:

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "metadata": {"name": "allow-all"}, "spec": {"allow": true, "subjects": [{"subject": "system:serviceaccount:kube-system:default"}], "resources": [{"resource": "events", "verbs": ["get", "list", "watch"]}], "nonResourceURLs": ["*"]}}
```

The apiserver will need to be restarted to pick up the new policy lines.

# 3.8 - Admission Controllers Reference

This page provides an overview of Admission Controllers.

## What are they?

An *admission controller* is a piece of code that intercepts requests to the Kubernetes API server prior to persistence of the object, but after the request is authenticated and authorized.

Admission controllers may be *validating*, *mutating*, or both. Mutating controllers may modify objects related to the requests they admit; validating controllers may not.

Admission controllers limit requests to create, delete, modify objects. Admission controllers can also block custom verbs, such as a request connect to a Pod via an API server proxy. Admission controllers do *not* (and cannot) block requests to read (**get**, **watch** or **list**) objects.

The admission controllers in Kubernetes 1.31 consist of the [list](#) below, are compiled into the `kube-apiserver` binary, and may only be configured by the cluster administrator. In that list, there are two special controllers: `MutatingAdmissionWebhook` and `ValidatingAdmissionWebhook`. These execute the mutating and validating (respectively) [admission control webhooks](#) which are configured in the API.

## Admission control phases

The admission control process proceeds in two phases. In the first phase, mutating admission controllers are run. In the second phase, validating admission controllers are run. Note again that some of the controllers are both.

If any of the controllers in either phase reject the request, the entire request is rejected immediately and an error is returned to the end-user.

Finally, in addition to sometimes mutating the object in question, admission controllers may sometimes have side effects, that is, mutate related resources as part of request processing. Incrementing quota usage is the canonical example of why this is necessary. Any such side-effect needs a corresponding reclamation or reconciliation process, as a given admission controller does not know for sure that a given request will pass all of the other admission controllers.

## Why do I need them?

Several important features of Kubernetes require an admission controller to be enabled in order to properly support the feature. As a result, a Kubernetes API server that is not properly configured with the right set of admission controllers is an incomplete server and will not support all the features you expect.

## How do I turn on an admission controller?

The Kubernetes API server flag `enable-admission-plugins` takes a comma-delimited list of admission control plugins to invoke prior to modifying objects in the cluster. For example, the following command line enables the `NamespaceLifecycle` and the `LimitRanger` admission control plugins:

```
kube-apiserver --enable-admission-plugins=NamespaceLifecycle,LimitRan
```

**Note:**

Depending on the way your Kubernetes cluster is deployed and how the API server is started, you may need to apply the settings in different ways. For example, you may have to modify the systemd unit file if the API server is deployed as a systemd service, you may modify the manifest file for the API server if Kubernetes is deployed in a self-hosted way.

## How do I turn off an admission controller?

The Kubernetes API server flag `disable-admission-plugins` takes a comma-delimited list of admission control plugins to be disabled, even if they are in the list of plugins enabled by default.

```
kube-apiserver --disable-admission-plugins=PodNodeSelector,AlwaysDeny
```

## Which plugins are enabled by default?

To see which admission plugins are enabled:

```
kube-apiserver -h | grep enable-admission-plugins
```

In Kubernetes 1.31, the default ones are:

```
CertificateApproval, CertificateSigning, CertificateSubjectRestriction
```

**Note:**

The [ValidatingAdmissionPolicy](#) admission plugin is enabled by default, but is only active if you enable the [ValidatingAdmissionPolicy feature gate](#) and the [admissionregistration.k8s.io/v1alpha1](#) API.

## What does each admission controller do?

AlwaysAdmit

## ⓘ FEATURE STATE: Kubernetes v1.13 [deprecated]

**Type:** Validating.

This admission controller allows all pods into the cluster. It is **deprecated** because its behavior is the same as if there were no admission controller at all.

## AlwaysDeny

## ⓘ FEATURE STATE: Kubernetes v1.13 [deprecated]

**Type:** Validating.

Rejects all requests. AlwaysDeny is **deprecated** as it has no real meaning.

## AlwaysPullImages

**Type:** Mutating and Validating.

This admission controller modifies every new Pod to force the image pull policy to `Always`. This is useful in a multitenant cluster so that users can be assured that their private images can only be used by those who have the credentials to pull them. Without this admission controller, once an image has been pulled to a node, any pod from any user can use it by knowing the image's name (assuming the Pod is scheduled onto the right node), without any authorization check against the image. When this admission controller is enabled, images are always pulled prior to starting containers, which means valid credentials are required.

## CertificateApproval

**Type:** Validating.

This admission controller observes requests to approve CertificateSigningRequest resources and performs additional authorization checks to ensure the approving user has permission to **approve** certificate requests with the `spec.signerName` requested on the CertificateSigningRequest resource.

See [Certificate Signing Requests](#) for more information on the permissions required to perform different actions on CertificateSigningRequest resources.

## CertificateSigning

**Type:** Validating.

This admission controller observes updates to the `status.certificate` field of CertificateSigningRequest resources and performs an additional authorization checks to ensure the signing user has permission to **sign** certificate requests with the `spec.signerName` requested on the CertificateSigningRequest resource.

See [Certificate Signing Requests](#) for more information on the permissions required to perform different actions on CertificateSigningRequest resources.

## CertificateSubjectRestriction

**Type:** Validating.

This admission controller observes creation of `CertificateSigningRequest` resources that have a `spec.signerName` of `kubernetes.io/kube-apiserver-client`. It rejects any request that specifies a 'group' (or 'organization attribute') of `system:masters`.

## DefaultIngressClass

**Type:** Mutating.

This admission controller observes creation of `Ingress` objects that do not request any specific ingress class and automatically adds a default ingress class to them. This way, users that do not request any special ingress class do not need to care about them at all and they will get the default one.

This admission controller does not do anything when no default ingress class is configured. When more than one ingress class is marked as default, it rejects any creation of `Ingress` with an error and an administrator must revisit their `IngressClass` objects and mark only one as default (with the annotation "ingressclass.kubernetes.io/is-default-class"). This admission controller ignores any `Ingress` updates; it acts only on creation.

See the [Ingress](#) documentation for more about ingress classes and how to mark one as default.

## DefaultStorageClass

**Type:** Mutating.

This admission controller observes creation of `PersistentVolumeClaim` objects that do not request any specific storage class and automatically adds a default storage class to them. This way, users that do not request any special storage class do not need to care about them at all and they will get the default one.

This admission controller does not do anything when no default storage class is configured. When more than one storage class is marked as default, it rejects any creation of `PersistentVolumeClaim` with an error and an administrator must revisit their `StorageClass` objects and mark only one as default. This admission controller ignores any `PersistentVolumeClaim` updates; it acts only on creation.

See [persistent volume](#) documentation about persistent volume claims and storage classes and how to mark a storage class as default.

## DefaultTolerationSeconds

**Type:** Mutating.

This admission controller sets the default forgiveness toleration for pods to tolerate the taints `notready:NoExecute` and `unreachable:NoExecute` based on the k8s-apiserver input parameters `default-not-ready-toleration-seconds` and `default-unreachable-toleration-seconds` if the pods don't already have toleration for taints `node.kubernetes.io/not-ready:NoExecute` OR `node.kubernetes.io/unreachable:NoExecute`. The default value for `default-not-ready-toleration-seconds` and `default-unreachable-toleration-seconds` is 5 minutes.

## DenyServiceExternalIPs

**Type:** Validating.

This admission controller rejects all net-new usage of the `Service` field `externalIPs`. This feature is very powerful (allows network traffic interception) and not well controlled by policy. When enabled, users of the cluster may not create new Services which use `externalIPs` and may not add new values to `externalIPs` on existing `Service` objects. Existing uses of `externalIPs` are not affected, and users may remove values from `externalIPs` on existing `Service` objects.

Most users do not need this feature at all, and cluster admins should consider disabling it. Clusters that do need to use this feature should consider using some custom policy to manage usage of it.

This admission controller is disabled by default.

## EventRateLimit

ⓘ **FEATURE STATE:** Kubernetes v1.13 [alpha]

**Type:** Validating.

This admission controller mitigates the problem where the API server gets flooded by requests to store new Events. The cluster admin can specify event rate limits by:

- Enabling the `EventRateLimit` admission controller;
- Referencing an `EventRateLimit` configuration file from the file provided to the API server's command line flag `--admission-control-config-file` :

```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
  - name: EventRateLimit
    path: eventconfig.yaml
...
```

There are four types of limits that can be specified in the configuration:

- `Server` : All Event requests (creation or modifications) received by the API server share a single bucket.
- `Namespace` : Each namespace has a dedicated bucket.
- `User` : Each user is allocated a bucket.
- `SourceAndObject` : A bucket is assigned by each combination of source and involved object of the event.

Below is a sample `eventconfig.yaml` for such a configuration:

```
apiVersion: eventratelimit.admission.k8s.io/v1alpha1
kind: Configuration
limits:
  - type: Namespace
    qps: 50
    burst: 100
    cacheSize: 2000
  - type: User
    qps: 10
    burst: 50
```

See the [EventRateLimit Config API \(v1alpha1\)](#) for more details.

This admission controller is disabled by default.

## ExtendedResourceToleration

**Type:** Mutating.

This plug-in facilitates creation of dedicated nodes with extended resources. If operators want to create dedicated nodes with extended resources (like GPUs, FPGAs etc.), they are expected to [taint the node](#) with the extended resource name as the key. This admission controller, if enabled, automatically adds tolerations for such taints to pods requesting extended resources, so users don't have to manually add these tolerations.

This admission controller is disabled by default.

## ImagePolicyWebhook

**Type:** Validating.

The ImagePolicyWebhook admission controller allows a backend webhook to make admission decisions.

This admission controller is disabled by default.

### Configuration file format

ImagePolicyWebhook uses a configuration file to set options for the behavior of the backend. This file may be json or yaml and has the following format:

```
imagePolicy:  
  kubeConfigFile: /path/to/kubeconfig/for/backend  
  # time in s to cache approval  
  allowTTL: 50  
  # time in s to cache denial  
  denyTTL: 50  
  # time in ms to wait between retries  
  retryBackoff: 500  
  # determines behavior if the webhook backend fails  
  defaultAllow: true
```

Reference the ImagePolicyWebhook configuration file from the file provided to the API server's command line flag `--admission-control-config-file`:

```
apiVersion: apiserver.config.k8s.io/v1  
kind: AdmissionConfiguration  
plugins:  
  - name: ImagePolicyWebhook  
    path: imagepolicyconfig.yaml  
...
```

Alternatively, you can embed the configuration directly in the file:

```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
  - name: ImagePolicyWebhook
    configuration:
      imagePolicy:
        kubeConfigFile: <path-to-kubeconfig-file>
        allowTTL: 50
        denyTTL: 50
        retryBackoff: 500
        defaultAllow: true
```

The ImagePolicyWebhook config file must reference a [kubeconfig](#) formatted file which sets up the connection to the backend. It is required that the backend communicate over TLS.

The kubeconfig file's `cluster` field must point to the remote service, and the `user` field must contain the returned authorizer.

```
# clusters refers to the remote service.
clusters:
  - name: name-of-remote-imagepolicy-service
    cluster:
      certificate-authority: /path/to/ca.pem    # CA for verifying the
      server: https://images.example.com/policy # URL of remote service

# users refers to the API server's webhook configuration.
users:
  - name: name-of-api-server
    user:
      client-certificate: /path/to/cert.pem # cert for the webhook authorizer
      client-key: /path/to/key.pem          # key matching the cert
```

For additional HTTP configuration, refer to the [kubeconfig](#) documentation.

## Request payloads

When faced with an admission decision, the API Server POSTs a JSON serialized `imagepolicy.k8s.io/v1alpha1 ImageReview` object describing the action. This object contains fields describing the containers being admitted, as well as any pod annotations that match `*.imagepolicy.k8s.io/*`.

### Note:

The webhook API objects are subject to the same versioning compatibility rules as other Kubernetes API objects. Implementers should be aware of looser compatibility promises for alpha objects and check the `apiVersion` field of the request to ensure correct deserialization. Additionally, the API Server must enable the `imagepolicy.k8s.io/v1alpha1` API extensions group (`--runtime-config=imagepolicy.k8s.io/v1alpha1=true`).

An example request body:

```
{  
  "apiVersion": "imagepolicy.k8s.io/v1alpha1",  
  "kind": "ImageReview",  
  "spec": {  
    "containers": [  
      {  
        "image": "myrepo/myimage:v1"  
      },  
      {  
        "image": "myrepo/myimage@sha256:beb6bd6a68f114c1dc2ea4b28db81"  
      }  
    ],  
    "annotations": {  
      "mycluster.image-policy.k8s.io/ticket-1234": "break-glass"  
    },  
    "namespace": "mynamespace"  
  }  
}
```

The remote service is expected to fill the `status` field of the request and respond to either allow or disallow access. The response body's `spec` field is ignored, and may be omitted. A permissive response would return:

```
{  
  "apiVersion": "imagepolicy.k8s.io/v1alpha1",  
  "kind": "ImageReview",  
  "status": {  
    "allowed": true  
  }  
}
```

To disallow access, the service would return:

```
{  
  "apiVersion": "imagepolicy.k8s.io/v1alpha1",  
  "kind": "ImageReview",  
  "status": {  
    "allowed": false,  
    "reason": "image currently blacklisted"  
  }  
}
```

For further documentation refer to the [imagepolicy.v1alpha1 API](#).

## Extending with Annotations

All annotations on a Pod that match `*.image-policy.k8s.io/*` are sent to the webhook. Sending annotations allows users who are aware of the image policy backend to send extra information to it, and for different backends implementations to accept different information.

Examples of information you might put here are:

- request to "break glass" to override a policy, in case of emergency.
- a ticket number from a ticket system that documents the break-glass request
- provide a hint to the policy server as to the imageID of the image being provided, to save it a lookup

In any case, the annotations are provided by the user and are not validated by Kubernetes in any way.

## LimitPodHardAntiAffinityTopology

**Type:** Validating.

This admission controller denies any pod that defines `AntiAffinity` topology key other than `kubernetes.io/hostname` in `requiredDuringSchedulingRequiredDuringExecution`.

This admission controller is disabled by default.

## LimitRanger

**Type:** Mutating and Validating.

This admission controller will observe the incoming request and ensure that it does not violate any of the constraints enumerated in the `LimitRange` object in a `Namespace`. If you are using `LimitRange` objects in your Kubernetes deployment, you MUST use this admission controller to enforce those constraints. LimitRanger can also be used to apply default resource requests to Pods that don't specify any; currently, the default LimitRanger applies a 0.1 CPU requirement to all Pods in the `default` namespace.

See the [LimitRange API reference](#) and the [example of LimitRange](#) for more details.

## MutatingAdmissionWebhook

**Type:** Mutating.

This admission controller calls any mutating webhooks which match the request. Matching webhooks are called in serial; each one may modify the object if it desires.

This admission controller (as implied by the name) only runs in the mutating phase.

If a webhook called by this has side effects (for example, decrementing quota) it *must* have a reconciliation system, as it is not guaranteed that subsequent webhooks or validating admission controllers will permit the request to finish.

If you disable the `MutatingAdmissionWebhook`, you must also disable the `MutatingWebhookConfiguration` object in the `admissionregistration.k8s.io/v1` group/version via the `--runtime-config` flag, both are on by default.

## Use caution when authoring and installing mutating webhooks

- Users may be confused when the objects they try to create are different from what they get back.
- Built in control loops may break when the objects they try to create are different when read back.
  - Setting originally unset fields is less likely to cause problems than overwriting fields set in the original request. Avoid doing the latter.
- Future changes to control loops for built-in resources or third-party resources may break webhooks that work well today. Even when the webhook installation API is finalized, not all possible webhook behaviors will be guaranteed to be supported indefinitely.

## NamespaceAutoProvision

**Type:** Mutating.

This admission controller examines all incoming requests on namespaced resources and checks if the referenced namespace does exist. It creates a namespace if it cannot be found. This admission controller is useful in deployments that do not want to restrict creation of a namespace prior to its usage.

## NamespaceExists

**Type:** Validating.

This admission controller checks all requests on namespaced resources other than `Namespace` itself. If the namespace referenced from a request doesn't exist, the request is rejected.

## NamespaceLifecycle

**Type:** Validating.

This admission controller enforces that a `Namespace` that is undergoing termination cannot have new objects created in it, and ensures that requests in a non-existent `Namespace` are rejected. This admission controller also prevents deletion of three system reserved namespaces `default` , `kube-system` , `kube-public` .

A `Namespace` deletion kicks off a sequence of operations that remove all objects (pods, services, etc.) in that namespace. In order to enforce integrity of that process, we strongly recommend running this admission controller.

## NodeRestriction

**Type:** Validating.

This admission controller limits the `Node` and `Pod` objects a kubelet can modify. In order to be limited by this admission controller, kubelets must use credentials in the `system:nodes` group, with a username in the form `system:node:<nodeName>` . Such kubelets will only be allowed to modify their own `Node` API object, and only modify `Pod` API objects that are bound to their node. kubelets are not allowed to update or remove taints from their `Node` API object.

The `NodeRestriction` admission plugin prevents kubelets from deleting their `Node` API object, and enforces kubelet modification of labels under the `kubernetes.io/` or `k8s.io/` prefixes as follows:

- **Prevents** kubelets from adding/removing/updating labels with a `node-restriction.kubernetes.io/` prefix. This label prefix is reserved for administrators to label their `Node` objects for workload isolation purposes, and kubelets will not be allowed to modify labels with that prefix.
- **Allows** kubelets to add/remove/update these labels and label prefixes:
  - `kubernetes.io/hostname`
  - `kubernetes.io/arch`
  - `kubernetes.io/os`
  - `beta.kubernetes.io/instance-type`
  - `node.kubernetes.io/instance-type`
  - `failure-domain.beta.kubernetes.io/region` (deprecated)

- `failure-domain.beta.kubernetes.io/zone` (deprecated)
- `topology.kubernetes.io/region`
- `topology.kubernetes.io/zone`
- `kubelet.kubernetes.io/` -prefixed labels
- `node.kubernetes.io/` -prefixed labels

Use of any other labels under the `kubernetes.io` or `k8s.io` prefixes by kubelets is reserved, and may be disallowed or allowed by the `NodeRestriction` admission plugin in the future.

Future versions may add additional restrictions to ensure kubelets have the minimal set of permissions required to operate correctly.

## OwnerReferencesPermissionEnforcement

**Type:** Validating.

This admission controller protects the access to the `metadata.ownerReferences` of an object so that only users with **delete** permission to the object can change it. This admission controller also protects the access to `metadata.ownerReferences[x].blockOwnerDeletion` of an object, so that only users with **update** permission to the `finalizers` subresource of the referenced `owner` can change it.

## PersistentVolumeClaimResize

ⓘ **FEATURE STATE:** Kubernetes v1.24 [stable]

**Type:** Validating.

This admission controller implements additional validations for checking incoming `PersistentVolumeClaim` resize requests.

Enabling the `PersistentVolumeClaimResize` admission controller is recommended. This admission controller prevents resizing of all claims by default unless a claim's `StorageClass` explicitly enables resizing by setting `allowVolumeExpansion` to `true`.

For example: all `PersistentVolumeClaim`s created from the following `StorageClass` support volume expansion:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gluster-vol-default
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://192.168.10.100:8080"
  restuser: ""
  secretNamespace: ""
  secretName: ""
allowVolumeExpansion: true
```

For more information about persistent volume claims, see [PersistentVolumeClaims](#).

## PodNodeSelector

## ⓘ FEATURE STATE: Kubernetes v1.5 [alpha]

**Type:** Validating.

This admission controller defaults and limits what node selectors may be used within a namespace by reading a namespace annotation and a global configuration.

This admission controller is disabled by default.

### Configuration file format

`PodNodeSelector` uses a configuration file to set options for the behavior of the backend. Note that the configuration file format will move to a versioned file in a future release. This file may be json or yaml and has the following format:

```
podNodeSelectorPluginConfig:  
  clusterDefaultNodeSelector: name-of-node-selector  
  namespace1: name-of-node-selector  
  namespace2: name-of-node-selector
```

Reference the `PodNodeSelector` configuration file from the file provided to the API server's command line flag `--admission-control-config-file` :

```
apiVersion: apiserver.config.k8s.io/v1  
kind: AdmissionConfiguration  
plugins:  
- name: PodNodeSelector  
  path: podnodedeleteor.yaml  
...
```

### Configuration Annotation Format

`PodNodeSelector` uses the annotation key `scheduler.alpha.kubernetes.io/node-selector` to assign node selectors to namespaces.

```
apiVersion: v1  
kind: Namespace  
metadata:  
  annotations:  
    scheduler.alpha.kubernetes.io/node-selector: name-of-node-selecto  
  name: namespace3
```

### Internal Behavior

This admission controller has the following behavior:

1. If the `Namespace` has an annotation with a key `scheduler.alpha.kubernetes.io/node-selector`, use its value as the node selector.
2. If the namespace lacks such an annotation, use the `clusterDefaultNodeSelector` defined in the `PodNodeSelector` plugin configuration file as the node selector.
3. Evaluate the pod's node selector against the namespace node selector for conflicts. Conflicts result in rejection.

4. Evaluate the pod's node selector against the namespace-specific allowed selector defined the plugin configuration file. Conflicts result in rejection.

**Note:**

PodNodeSelector allows forcing pods to run on specifically labeled nodes. Also see the PodTolerationRestriction admission plugin, which allows preventing pods from running on specifically tainted nodes.

## PodSecurity

 **ⓘ FEATURE STATE: Kubernetes v1.25 [stable]**

**Type:** Validating.

The PodSecurity admission controller checks new Pods before they are admitted, determines if it should be admitted based on the requested security context and the restrictions on permitted [Pod Security Standards](#) for the namespace that the Pod would be in.

See the [Pod Security Admission](#) documentation for more information.

PodSecurity replaced an older admission controller named PodSecurityPolicy.

## PodTolerationRestriction

 **ⓘ FEATURE STATE: Kubernetes v1.7 [alpha]**

**Type:** Mutating and Validating.

The PodTolerationRestriction admission controller verifies any conflict between tolerations of a pod and the tolerations of its namespace. It rejects the pod request if there is a conflict. It then merges the tolerations annotated on the namespace into the tolerations of the pod. The resulting tolerations are checked against a list of allowed tolerations annotated to the namespace. If the check succeeds, the pod request is admitted otherwise it is rejected.

If the namespace of the pod does not have any associated default tolerations or allowed tolerations annotated, the cluster-level default tolerations or cluster-level list of allowed tolerations are used instead if they are specified.

Tolerations to a namespace are assigned via the `scheduler.alpha.kubernetes.io/defaultTolerations` annotation key. The list of allowed tolerations can be added via the `scheduler.alpha.kubernetes.io/tolerationsWhitelist` annotation key.

Example for namespace annotations:

```
apiVersion: v1
kind: Namespace
metadata:
  name: apps-that-need-nodes-exclusively
  annotations:
    scheduler.alpha.kubernetes.io/defaultTolerations: '[{"operator": "Exists", "key": "node.kubernetes.io/allow-node-affinity"}]'
    scheduler.alpha.kubernetes.io/tolerationsWhitelist: '[{"operator": "Exists", "key": "node.kubernetes.io/allow-node-affinity"}]'
```

This admission controller is disabled by default.

## Priority

**Type:** Mutating and Validating.

The priority admission controller uses the `priorityClassName` field and populates the integer value of the priority. If the priority class is not found, the Pod is rejected.

## ResourceQuota

**Type:** Validating.

This admission controller will observe the incoming request and ensure that it does not violate any of the constraints enumerated in the `ResourceQuota` object in a `Namespace`. If you are using `ResourceQuota` objects in your Kubernetes deployment, you MUST use this admission controller to enforce quota constraints.

See the [ResourceQuota API reference](#) and the [example of Resource Quota](#) for more details.

## RuntimeClass

**Type:** Mutating and Validating.

If you define a `RuntimeClass` with [Pod overhead](#) configured, this admission controller checks incoming Pods. When enabled, this admission controller rejects any Pod create requests that have the overhead already set. For Pods that have a `RuntimeClass` configured and selected in their `.spec`, this admission controller sets `.spec.overhead` in the Pod based on the value defined in the corresponding `RuntimeClass`.

See also [Pod Overhead](#) for more information.

## ServiceAccount

**Type:** Mutating and Validating.

This admission controller implements automation for [serviceAccounts](#). The Kubernetes project strongly recommends enabling this admission controller. You should enable this admission controller if you intend to make any use of Kubernetes `ServiceAccount` objects.

Regarding the annotation `kubernetes.io/enforce-mountable-secrets` : While the annotation's name suggests it only concerns the mounting of Secrets, its enforcement also extends to other ways Secrets are used in the context of a Pod. Therefore, it is crucial to ensure that all the referenced secrets are correctly specified in the `ServiceAccount`.

## StorageObjectInUseProtection

**Type:** Mutating.

The `StorageObjectInUseProtection` plugin adds the `kubernetes.io/pvc-protection` or `kubernetes.io/pv-protection` finalizers to newly created Persistent Volume Claims (PVCs) or Persistent Volumes (PV). In case a user deletes a PVC or PV the PVC or PV is not removed until the finalizer is removed from the PVC or PV by PVC or PV Protection Controller. Refer to the [Storage Object in Use Protection](#) for more detailed information.

## TaintNodesByCondition

**Type:** Mutating.

This admission controller [taints](#) newly created Nodes as `NotReady` and `NoSchedule`. That tainting avoids a race condition that could cause Pods to be scheduled on new Nodes before their taints were updated to accurately reflect their reported conditions.

## ValidatingAdmissionPolicy

**Type:** Validating.

[This admission controller](#) implements the CEL validation for incoming matched requests. It is enabled when both feature gate `validatingadmissionpolicy` and `admissionregistration.k8s.io/v1alpha1` group/version are enabled. If any of the `ValidatingAdmissionPolicy` fails, the request fails.

## ValidatingAdmissionWebhook

**Type:** Validating.

This admission controller calls any validating webhooks which match the request. Matching webhooks are called in parallel; if any of them rejects the request, the request fails. This admission controller only runs in the validation phase; the webhooks it calls may not mutate the object, as opposed to the webhooks called by the `MutatingAdmissionWebhook` admission controller.

If a webhook called by this has side effects (for example, decrementing quota) it *must* have a reconciliation system, as it is not guaranteed that subsequent webhooks or other validating admission controllers will permit the request to finish.

If you disable the `ValidatingAdmissionWebhook`, you must also disable the `ValidatingWebhookConfiguration` object in the `admissionregistration.k8s.io/v1` group/version via the `--runtime-config` flag.

## Is there a recommended set of admission controllers to use?

Yes. The recommended admission controllers are enabled by default (shown [here](#)), so you do not need to explicitly specify them. You can enable additional admission controllers beyond the default set using the `--enable-admission-plugins` flag (**order doesn't matter**).

# 3.9 - Dynamic Admission Control

In addition to [compiled-in admission plugins](#), admission plugins can be developed as extensions and run as webhooks configured at runtime. This page describes how to build, configure, use, and monitor admission webhooks.

## What are admission webhooks?

Admission webhooks are HTTP callbacks that receive admission requests and do something with them. You can define two types of admission webhooks, [validating admission webhook](#) and [mutating admission webhook](#). Mutating admission webhooks are invoked first, and can modify objects sent to the API server to enforce custom defaults. After all object modifications are complete, and after the incoming object is validated by the API server, validating admission webhooks are invoked and can reject requests to enforce custom policies.

**Note:**

Admission webhooks that need to guarantee they see the final state of the object in order to enforce policy should use a validating admission webhook, since objects can be modified after being seen by mutating webhooks.

## Experimenting with admission webhooks

Admission webhooks are essentially part of the cluster control-plane. You should write and deploy them with great caution. Please read the [user guides](#) for instructions if you intend to write/deploy production-grade admission webhooks. In the following, we describe how to quickly experiment with admission webhooks.

### Prerequisites

- Ensure that `MutatingAdmissionWebhook` and `ValidatingAdmissionWebhook` admission controllers are enabled. [Here](#) is a recommended set of admission controllers to enable in general.
- Ensure that the `admissionregistration.k8s.io/v1` API is enabled.

### Write an admission webhook server

Please refer to the implementation of the [admission webhook server](#) that is validated in a Kubernetes e2e test. The webhook handles the `AdmissionReview` request sent by the API servers, and sends back its decision as an `AdmissionReview` object in the same version it received.

See the [webhook request](#) section for details on the data sent to webhooks.

See the [webhook response](#) section for the data expected from webhooks.

The example admission webhook server leaves the `clientAuth` field

[empty](#), which defaults to `NoClientCert`. This means that the webhook server does not authenticate the identity of the clients, supposedly API servers. If you need mutual TLS or other ways to authenticate the clients, see how to [authenticate API servers](#).

## Deploy the admission webhook service

The webhook server in the e2e test is deployed in the Kubernetes cluster, via the [deployment API](#). The test also creates a [service](#) as the front-end of the webhook server. See [code](#).

You may also deploy your webhooks outside of the cluster. You will need to update your webhook configurations accordingly.

## Configure admission webhooks on the fly

You can dynamically configure what resources are subject to what admission webhooks via [ValidatingWebhookConfiguration](#) or [MutatingWebhookConfiguration](#).

The following is an example `ValidatingWebhookConfiguration`, a mutating webhook configuration is similar. See the [webhook configuration](#) section for details about each config field.

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: "pod-policy.example.com"
webhooks:
- name: "pod-policy.example.com"
  rules:
  - apiGroups: [""]
    apiVersions: ["v1"]
    operations: ["CREATE"]
    resources: ["pods"]
    scope: "Namespaced"
  clientConfig:
    service:
      namespace: "example-namespace"
      name: "example-service"
      caBundle: <CA_BUNDLE>
    admissionReviewVersions: ["v1"]
    sideEffects: None
    timeoutSeconds: 5
```

### Note:

You must replace the `<CA_BUNDLE>` in the above example by a valid CA bundle which is a PEM-encoded (field value is Base64 encoded) CA bundle for validating the webhook's server certificate.

The `scope` field specifies if only cluster-scoped resources ("Cluster") or namespace-scoped resources ("Namespaced") will match this rule. "\*" means that there are no scope restrictions.

### Note:

When using `clientConfig.service`, the server cert must be valid for `<svc_name>.<svc_namespace>.svc`.

**Note:**

Default timeout for a webhook call is 10 seconds. You can set the `timeout` and it is encouraged to use a short timeout for webhooks. If the webhook call times out, the request is handled according to the webhook's failure policy.

When an API server receives a request that matches one of the `rules`, the API server sends an `admissionReview` request to webhook as specified in the `clientConfig`.

After you create the webhook configuration, the system will take a few seconds to honor the new configuration.

## Authenticate API servers

If your admission webhooks require authentication, you can configure the API servers to use basic auth, bearer token, or a cert to authenticate itself to the webhooks. There are three steps to complete the configuration.

- When starting the API server, specify the location of the admission control configuration file via the `--admission-control-config-file` flag.
- In the admission control configuration file, specify where the `MutatingAdmissionWebhook` controller and `ValidatingAdmissionWebhook` controller should read the credentials. The credentials are stored in `kubeConfig` files (yes, the same schema that's used by `kubectl`), so the field name is `kubeConfigFile`. Here is an example admission control configuration file:

[apiserver.config.k8s.io/v1](#) [apiserver.k8s.io/v1alpha1](#)

```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
- name: ValidatingAdmissionWebhook
  configuration:
    apiVersion: apiserver.config.k8s.io/v1
    kind: WebhookAdmissionConfiguration
    kubeConfigFile: "<path-to-kubeconfig-file>"
- name: MutatingAdmissionWebhook
  configuration:
    apiVersion: apiserver.config.k8s.io/v1
    kind: WebhookAdmissionConfiguration
    kubeConfigFile: "<path-to-kubeconfig-file>"
```

For more information about `AdmissionConfiguration`, see the [AdmissionConfiguration \(v1\) reference](#). See the [webhook configuration](#) section for details about each config field.

In the `kubeConfig` file, provide the credentials:

```
apiVersion: v1
kind: Config
users:
  # name should be set to the DNS name of the service or the host (incl
  # If a non-443 port is used for services, it must be included in the
  #
  # For a webhook configured to speak to a service on the default port
  # - name: webhook1.ns1.svc
  #   user: ...
  #
  # For a webhook configured to speak to a service on non-default port
  # - name: webhook1.ns1.svc:8443
  #   user: ...
  #
  # and optionally create a second stanza using only the DNS name of the
  # - name: webhook1.ns1.svc
  #   user: ...
  #
  # For webhooks configured to speak to a URL, match the host (and port)
  # A webhook with `url: https://www.example.com`:
  # - name: www.example.com
  #   user: ...
  #
  # A webhook with `url: https://www.example.com:443`:
  # - name: www.example.com:443
  #   user: ...
  #
  # A webhook with `url: https://www.example.com:8443`:
  # - name: www.example.com:8443
  #   user: ...
  #
  - name: 'webhook1.ns1.svc'
    user:
      client-certificate-data: "<pem encoded certificate>"
      client-key-data: "<pem encoded key>"
  # The `name` supports using * to wildcard-match prefixing segments.
  - name: '*.webhook-company.org'
    user:
      password: "<password>"
      username: "<name>"
  # '*' is the default match.
  - name: '*'
    user:
      token: "<token>"
```

Of course you need to set up the webhook server to handle these authentication requests.

## Webhook request and response

### Request

Webhooks are sent as POST requests, with `Content-Type: application/json`, with an `AdmissionReview` API object in the `admission.k8s.io` API group serialized to JSON as the body.

Webhooks can specify what versions of `AdmissionReview` objects they accept with the `admissionReviewVersions` field in their configuration:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
webhooks:
- name: my-webhook.example.com
  admissionReviewVersions: ["v1", "v1beta1"]
```

`admissionReviewVersions` is a required field when creating webhook configurations. Webhooks are required to support at least one `AdmissionReview` version understood by the current and previous API server.

API servers send the first `AdmissionReview` version in the `admissionReviewVersions` list they support. If none of the versions in the list are supported by the API server, the configuration will not be allowed to be created. If an API server encounters a webhook configuration that was previously created and does not support any of the `AdmissionReview` versions the API server knows how to send, attempts to call to the webhook will fail and be subject to the [failure policy](#).

This example shows the data contained in an `AdmissionReview` object for a request to update the `scale` subresource of an `apps/v1 Deployment`:

```
apiVersion: admission.k8s.io/v1
kind: AdmissionReview
request:
  # Random uid uniquely identifying this admission call
  uid: 705ab4f5-6393-11e8-b7cc-42010a800002

  # Fully-qualified group/version/kind of the incoming object
  kind:
    group: autoscaling
    version: v1
    kind: Scale

  # Fully-qualified group/version/kind of the resource being modified
  resource:
    group: apps
    version: v1
    resource: deployments

  # subresource, if the request is to a subresource
  subResource: scale

  # Fully-qualified group/version/kind of the incoming object in the
  # This only differs from `kind` if the webhook specified `matchPolicy`
  # original request to the API server was converted to a version the
  requestKind:
    group: autoscaling
    version: v1
    kind: Scale

  # Fully-qualified group/version/kind of the resource being modified
  # This only differs from `resource` if the webhook specified `matchPolicy`
  # original request to the API server was converted to a version the
  requestResource:
    group: apps
    version: v1
    resource: deployments

  # subresource, if the request is to a subresource
  # This only differs from `subResource` if the webhook specified `matchPolicy`
  # original request to the API server was converted to a version the
  requestSubResource: scale

  # Name of the resource being modified
  name: my-deployment

  # Namespace of the resource being modified, if the resource is namespaced
  namespace: my-namespace

  # operation can be CREATE, UPDATE, DELETE, or CONNECT
  operation: UPDATE

  userInfo:
    # Username of the authenticated user making the request to the API
    username: admin

    # UID of the authenticated user making the request to the API server
    uid: 014fbff9a07c

    # Group memberships of the authenticated user making the request
    groups:
      - system:authenticated
      - my-admin-group

    # Arbitrary extra info associated with the user making the request
    # This is populated by the API server authentication layer and should
    # if any SubjectAccessReview checks are performed by the webhook.
```

```
extra:
  some-key:
    - some-value1
    - some-value2

  # object is the new object being admitted.
  # It is null for DELETE operations.
  object:
    apiVersion: autoscaling/v1
    kind: Scale

  # oldObject is the existing object.
  # It is null for CREATE and CONNECT operations.
  oldObject:
    apiVersion: autoscaling/v1
    kind: Scale

  # options contains the options for the operation being admitted, like
  # It is null for CONNECT operations.
  options:
    apiVersion: meta.k8s.io/v1
    kind: UpdateOptions

  # dryRun indicates the API request is running in dry run mode and whether
  # Webhooks with side effects should avoid actuating those side effects.
  # See http://k8s.io/docs/reference/using-api/api-concepts/#make-a-dry-run
  dryRun: False
```

## Response

Webhooks respond with a 200 HTTP status code, Content-Type: application/json , and a body containing an AdmissionReview object (in the same version they were sent), with the response stanza populated, serialized to JSON.

At a minimum, the response stanza must contain the following fields:

- uid , copied from the request.uid sent to the webhook
- allowed , either set to true or false

Example of a minimal response from a webhook to allow a request:

```
{
  "apiVersion": "admission.k8s.io/v1",
  "kind": "AdmissionReview",
  "response": {
    "uid": "<value from request.uid>",
    "allowed": true
  }
}
```

Example of a minimal response from a webhook to forbid a request:

```
{
  "apiVersion": "admission.k8s.io/v1",
  "kind": "AdmissionReview",
  "response": {
    "uid": "<value from request.uid>",
    "allowed": false
  }
}
```

When rejecting a request, the webhook can customize the http code and message returned to the user using the `status` field. The specified status object is returned to the user. See the [API documentation](#) for details about the `status` type. Example of a response to forbid a request, customizing the HTTP status code and message presented to the user:

```
{  
  "apiVersion": "admission.k8s.io/v1",  
  "kind": "AdmissionReview",  
  "response": {  
    "uid": "<value from request.uid>",  
    "allowed": false,  
    "status": {  
      "code": 403,  
      "message": "You cannot do this because it is Tuesday and your n  
    }  
  }  
}
```

When allowing a request, a mutating admission webhook may optionally modify the incoming object as well. This is done using the `patch` and `patchType` fields in the response. The only currently supported `patchType` is `JSONPatch`. See [JSON patch](#) documentation for more details. For `patchType: JSONPatch`, the `patch` field contains a base64-encoded array of JSON patch operations.

As an example, a single patch operation that would set `spec.replicas` would be `[{"op": "add", "path": "/spec/replicas", "value": 3}]`

Base64-encoded, this would be

```
W3sib3Ai0iAiYWRkIiwgInBhdGgi0iAiL3NwZWVmcmVwbG1jYXMiLCAidmFsdWUi0iAzfV0  
=
```

So a webhook response to add that label would be:

```
{  
  "apiVersion": "admission.k8s.io/v1",  
  "kind": "AdmissionReview",  
  "response": {  
    "uid": "<value from request.uid>",  
    "allowed": true,  
    "patchType": "JSONPatch",  
    "patch": "W3sib3Ai0iAiYWRkIiwgInBhdGgi0iAiL3NwZWVmcmVwbG1jYXMiLCA  
  }  
}
```

Admission webhooks can optionally return warning messages that are returned to the requesting client in HTTP `Warning` headers with a warning code of 299. Warnings can be sent with allowed or rejected admission responses.

If you're implementing a webhook that returns a warning:

- Don't include a "Warning:" prefix in the message
- Use warning messages to describe problems the client making the API request should correct or be aware of
- Limit warnings to 120 characters if possible

#### Caution:

Individual warning messages over 256 characters may be truncated

by the API server before being returned to clients. If more than 4096 characters of warning messages are added (from all sources), additional warning messages are ignored.

```
{  
  "apiVersion": "admission.k8s.io/v1",  
  "kind": "AdmissionReview",  
  "response": {  
    "uid": "<value from request.uid>",  
    "allowed": true,  
    "warnings": [  
      "duplicate envvar entries specified with name MY_ENV",  
      "memory request less than 4MB specified for container mycontain  
    ]  
  }  
}
```

## Webhook configuration

To register admission webhooks, create `MutatingWebhookConfiguration` or `ValidatingWebhookConfiguration` API objects. The name of a `MutatingWebhookConfiguration` or a `ValidatingWebhookConfiguration` object must be a valid [DNS subdomain name](#).

Each configuration can contain one or more webhooks. If multiple webhooks are specified in a single configuration, each must be given a unique name. This is required in order to make resulting audit logs and metrics easier to match up to active configurations.

Each webhook defines the following things.

### Matching requests: rules

Each webhook must specify a list of rules used to determine if a request to the API server should be sent to the webhook. Each rule specifies one or more operations, apiGroups, apiVersions, and resources, and a resource scope:

- `operations` lists one or more operations to match. Can be `"CREATE"` , `"UPDATE"` , `"DELETE"` , `"CONNECT"` , or `"*"` to match all.
- `apiGroups` lists one or more API groups to match. `""` is the core API group. `"*"` matches all API groups.
- `apiVersions` lists one or more API versions to match. `"*"` matches all API versions.
- `resources` lists one or more resources to match.
  - `"*"` matches all resources, but not subresources.
  - `"*/"` matches all resources and subresources.
  - `"pods/*"` matches all subresources of pods.
  - `"*/status"` matches all status subresources.
- `scope` specifies a scope to match. Valid values are `"Cluster"` , `"Namespaced"` , and `"*"` . Subresources match the scope of their parent resource. Default is `"*"` .
  - `"Cluster"` means that only cluster-scoped resources will match this rule (Namespace API objects are cluster-scoped).
  - `"Namespaced"` means that only namespaced resources will match this rule.

- "\*" means that there are no scope restrictions.

If an incoming request matches one of the specified `operations`, `groups`, `versions`, `resources`, and `scope` for any of a webhook's `rules`, the request is sent to the webhook.

Here are other examples of rules that could be used to specify which resources should be intercepted.

Match `CREATE` or `UPDATE` requests to `apps/v1` and `apps/v1beta1` deployments and `replicasets`:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  rules:
    - operations: ["CREATE", "UPDATE"]
      apiGroups: ["apps"]
      apiVersions: ["v1", "v1beta1"]
      resources: ["deployments", "replicasets"]
      scope: "Namespaced"
...
...
```

Match create requests for all resources (but not subresources) in all API groups and versions:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
webhooks:
- name: my-webhook.example.com
  rules:
    - operations: ["CREATE"]
      apiGroups: ["*"]
      apiVersions: ["*"]
      resources: ["*"]
      scope: "*"
...
```

Match update requests for all `status` subresources in all API groups and versions:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
webhooks:
- name: my-webhook.example.com
  rules:
    - operations: ["UPDATE"]
      apiGroups: ["*"]
      apiVersions: ["*"]
      resources: ["*/status"]
      scope: "*"
...
```

## Matching requests: objectSelector

Webhooks may optionally limit which requests are intercepted based on the labels of the objects they would be sent, by specifying an `objectSelector`. If specified, the `objectSelector` is evaluated against both the `object` and `oldObject` that would be sent to the webhook, and is

considered to match if either object matches the selector.

A null object (`oldObject` in the case of create, or `newObject` in the case of delete), or an object that cannot have labels (like a `DeploymentRollback` or a `PodProxyOptions` object) is not considered to match.

Use the object selector only if the webhook is opt-in, because end users may skip the admission webhook by setting the labels.

This example shows a mutating webhook that would match a `CREATE` of any resource (but not subresources) with the label `foo: bar`:

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
webhooks:
- name: my-webhook.example.com
  objectSelector:
    matchLabels:
      foo: bar
  rules:
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "*"
```

See [labels concept](#) for more examples of label selectors.

## Matching requests: namespaceSelector

Webhooks may optionally limit which requests for namespaced resources are intercepted, based on the labels of the containing namespace, by specifying a `namespaceSelector`.

The `namespaceSelector` decides whether to run the webhook on a request for a namespaced resource (or a Namespace object), based on whether the namespace's labels match the selector. If the object itself is a namespace, the matching is performed on `object.metadata.labels`. If the object is a cluster scoped resource other than a Namespace, `namespaceSelector` has no effect.

This example shows a mutating webhook that matches a `CREATE` of any namespaced resource inside a namespace that does not have a "runlevel" label of "0" or "1":

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
webhooks:
- name: my-webhook.example.com
  namespaceSelector:
    matchExpressions:
    - key: runlevel
      operator: NotIn
      values: ["0", "1"]
  rules:
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "Namespaced"
```

This example shows a validating webhook that matches a `CREATE` of any namespaced resource inside a namespace that is associated with the "environment" of "prod" or "staging":

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
webhooks:
  - name: my-webhook.example.com
    namespaceSelector:
      matchExpressions:
        - key: environment
          operator: In
          values: ["prod", "staging"]
    rules:
      - operations: ["CREATE"]
        apiGroups: ["*"]
        apiVersions: ["*"]
        resources: ["*"]
        scope: "Namespaced"
```

See [labels concept](#) for more examples of label selectors.

## Matching requests: matchPolicy

API servers can make objects available via multiple API groups or versions.

For example, if a webhook only specified a rule for some API groups/versions (like `apiGroups:["apps"]`, `apiVersions:["v1", "v1beta1"]` ), and a request was made to modify the resource via another API group/version (like `extensions/v1beta1` ), the request would not be sent to the webhook.

The `matchPolicy` lets a webhook define how its `rules` are used to match incoming requests. Allowed values are `Exact` or `Equivalent`.

- `Exact` means a request should be intercepted only if it exactly matches a specified rule.
- `Equivalent` means a request should be intercepted if it modifies a resource listed in `rules`, even via another API group or version.

In the example given above, the webhook that only registered for `apps/v1` could use `matchPolicy`:

- `matchPolicy: Exact` would mean the `extensions/v1beta1` request would not be sent to the webhook
- `matchPolicy: Equivalent` means the `extensions/v1beta1` request would be sent to the webhook (with the objects converted to a version the webhook had specified: `apps/v1` )

Specifying `Equivalent` is recommended, and ensures that webhooks continue to intercept the resources they expect when upgrades enable new versions of the resource in the API server.

When a resource stops being served by the API server, it is no longer considered equivalent to other versions of that resource that are still served. For example, `extensions/v1beta1` deployments were first deprecated and then removed (in Kubernetes v1.16).

Since that removal, a webhook with a `apiGroups:["extensions"]`, `apiVersions:["v1beta1"]`, `resources:["deployments"]` rule does not intercept deployments created via `apps/v1` APIs. For that reason, webhooks should prefer registering for stable versions of resources.

This example shows a validating webhook that intercepts modifications to deployments (no matter the API group or version), and is always sent an `apps/v1 Deployment` object:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
webhooks:
- name: my-webhook.example.com
  matchPolicy: Equivalent
  rules:
  - operations: ["CREATE", "UPDATE", "DELETE"]
    apiGroups: ["apps"]
    apiVersions: ["v1"]
    resources: ["deployments"]
    scope: "Namespaced"
```

The `matchPolicy` for an admission webhooks defaults to `Equivalent`.

## Matching requests: `matchConditions`

ⓘ **FEATURE STATE:** Kubernetes v1.30 [stable]

You can define *match conditions* for webhooks if you need fine-grained request filtering. These conditions are useful if you find that match rules, `objectSelectors` and `namespaceSelectors` still doesn't provide the filtering you want over when to call out over HTTP. Match conditions are [CEL expressions](#). All match conditions must evaluate to true for the webhook to be called.

Here is an example illustrating a few different uses for match conditions:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
webhooks:
  - name: my-webhook.example.com
    matchPolicy: Equivalent
    rules:
      - operations: ['CREATE', 'UPDATE']
        apiGroups: ['*']
        apiVersions: ['*']
        resources: ['*']
    failurePolicy: 'Ignore' # Fail-open (optional)
    sideEffects: None
    clientConfig:
      service:
        namespace: my-namespace
        name: my-webhook
        caBundle: '<omitted>'
    # You can have up to 64 matchConditions per webhook
    matchConditions:
      - name: 'exclude-leases' # Each match condition must have a unique expression: '!request.resource.group == "coordination.k8s.io"'
      - name: 'exclude-kubelet-requests'
        expression: '!("system:nodes" in request.userInfo.groups)' #
      - name: 'rbac' # Skip RBAC requests, which are handled by the service account
        expression: 'request.resource.group != "rbac.authorization.k8s.io"'
    # This example illustrates the use of the 'authorizer'. The authorizer is more powerful than a simple expression, so in this example it is scoped to only one webhook. Both webhooks can be served by the same endpoint.
  - name: rbac.my-webhook.example.com
    matchPolicy: Equivalent
    rules:
      - operations: ['CREATE', 'UPDATE']
        apiGroups: ['rbac.authorization.k8s.io']
        apiVersions: ['*']
        resources: ['*']
    failurePolicy: 'Fail' # Fail-closed (the default)
    sideEffects: None
    clientConfig:
      service:
        namespace: my-namespace
        name: my-webhook
        caBundle: '<omitted>'
    # You can have up to 64 matchConditions per webhook
    matchConditions:
      - name: 'breakglass'
        # Skip requests made by users authorized to 'breakglass' on the authorizer
        # The 'breakglass' API verb does not need to exist outside the authorizer
        expression: '!authorizer.group("admissionregistration.k8s.io")'
```

**Note:**

You can define up to 64 elements in the `matchConditions` field per webhook.

Match conditions have access to the following CEL variables:

- `object` - The object from the incoming request. The value is null for DELETE requests. The object version may be converted based on the [matchPolicy](#).
- `oldObject` - The existing object. The value is null for CREATE requests.
- `request` - The request portion of the [AdmissionReview](#), excluding

- object and oldObject .
- `authorizer` - A CEL Authorizer. May be used to perform authorization checks for the principal (authenticated user) of the request. See [Authz](#) in the Kubernetes CEL library documentation for more details.
  - `authorizer.requestResource` - A shortcut for an authorization check configured with the request resource (group, resource, (subresource), namespace, name).

For more information on CEL expressions, refer to the [Common Expression Language in Kubernetes reference](#).

In the event of an error evaluating a match condition the webhook is never called. Whether to reject the request is determined as follows:

1. If **any** match condition evaluated to `false` (regardless of other errors), the API server skips the webhook.
2. Otherwise:
  - for [`failurePolicy: Fail`](#), reject the request (without calling the webhook).
  - for [`failurePolicy: Ignore`](#), proceed with the request but skip the webhook.

## Contacting the webhook

Once the API server has determined a request should be sent to a webhook, it needs to know how to contact the webhook. This is specified in the `clientConfig` stanza of the webhook configuration.

Webhooks can either be called via a URL or a service reference, and can optionally include a custom CA bundle to use to verify the TLS connection.

### URL

`url` gives the location of the webhook, in standard URL form ( `scheme://host:port/path` ).

The `host` should not refer to a service running in the cluster; use a service reference by specifying the `service` field instead. The host might be resolved via external DNS in some API servers (e.g., `kube-apiserver` cannot resolve in-cluster DNS as that would be a layering violation). `host` may also be an IP address.

Please note that using `localhost` or `127.0.0.1` as a `host` is risky unless you take great care to run this webhook on all hosts which run an API server which might need to make calls to this webhook. Such installations are likely to be non-portable or not readily run in a new cluster.

The scheme must be "https"; the URL must begin with "https://".

Attempting to use a user or basic auth (for example `user:password@`) is not allowed. Fragments ( `#...` ) and query parameters ( `?...` ) are also not allowed.

Here is an example of a mutating webhook configured to call a URL (and expects the TLS certificate to be verified using system trust roots, so does not specify a `caBundle`):

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
webhooks:
- name: my-webhook.example.com
  clientConfig:
    url: "https://my-webhook.example.com:9443/my-webhook-path"
```

## Service reference

The `service` stanza inside `clientConfig` is a reference to the service for this webhook. If the webhook is running within the cluster, then you should use `service` instead of `url`. The service namespace and name are required. The port is optional and defaults to 443. The path is optional and defaults to "/".

Here is an example of a mutating webhook configured to call a service on port "1234" at the subpath "/my-path", and to verify the TLS connection against the ServerName `my-service-name.my-service-namespace.svc` using a custom CA bundle:

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
webhooks:
- name: my-webhook.example.com
  clientConfig:
    caBundle: <CA_BUNDLE>
    service:
      namespace: my-service-namespace
      name: my-service-name
      path: /my-path
      port: 1234
```

### Note:

You must replace the `<CA_BUNDLE>` in the above example by a valid CA bundle which is a PEM-encoded CA bundle for validating the webhook's server certificate.

## Side effects

Webhooks typically operate only on the content of the `AdmissionReview` sent to them. Some webhooks, however, make out-of-band changes as part of processing admission requests.

Webhooks that make out-of-band changes ("side effects") must also have a reconciliation mechanism (like a controller) that periodically determines the actual state of the world, and adjusts the out-of-band data modified by the admission webhook to reflect reality. This is because a call to an admission webhook does not guarantee the admitted object will be persisted as is, or at all. Later webhooks can modify the content of the object, a conflict could be encountered while writing to storage, or the server could power off before persisting the object.

Additionally, webhooks with side effects must skip those side-effects when `dryRun: true` admission requests are handled. A webhook must explicitly indicate that it will not have side-effects when run with `dryRun`, or the dry-run request will not be sent to the webhook and the API request will fail instead.

Webhooks indicate whether they have side effects using the `sideEffects`

field in the webhook configuration:

- `None` : calling the webhook will have no side effects.
- `NoneOnDryRun` : calling the webhook will possibly have side effects, but if a request with `dryRun: true` is sent to the webhook, the webhook will suppress the side effects (the webhook is `dryRun` - aware).

Here is an example of a validating webhook indicating it has no side effects on `dryRun: true` requests:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
webhooks:
- name: my-webhook.example.com
  sideEffects: NoneOnDryRun
```

## Timeouts

Because webhooks add to API request latency, they should evaluate as quickly as possible. `timeoutSeconds` allows configuring how long the API server should wait for a webhook to respond before treating the call as a failure.

If the timeout expires before the webhook responds, the webhook call will be ignored or the API call will be rejected based on the [failure policy](#).

The timeout value must be between 1 and 30 seconds.

Here is an example of a validating webhook with a custom timeout of 2 seconds:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
webhooks:
- name: my-webhook.example.com
  timeoutSeconds: 2
```

The timeout for an admission webhook defaults to 10 seconds.

## Reinvocation policy

A single ordering of mutating admissions plugins (including webhooks) does not work for all cases (see <https://issue.k8s.io/64333> as an example). A mutating webhook can add a new sub-structure to the object (like adding a `container` to a `pod`), and other mutating plugins which have already run may have opinions on those new structures (like setting an `imagePullPolicy` on all containers).

To allow mutating admission plugins to observe changes made by other plugins, built-in mutating admission plugins are re-run if a mutating webhook modifies an object, and mutating webhooks can specify a `reinvocationPolicy` to control whether they are reinvoked as well.

`reinvocationPolicy` may be set to `Never` or `IfNeeded`. It defaults to `Never`.

- `Never` : the webhook must not be called more than once in a single admission evaluation.
- `IfNeeded` : the webhook may be called again as part of the

admission evaluation if the object being admitted is modified by other admission plugins after the initial webhook call.

The important elements to note are:

- The number of additional invocations is not guaranteed to be exactly one.
- If additional invocations result in further modifications to the object, webhooks are not guaranteed to be invoked again.
- Webhooks that use this option may be reordered to minimize the number of additional invocations.
- To validate an object after all mutations are guaranteed complete, use a validating admission webhook instead (recommended for webhooks with side-effects).

Here is an example of a mutating webhook opting into being re-invoked if later admission plugins modify the object:

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
webhooks:
- name: my-webhook.example.com
  reinvocationPolicy: IfNeeded
```

Mutating webhooks must be [idempotent](#), able to successfully process an object they have already admitted and potentially modified. This is true for all mutating admission webhooks, since any change they can make in an object could already exist in the user-provided object, but it is essential for webhooks that opt into reinvocation.

## Failure policy

`failurePolicy` defines how unrecognized errors and timeout errors from the admission webhook are handled. Allowed values are `Ignore` or `Fail`.

- `Ignore` means that an error calling the webhook is ignored and the API request is allowed to continue.
- `Fail` means that an error calling the webhook causes the admission to fail and the API request to be rejected.

Here is a mutating webhook configured to reject an API request if errors are encountered calling the admission webhook:

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
webhooks:
- name: my-webhook.example.com
  failurePolicy: Fail
```

The default `failurePolicy` for an admission webhooks is `Fail`.

## Monitoring admission webhooks

The API server provides ways to monitor admission webhook behaviors. These monitoring mechanisms help cluster admins to answer questions like:

1. Which mutating webhook mutated the object in a API request?
2. What change did the mutating webhook applied to the object?
3. Which webhooks are frequently rejecting API requests? What's the reason for a rejection?

## Mutating webhook auditing annotations

Sometimes it's useful to know which mutating webhook mutated the object in a API request, and what change did the webhook apply.

The Kubernetes API server performs [auditing](#) on each mutating webhook invocation. Each invocation generates an auditing annotation capturing if a request object is mutated by the invocation, and optionally generates an annotation capturing the applied patch from the webhook admission response. The annotations are set in the audit event for given request on given stage of its execution, which is then pre-processed according to a certain policy and written to a backend.

The audit level of a event determines which annotations get recorded:

- At `Metadata` audit level or higher, an annotation with key `mutation.webhook.admission.k8s.io/round_{round_idx}_index_{order_idx}` gets logged with JSON payload indicating a webhook gets invoked for given request and whether it mutated the object or not.

For example, the following annotation gets recorded for a webhook being reinvoked. The webhook is ordered the third in the mutating webhook chain, and didn't mutated the request object during the invocation.

```
# the audit event recorded
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "annotations": {
    "mutation.webhook.admission.k8s.io/round_1_index_2": "{\n      # other annotations\n      ...\n    }\n    # other fields\n    ...\n  }
}
```

```
# the annotation value deserialized
{
  "configuration": "my-mutating-webhook-configuration.example",
  "webhook": "my-webhook.example.com",
  "mutated": false
}
```

The following annotation gets recorded for a webhook being invoked in the first round. The webhook is ordered the first in the mutating webhook chain, and mutated the request object during the invocation.

```
# the audit event recorded
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "annotations": {
    "mutation.webhook.admission.k8s.io/round_0_index_0": "{\n      # other annotations\n      ...\n    }\n    # other fields\n    ...\n  }
```

```
# the annotation value deserialized
{
  "configuration": "my-mutating-webhook-configuration.example.com",
  "webhook": "my-webhook-always-mutate.example.com",
  "mutated": true
}
```

- At Request audit level or higher, an annotation with key `patch.webhook.admission.k8s.io/round_{round_idx}_index_{order_idx}` gets logged with JSON payload indicating a webhook gets invoked for given request and what patch gets applied to the request object.

For example, the following annotation gets recorded for a webhook being reinvoked. The webhook is ordered the fourth in the mutating webhook chain, and responded with a JSON patch which got applied to the request object.

```
# the audit event recorded
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "annotations": {
    "patch.webhook.admission.k8s.io/round_1_index_3": "{\"op\": \"add\", \"path\": \"/data/mutation-stage\", \"value\": \"yes\"}\n    # other annotations\n    ...\n  }\n  # other fields\n  ...\n}
```

```
# the annotation value deserialized
{
  "configuration": "my-other-mutating-webhook-configuration.example.com",
  "webhook": "my-webhook-always-mutate.example.com",
  "patchType": "JSONPatch",
  "patch": [
    {
      "op": "add",
      "path": "/data/mutation-stage",
      "value": "yes"
    }
  ]
}
```

## Admission webhook metrics

The API server exposes Prometheus metrics from the `/metrics` endpoint, which can be used for monitoring and diagnosing API server status. The following metrics record status related to admission webhooks.

### API server admission webhook rejection count

Sometimes it's useful to know which admission webhooks are frequently rejecting API requests, and the reason for a rejection.

The API server exposes a Prometheus counter metric recording admission webhook rejections. The metrics are labelled to identify the causes of webhook rejection(s):

- `name` : the name of the webhook that rejected a request.
- `operation` : the operation type of the request, can be one of `CREATE` , `UPDATE` , `DELETE` and `CONNECT` .
- `type` : the admission webhook type, can be one of `admit` and `validating` .
- `error_type` : identifies if an error occurred during the webhook invocation that caused the rejection. Its value can be one of:
  - `calling_webhook_error` : unrecognized errors or timeout errors from the admission webhook happened and the webhook's [Failure policy](#) is set to `Fail` .
  - `no_error` : no error occurred. The webhook rejected the request with `allowed: false` in the admission response. The metrics label `rejection_code` records the `.status.code` set in the admission response.
  - `apiserver_internal_error` : an API server internal error happened.
- `rejection_code` : the HTTP status code set in the admission response when a webhook rejected a request.

Example of the rejection count metrics:

```
# HELP apiserver_admission_webhook_rejection_count [ALPHA] Admission
# TYPE apiserver_admission_webhook_rejection_count counter
apiserver_admission_webhook_rejection_count{error_type="calling_webho
apiserver_admission_webhook_rejection_count{error_type="calling_webho
apiserver_admission_webhook_rejection_count{error_type="no_error",nam
```

## Best practices and warnings

### Idempotence

An idempotent mutating admission webhook is able to successfully process an object it has already admitted and potentially modified. The admission can be applied multiple times without changing the result beyond the initial application.

### Example of idempotent mutating admission webhooks:

1. For a `CREATE` pod request, set the field `.spec.securityContext.runAsNonRoot` of the pod to true, to enforce security best practices.
2. For a `CREATE` pod request, if the field

`.spec.containers[].resources.limits` of a container is not set, set default resource limits.

3. For a `CREATE` pod request, inject a sidecar container with name `foo-sidecar` if no container with the name `foo-sidecar` already exists.

In the cases above, the webhook can be safely reinvoked, or admit an object that already has the fields set.

## Example of non-idempotent mutating admission webhooks:

1. For a `CREATE` pod request, inject a sidecar container with name `foo-sidecar` suffixed with the current timestamp (e.g. `foo-sidecar-19700101-000000` ).
2. For a `CREATE / UPDATE` pod request, reject if the pod has label `"env"` set, otherwise add an `"env": "prod"` label to the pod.
3. For a `CREATE` pod request, blindly append a sidecar container named `foo-sidecar` without looking to see if there is already a `foo-sidecar` container in the pod.

In the first case above, reinvoking the webhook can result in the same sidecar being injected multiple times to a pod, each time with a different container name. Similarly the webhook can inject duplicated containers if the sidecar already exists in a user-provided pod.

In the second case above, reinvoking the webhook will result in the webhook failing on its own output.

In the third case above, reinvoking the webhook will result in duplicated containers in the pod spec, which makes the request invalid and rejected by the API server.

## Intercepting all versions of an object

It is recommended that admission webhooks should always intercept all versions of an object by setting `.webhooks[].matchPolicy` to `Equivalent`. It is also recommended that admission webhooks should prefer registering for stable versions of resources. Failure to intercept all versions of an object can result in admission policies not being enforced for requests in certain versions. See [Matching requests: matchPolicy](#) for examples.

## Availability

It is recommended that admission webhooks should evaluate as quickly as possible (typically in milliseconds), since they add to API request latency. It is encouraged to use a small timeout for webhooks. See [Timeouts](#) for more detail.

It is recommended that admission webhooks should leverage some format of load-balancing, to provide high availability and performance benefits. If a webhook is running within the cluster, you can run multiple webhook backends behind a service to leverage the load-balancing that service supports.

## Guaranteeing the final state of the object is seen

Admission webhooks that need to guarantee they see the final state of the object in order to enforce policy should use a validating admission webhook, since objects can be modified after being seen by mutating webhooks.

For example, a mutating admission webhook is configured to inject a sidecar container with name "foo-sidecar" on every `CREATE` pod request. If the sidecar *must* be present, a validating admission webhook should also be configured to intercept `CREATE` pod requests, and validate that a container with name "foo-sidecar" with the expected configuration exists in the to-be-created object.

## Avoiding deadlocks in self-hosted webhooks

A webhook running inside the cluster might cause deadlocks for its own deployment if it is configured to intercept resources required to start its own pods.

For example, a mutating admission webhook is configured to admit `CREATE` pod requests only if a certain label is set in the pod (e.g. `"env": "prod"` ). The webhook server runs in a deployment which doesn't set the `"env"` label. When a node that runs the webhook server pods becomes unhealthy, the webhook deployment will try to reschedule the pods to another node. However the requests will get rejected by the existing webhook server since the `"env"` label is unset, and the migration cannot happen.

It is recommended to exclude the namespace where your webhook is running with a [namespaceSelector](#).

## Side effects

It is recommended that admission webhooks should avoid side effects if possible, which means the webhooks operate only on the content of the `AdmissionReview` sent to them, and do not make out-of-band changes. The `.webhooks[].sideEffects` field should be set to `None` if a webhook doesn't have any side effect.

If side effects are required during the admission evaluation, they must be suppressed when processing an `AdmissionReview` object with `dryRun` set to `true`, and the `.webhooks[].sideEffects` field should be set to `NoneOnDryRun`. See [Side effects](#) for more detail.

## Avoiding operating on the kube-system namespace

The `kube-system` namespace contains objects created by the Kubernetes system, e.g. service accounts for the control plane components, pods like `kube-dns` . Accidentally mutating or rejecting requests in the `kube-system` namespace may cause the control plane components to stop functioning or introduce unknown behavior. If your admission webhooks don't intend to modify the behavior of the Kubernetes control plane, exclude the `kube-system` namespace from being intercepted using a [namespaceSelector](#) .

## 3.10 - Managing Service Accounts

A *ServiceAccount* provides an identity for processes that run in a Pod.

A process inside a Pod can use the identity of its associated service account to authenticate to the cluster's API server.

For an introduction to service accounts, read [configure service accounts](#).

This task guide explains some of the concepts behind ServiceAccounts. The guide also explains how to obtain or revoke tokens that represent ServiceAccounts.

### Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. It is recommended to run this tutorial on a cluster with at least two nodes that are not acting as control plane hosts. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Killercoda](#)
- [Play with Kubernetes](#)

To be able to follow these steps exactly, ensure you have a namespace named `exampelens`. If you don't, create one by running:

```
kubectl create namespace exampelens
```

### User accounts versus service accounts

Kubernetes distinguishes between the concept of a user account and a service account for a number of reasons:

- User accounts are for humans. Service accounts are for application processes, which (for Kubernetes) run in containers that are part of pods.
- User accounts are intended to be global: names must be unique across all namespaces of a cluster. No matter what namespace you look at, a particular username that represents a user represents the same user. In Kubernetes, service accounts are namespaced: two different namespaces can contain ServiceAccounts that have identical names.
- Typically, a cluster's user accounts might be synchronised from a corporate database, where new user account creation requires special privileges and is tied to complex business processes. By contrast, service account creation is intended to be more lightweight, allowing cluster users to create service accounts for specific tasks on demand. Separating ServiceAccount creation from the steps to onboard human users makes it easier for workloads to follow the principle of least privilege.
- Auditing considerations for humans and service accounts may differ; the separation makes that easier to achieve.
- A configuration bundle for a complex system may include definition

of various service accounts for components of that system. Because service accounts can be created without many constraints and have namespaced names, such configuration is usually portable.

## Bound service account tokens

ServiceAccount tokens can be bound to API objects that exist in the kube-apiserver. This can be used to tie the validity of a token to the existence of another API object. Supported object types are as follows:

- Pod (used for projected volume mounts, see below)
- Secret (can be used to allow revoking a token by deleting the Secret)
- Node (in v1.30, creating new node-bound tokens is alpha, using existing node-bound tokens is beta)

When a token is bound to an object, the object's `metadata.name` and `metadata.uid` are stored as extra 'private claims' in the issued JWT.

When a bound token is presented to the kube-apiserver, the service account authenticator will extract and verify these claims. If the referenced object or the ServiceAccount is pending deletion (for example, due to finalizers), then for any instant that is 60 seconds (or more) after the `.metadata.deletionTimestamp` date, authentication with that token would fail. If the referenced object no longer exists (or its `metadata.uid` does not match), the request will not be authenticated.

## Additional metadata in Pod bound tokens

### ⓘ FEATURE STATE: Kubernetes v1.30 [beta]

When a service account token is bound to a Pod object, additional metadata is also embedded into the token that indicates the value of the bound pod's `spec.nodeName` field, and the uid of that Node, if available.

This node information is **not** verified by the kube-apiserver when the token is used for authentication. It is included so integrators do not have to fetch Pod or Node API objects to check the associated Node name and uid when inspecting a JWT.

## Verifying and inspecting private claims

The `TokenReview` API can be used to verify and extract private claims from a token:

1. First, assume you have a pod named `test-pod` and a service account named `my-sa`.
2. Create a token that is bound to this Pod:

```
kubectl create token my-sa --bound-object-kind="Pod" --bound-object-r
```

3. Copy this token into a new file named `tokenreview.yaml`:

```
apiVersion: authentication.k8s.io/v1
kind: TokenReview
spec:
  token: <token from step 2>
```

4. Submit this resource to the apiserver for review:

```
kubectl create -o yaml -f tokenreview.yaml # we use '-o yaml' so we can
```

You should see an output like below:

```
apiVersion: authentication.k8s.io/v1
kind: TokenReview
metadata:
  creationTimestamp: null
spec:
  token: <token>
status:
  audiences:
  - https://kubernetes.default.svc.cluster.local
  authenticated: true
  user:
    extra:
      authentication.kubernetes.io/credential-id:
      - JTI=7ee52be0-9045-4653-aa5e-0da57b8dccdc
      authentication.kubernetes.io/node-name:
      - kind-control-plane
      authentication.kubernetes.io/node-uid:
      - 497e9d9a-47aa-4930-b0f6-9f2fb574c8c6
      authentication.kubernetes.io/pod-name:
      - test-pod
      authentication.kubernetes.io/pod-uid:
      - e87dbbd6-3d7e-45db-aafb-72b24627dff5
  groups:
  - system:serviceaccounts
  - system:serviceaccounts:default
  - system:authenticated
  uid: f8b4161b-2e2b-11e9-86b7-2afc33b31a7e
  username: system:serviceaccount:default:my-sa
```

#### Note:

Despite using `kubectl create -f` to create this resource, and defining it similar to other resource types in Kubernetes, TokenReview is a special type and the kube-apiserver does not actually persist the TokenReview object into etcd. Hence `kubectl get tokenreview` is not a valid command.

## Bound service account token volume mechanism

ⓘ **FEATURE STATE:** Kubernetes v1.22 [stable]

By default, the Kubernetes control plane (specifically, the [ServiceAccount admission controller](#)) adds a [projected volume](#) to Pods, and this volume includes a token for Kubernetes API access.

Here's an example of how that looks for a launched Pod:

```
...
  - name: kube-api-access-
    projected:
      sources:
        - serviceAccountToken:
          path: token # must match the path the app expects
        - configMap:
          items:
            - key: ca.crt
            path: ca.crt
            name: kube-root-ca.crt
        - downwardAPI:
          items:
            - fieldRef:
              apiVersion: v1
              fieldPath: metadata.namespace
            path: namespace
```

That manifest snippet defines a projected volume that consists of three sources. In this case, each source also represents a single path within that volume. The three sources are:

1. A `serviceAccountToken` source, that contains a token that the kubelet acquires from kube-apiserver. The kubelet fetches time-bound tokens using the TokenRequest API. A token served for a TokenRequest expires either when the pod is deleted or after a defined lifespan (by default, that is 1 hour). The kubelet also refreshes that token before the token expires. The token is bound to the specific Pod and has the kube-apiserver as its audience. This mechanism superseded an earlier mechanism that added a volume based on a Secret, where the Secret represented the ServiceAccount for the Pod, but did not expire.
2. A `configMap` source. The ConfigMap contains a bundle of certificate authority data. Pods can use these certificates to make sure that they are connecting to your cluster's kube-apiserver (and not to middlebox or an accidentally misconfigured peer).
3. A `downwardAPI` source that looks up the name of the namespace containing the Pod, and makes that name information available to application code running inside the Pod.

Any container within the Pod that mounts this particular volume can access the above information.

**Note:**

There is no specific mechanism to invalidate a token issued via TokenRequest. If you no longer trust a bound service account token for a Pod, you can delete that Pod. Deleting a Pod expires its bound service account tokens.

## Manual Secret management for ServiceAccounts

Versions of Kubernetes before v1.22 automatically created credentials for accessing the Kubernetes API. This older mechanism was based on creating token Secrets that could then be mounted into running Pods.

In more recent versions, including Kubernetes v1.31, API credentials are [obtained directly](#) using the [TokenRequest API](#), and are mounted into Pods using a projected volume. The tokens obtained using this method

have bounded lifetimes, and are automatically invalidated when the Pod they are mounted into is deleted.

You can still [manually create](#) a Secret to hold a service account token; for example, if you need a token that never expires.

Once you manually create a Secret and link it to a ServiceAccount, the Kubernetes control plane automatically populates the token into that Secret.

**Note:**

Although the manual mechanism for creating a long-lived ServiceAccount token exists, using [TokenRequest](#) to obtain short-lived API access tokens is recommended instead.

## Auto-generated legacy ServiceAccount token clean up

Before version 1.24, Kubernetes automatically generated Secret-based tokens for ServiceAccounts. To distinguish between automatically generated tokens and manually created ones, Kubernetes checks for a reference from the ServiceAccount's secrets field. If the Secret is referenced in the `secrets` field, it is considered an auto-generated legacy token. Otherwise, it is considered a manually created legacy token. For example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-robot
  namespace: default
secrets:
  - name: build-robot-secret # usually NOT present for a manually gen
```

Beginning from version 1.29, legacy ServiceAccount tokens that were generated automatically will be marked as invalid if they remain unused for a certain period of time (set to default at one year). Tokens that continue to be unused for this defined period (again, by default, one year) will subsequently be purged by the control plane.

If users use an invalidated auto-generated token, the token validator will

1. add an audit annotation for the key-value pair  
`authentication.k8s.io/legacy-token-invalidated: <secret name>/<namespace>`,
2. increment the `invalid_legacy_auto_token_uses_total` metric count,
3. update the Secret label `kubernetes.io/legacy-token-last-used` with the new date,
4. return an error indicating that the token has been invalidated.

When receiving this validation error, users can update the Secret to remove the `kubernetes.io/legacy-token-invalid-since` label to temporarily allow use of this token.

Here's an example of an auto-generated legacy token that has been marked with the `kubernetes.io/legacy-token-last-used` and `kubernetes.io/legacy-token-invalid-since` labels:

```
apiVersion: v1
kind: Secret
metadata:
  name: build-robot-secret
  namespace: default
  labels:
    kubernetes.io/legacy-token-last-used: 2022-10-24
    kubernetes.io/legacy-token-invalid-since: 2023-10-25
  annotations:
    kubernetes.io/service-account.name: build-robot
type: kubernetes.io/service-account-token
```

## Control plane details

### ServiceAccount controller

A ServiceAccount controller manages the ServiceAccounts inside namespaces, and ensures a ServiceAccount named "default" exists in every active namespace.

### Token controller

The service account token controller runs as part of `kube-controller-manager`. This controller acts asynchronously. It:

- watches for ServiceAccount deletion and deletes all corresponding ServiceAccount token Secrets.
- watches for ServiceAccount token Secret addition, and ensures the referenced ServiceAccount exists, and adds a token to the Secret if needed.
- watches for Secret deletion and removes a reference from the corresponding ServiceAccount if needed.

You must pass a service account private key file to the token controller in the `kube-controller-manager` using the `--service-account-private-key-file` flag. The private key is used to sign generated service account tokens. Similarly, you must pass the corresponding public key to the `kube-apiserver` using the `--service-account-key-file` flag. The public key will be used to verify the tokens during authentication.

### ServiceAccount admission controller

The modification of pods is implemented via a plugin called an [Admission Controller](#). It is part of the API server. This admission controller acts synchronously to modify pods as they are created. When this plugin is active (and it is by default on most distributions), then it does the following when a Pod is created:

1. If the pod does not have a `.spec.serviceAccountName` set, the admission controller sets the name of the ServiceAccount for this incoming Pod to `default`.
2. The admission controller ensures that the ServiceAccount referenced by the incoming Pod exists. If there is no ServiceAccount with a matching name, the admission controller rejects the incoming Pod. That check applies even for the `default` ServiceAccount.
3. Provided that neither the ServiceAccount's `automountServiceAccountToken` field nor the Pod's `automountServiceAccountToken` field is set to `false`:
  - the admission controller mutates the incoming Pod, adding an

- extra `volume` that contains a token for API access.
- the admission controller adds a `volumeMount` to each container in the Pod, skipping any containers that already have a volume mount defined for the path `/var/run/secrets/kubernetes.io/serviceaccount`. For Linux containers, that volume is mounted at `/var/run/secrets/kubernetes.io/serviceaccount`; on Windows nodes, the mount is at the equivalent path.
4. If the spec of the incoming Pod doesn't already contain any `imagePullSecrets`, then the admission controller adds `imagePullSecrets`, copying them from the `ServiceAccount`.

## Legacy ServiceAccount token tracking controller

### ⓘ **FEATURE STATE:** Kubernetes v1.28 [stable]

This controller generates a ConfigMap called `kube-system/kube-apiserver-legacy-service-account-token-tracking` in the `kube-system` namespace. The ConfigMap records the timestamp when legacy service account tokens began to be monitored by the system.

## Legacy ServiceAccount token cleaner

### ⓘ **FEATURE STATE:** Kubernetes v1.30 [stable]

The legacy ServiceAccount token cleaner runs as part of the `kube-controller-manager` and checks every 24 hours to see if any auto-generated legacy ServiceAccount token has not been used in a *specified amount of time*. If so, the cleaner marks those tokens as invalid.

The cleaner works by first checking the ConfigMap created by the control plane (provided that `LegacyServiceAccountTokenTracking` is enabled). If the current time is a *specified amount of time* after the date in the ConfigMap, the cleaner then loops through the list of Secrets in the cluster and evaluates each Secret that has the type `kubernetes.io/service-account-token`.

If a Secret meets all of the following conditions, the cleaner marks it as invalid:

- The Secret is auto-generated, meaning that it is bi-directionally referenced by a ServiceAccount.
- The Secret is not currently mounted by any pods.
- The Secret has not been used in a *specified amount of time* since it was created or since it was last used.

The cleaner marks a Secret invalid by adding a label called `kubernetes.io/legacy-token-invalid-since` to the Secret, with the current date as the value. If an invalid Secret is not used in a *specified amount of time*, the cleaner will delete it.

#### **Note:**

All the *specified amount of time* above defaults to one year. The cluster administrator can configure this value through the `--legacy-service-account-token-clean-up-period` command line argument for the `kube-controller-manager` component.

## TokenRequest API

## ⓘ FEATURE STATE: Kubernetes v1.22 [stable]

You use the [TokenRequest](#) subresource of a ServiceAccount to obtain a time-bound token for that ServiceAccount. You don't need to call this to obtain an API token for use within a container, since the kubelet sets this up for you using a *projected volume*.

If you want to use the TokenRequest API from `kubectl`, see [Manually create an API token for a ServiceAccount](#).

The Kubernetes control plane (specifically, the ServiceAccount admission controller) adds a projected volume to Pods, and the kubelet ensures that this volume contains a token that lets containers authenticate as the right ServiceAccount.

(This mechanism superseded an earlier mechanism that added a volume based on a Secret, where the Secret represented the ServiceAccount for the Pod but did not expire.)

Here's an example of how that looks for a launched Pod:

```
...
- name: kube-api-access-<random-suffix>
  projected:
    defaultMode: 420 # decimal equivalent of octal 0644
    sources:
      - serviceAccountToken:
          expirationSeconds: 3607
          path: token
      - configMap:
          items:
            - key: ca.crt
              path: ca.crt
            name: kube-root-ca.crt
      - downwardAPI:
          items:
            - fieldRef:
                apiVersion: v1
                fieldPath: metadata.namespace
                path: namespace
```

That manifest snippet defines a projected volume that combines information from three sources:

1. A `serviceAccountToken` source, that contains a token that the kubelet acquires from kube-apiserver. The kubelet fetches time-bound tokens using the TokenRequest API. A token served for a TokenRequest expires either when the pod is deleted or after a defined lifespan (by default, that is 1 hour). The token is bound to the specific Pod and has the kube-apiserver as its audience.
2. A `configMap` source. The ConfigMap contains a bundle of certificate authority data. Pods can use these certificates to make sure that they are connecting to your cluster's kube-apiserver (and not to middlebox or an accidentally misconfigured peer).
3. A `downwardAPI` source. This `downwardAPI` volume makes the name of the namespace containing the Pod available to application code running inside the Pod.

Any container within the Pod that mounts this volume can access the above information.

# Create additional API tokens

**Caution:**

Only create long-lived API tokens if the [token request](#) mechanism is not suitable. The token request mechanism provides time-limited tokens; because these expire, they represent a lower risk to information security.

To create a non-expiring, persisted API token for a ServiceAccount, create a Secret of type `kubernetes.io/service-account-token` with an annotation referencing the ServiceAccount. The control plane then generates a long-lived token and updates that Secret with that generated token data.

Here is a sample manifest for such a Secret:

[secret/serviceaccount/mysecretname.yaml](#) 

```
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: mysecretname
  annotations:
    kubernetes.io/service-account.name: myserviceaccount
```

To create a Secret based on this example, run:

```
kubectl -n exemplens create -f https://k8s.io/examples/secret/service
```

To see the details for that Secret, run:

```
kubectl -n exemplens describe secret mysecretname
```

The output is similar to:

```
Name:      mysecretname
Namespace: exemplens
Labels:    <none>
Annotations:  kubernetes.io/service-account.name=myserviceaccount
              kubernetes.io/service-account.uid=8a85c4c4-8483-11e9-
              000000000000

Type:  kubernetes.io/service-account-token

Data
=====
ca.crt:  1362 bytes
namespace:  9 bytes
token:  ...
```

If you launch a new Pod into the `exemplens` namespace, it can use the `myserviceaccount` service-account-token Secret that you just created.

**Caution:**

Do not reference manually created Secrets in the `secrets` field of a ServiceAccount. Or the manually created Secrets will be cleaned if it is not used for a long time. Please refer to [auto-generated legacy ServiceAccount token clean up](#).

## Delete/invalidate a ServiceAccount token

If you know the name of the Secret that contains the token you want to remove:

```
kubectl delete secret name-of-secret
```

Otherwise, first find the Secret for the ServiceAccount.

```
# This assumes that you already have a namespace named 'examplens'  
kubectl -n examplens get serviceaccount/example-automated-thing -o yaml
```

The output is similar to:

```
apiVersion: v1  
kind: ServiceAccount  
metadata:  
  annotations:  
    kubectl.kubernetes.io/last-applied-configuration: |  
      {"apiVersion":"v1","kind":"ServiceAccount","metadata":{"annotations":{},"  
  creationTimestamp: "2019-07-21T07:07:07Z"  
  name: example-automated-thing  
  namespace: examplens  
  resourceVersion: "777"  
  selfLink: /api/v1/namespaces/examplens/serviceaccounts/example-automated-thing  
  uid: f23fd170-66f2-4697-b049-e1e266b7f835  
  secrets:  
    - name: example-automated-thing-token-zyxwv
```

Then, delete the Secret you now know the name of:

```
kubectl -n examplens delete secret/example-automated-thing-token-zyxwv
```

## Clean up

If you created a namespace `examplens` to experiment with, you can remove it:

```
kubectl delete namespace examplens
```

## What's next

- Read more details about [projected volumes](#).

## 3.11 - Certificates and Certificate Signing Requests

Kubernetes certificate and trust bundle APIs enable automation of [X.509](#) credential provisioning by providing a programmatic interface for clients of the Kubernetes API to request and obtain X.509 certificates from a Certificate Authority (CA).

There is also experimental (alpha) support for distributing [trust bundles](#).

### Certificate signing requests

ⓘ **FEATURE STATE:** Kubernetes v1.19 [stable]

A [CertificateSigningRequest](#) (CSR) resource is used to request that a certificate be signed by a denoted signer, after which the request may be approved or denied before finally being signed.

#### Request signing process

The CertificateSigningRequest resource type allows a client to ask for an X.509 certificate be issued, based on a signing request. The CertificateSigningRequest object includes a PEM-encoded PKCS#10 signing request in the `spec.request` field. The CertificateSigningRequest denotes the signer (the recipient that the request is being made to) using the `spec.signerName` field. Note that `spec.signerName` is a required key after API version `certificates.k8s.io/v1`. In Kubernetes v1.22 and later, clients may optionally set the `spec.expirationSeconds` field to request a particular lifetime for the issued certificate. The minimum valid value for this field is `600`, i.e. ten minutes.

Once created, a CertificateSigningRequest must be approved before it can be signed. Depending on the signer selected, a CertificateSigningRequest may be automatically approved by a [controller](#). Otherwise, a CertificateSigningRequest must be manually approved either via the REST API (or client-go) or by running `kubectl certificate approve`. Likewise, a CertificateSigningRequest may also be denied, which tells the configured signer that it must not sign the request.

For certificates that have been approved, the next step is signing. The relevant signing controller first validates that the signing conditions are met and then creates a certificate. The signing controller then updates the CertificateSigningRequest, storing the new certificate into the `status.certificate` field of the existing CertificateSigningRequest object. The `status.certificate` field is either empty or contains a X.509 certificate, encoded in PEM format. The CertificateSigningRequest `status.certificate` field is empty until the signer does this.

Once the `status.certificate` field has been populated, the request has been completed and clients can now fetch the signed certificate PEM data from the CertificateSigningRequest resource. The signers can instead deny certificate signing if the approval conditions are not met.

In order to reduce the number of old CertificateSigningRequest resources left in a cluster, a garbage collection controller runs periodically. The garbage collection removes CertificateSigningRequests that have not changed state for some duration:

- Approved requests: automatically deleted after 1 hour

- Denied requests: automatically deleted after 1 hour
- Failed requests: automatically deleted after 1 hour
- Pending requests: automatically deleted after 24 hours
- All requests: automatically deleted after the issued certificate has expired

## Certificate signing authorization

To allow creating a CertificateSigningRequest and retrieving any CertificateSigningRequest:

- Verbs: `create` , `get` , `list` , `watch` , group: `certificates.k8s.io` , resource: `certificatesigningrequests`

For example:

[access/certificate-signing-request/clusterrole-create.yaml](#) 

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: csr-creator
rules:
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests
  verbs:
  - create
  - get
  - list
  - watch
```

To allow approving a CertificateSigningRequest:

- Verbs: `get` , `list` , `watch` , group: `certificates.k8s.io` , resource: `certificatesigningrequests`
- Verbs: `update` , group: `certificates.k8s.io` , resource: `certificatesigningrequests/approval`
- Verbs: `approve` , group: `certificates.k8s.io` , resource: `signers` , resourceName: `<signerNameDomain>/<signerNamePath>` or `<signerNameDomain>/*`

For example:

[access/certificate-signing-request/clusterrole-approve.yaml](#) 

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: csr-approver
rules:
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests/approval
  verbs:
  - update
- apiGroups:
  - certificates.k8s.io
  resources:
  - signers
  resourceNames:
  - example.com/my-signer-name # example.com/* can be used to authorize requests from multiple domains
  verbs:
  - approve
```

To allow signing a CertificateSigningRequest:

- Verbs: get , list , watch , group: certificates.k8s.io , resource: certificatesigningrequests
- Verbs: update , group: certificates.k8s.io , resource: certificatesigningrequests/status
- Verbs: sign , group: certificates.k8s.io , resource: signers , resourceName: <signerNameDomain>/<signerNamePath> or <signerNameDomain>/\*

[access/certificate-signing-request/clusterrole-sign.yaml](#) 

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: csr-signer
rules:
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests/status
  verbs:
  - update
- apiGroups:
  - certificates.k8s.io
  resources:
  - signers
  resourceNames:
  - example.com/my-signer-name # example.com/* can be used to authorize
  verbs:
  - sign
```

## Signers

Signers abstractly represent the entity or entities that might sign, or have signed, a security certificate.

Any signer that is made available for outside a particular cluster should provide information about how the signer works, so that consumers can understand what that means for CertificateSigningRequests and (if enabled) [ClusterTrustBundles](#). This includes:

1. **Trust distribution:** how trust anchors (CA certificates or certificate bundles) are distributed.
2. **Permitted subjects:** any restrictions on and behavior when a disallowed subject is requested.
3. **Permitted x509 extensions:** including IP subjectAltNames, DNS subjectAltNames, Email subjectAltNames, URI subjectAltNames etc, and behavior when a disallowed extension is requested.
4. **Permitted key usages / extended key usages:** any restrictions on and behavior when usages different than the signer-determined usages are specified in the CSR.
5. **Expiration/certificate lifetime:** whether it is fixed by the signer, configurable by the admin, determined by the CSR `spec.expirationSeconds` field, etc and the behavior when the signer-determined expiration is different from the CSR `spec.expirationSeconds` field.
6. **CA bit allowed/disallowed:** and behavior if a CSR contains a request a for a CA certificate when the signer does not permit it.

Commonly, the `status.certificate` field of a CertificateSigningRequest contains a single PEM-encoded X.509 certificate once the CSR is approved

and the certificate is issued. Some signers store multiple certificates into the `status.certificate` field. In that case, the documentation for the signer should specify the meaning of additional certificates; for example, this might be the certificate plus intermediates to be presented during TLS handshakes.

If you want to make the *trust anchor* (root certificate) available, this should be done separately from a `CertificateSigningRequest` and its `status.certificate` field. For example, you could use a `ClusterTrustBundle`.

The PKCS#10 signing request format does not have a standard mechanism to specify a certificate expiration or lifetime. The expiration or lifetime therefore has to be set through the `spec.expirationSeconds` field of the `CSR` object. The built-in signers use the `ClusterSigningDuration` configuration option, which defaults to 1 year, (the `--cluster-signing-duration` command-line flag of the `kube-controller-manager`) as the default when no `spec.expirationSeconds` is specified. When `spec.expirationSeconds` is specified, the minimum of `spec.expirationSeconds` and `ClusterSigningDuration` is used.

**Note:**

The `spec.expirationSeconds` field was added in Kubernetes v1.22. Earlier versions of Kubernetes do not honor this field. Kubernetes API servers prior to v1.22 will silently drop this field when the object is created.

## Kubernetes signers

Kubernetes provides built-in signers that each have a well-known `signerName`:

1. `kubernetes.io/kube-apiserver-client` : signs certificates that will be honored as client certificates by the API server. Never auto-approved by `kube-controller-manager`.
  1. Trust distribution: signed certificates must be honored as client certificates by the API server. The CA bundle is not distributed by any other means.
  2. Permitted subjects - no subject restrictions, but approvers and signers may choose not to approve or sign. Certain subjects like cluster-admin level users or groups vary between distributions and installations, but deserve additional scrutiny before approval and signing. The `CertificateSubjectRestriction` admission plugin is enabled by default to restrict `system:masters`, but it is often not the only cluster-admin subject in a cluster.
  3. Permitted x509 extensions - honors `subjectAltName` and `keyUsage` extensions and discards other extensions.
  4. Permitted key usages - must include `["client auth"]`. Must not include key usages beyond `["digital signature", "key encipherment", "client auth"]`.
  5. Expiration/certificate lifetime - for the `kube-controller-manager` implementation of this signer, set to the minimum of the `--cluster-signing-duration` option or, if specified, the `spec.expirationSeconds` field of the `CSR` object.
  6. CA bit allowed/disallowed - not allowed.
2. `kubernetes.io/kube-apiserver-client-kubelet` : signs client certificates that will be honored as client certificates by the API server. May be auto-approved by `kube-controller-manager`.

1. Trust distribution: signed certificates must be honored as client certificates by the API server. The CA bundle is not distributed by any other means.
  2. Permitted subjects - organizations are exactly `["system:nodes"]`, common name is `"system:node: ${NODE_NAME}"`.
  3. Permitted x509 extensions - honors key usage extensions, forbids subjectAltName extensions and drops other extensions.
  4. Permitted key usages - `["key encipherment", "digital signature", "client auth"]` or `["digital signature", "client auth"]`.
  5. Expiration/certificate lifetime - for the kube-controller-manager implementation of this signer, set to the minimum of the `--cluster-signing-duration` option or, if specified, the `spec.expirationSeconds` field of the CSR object.
  6. CA bit allowed/disallowed - not allowed.
3. `kubernetes.io/kubelet-serving` : signs serving certificates that are honored as a valid kubelet serving certificate by the API server, but has no other guarantees. Never auto-approved by `kube-controller-manager`.
1. Trust distribution: signed certificates must be honored by the API server as valid to terminate connections to a kubelet. The CA bundle is not distributed by any other means.
  2. Permitted subjects - organizations are exactly `["system:nodes"]`, common name is `"system:node: ${NODE_NAME}"`.
  3. Permitted x509 extensions - honors key usage and DNSName/ IPAddress subjectAltName extensions, forbids EmailAddress and URI subjectAltName extensions, drops other extensions. At least one DNS or IP subjectAltName must be present.
  4. Permitted key usages - `["key encipherment", "digital signature", "server auth"]` or `["digital signature", "server auth"]`.
  5. Expiration/certificate lifetime - for the kube-controller-manager implementation of this signer, set to the minimum of the `--cluster-signing-duration` option or, if specified, the `spec.expirationSeconds` field of the CSR object.
  6. CA bit allowed/disallowed - not allowed.
4. `kubernetes.io/legacy-unknown` : has no guarantees for trust at all. Some third-party distributions of Kubernetes may honor client certificates signed by it. The stable CertificateSigningRequest API (version `certificates.k8s.io/v1` and later) does not allow to set the `signerName` as `kubernetes.io/legacy-unknown`. Never auto-approved by `kube-controller-manager`.
1. Trust distribution: None. There is no standard trust or distribution for this signer in a Kubernetes cluster.
  2. Permitted subjects - any
  3. Permitted x509 extensions - honors subjectAltName and key usage extensions and discards other extensions.
  4. Permitted key usages - any
  5. Expiration/certificate lifetime - for the kube-controller-manager implementation of this signer, set to the minimum of the `--cluster-signing-duration` option or, if specified, the `spec.expirationSeconds` field of the CSR object.
  6. CA bit allowed/disallowed - not allowed.

The `kube-controller-manager` implements [control plane signing](#) for each

of the built in signers. Failures for all of these are only reported in kube-controller-manager logs.

**Note:**

The `spec.expirationSeconds` field was added in Kubernetes v1.22. Earlier versions of Kubernetes do not honor this field. Kubernetes API servers prior to v1.22 will silently drop this field when the object is created.

Distribution of trust happens out of band for these signers. Any trust outside of those described above are strictly coincidental. For instance, some distributions may honor `kubernetes.io/legacy-unknown` as client certificates for the kube-apiserver, but this is not a standard. None of these usages are related to ServiceAccount token secrets `.data[ca.crt]` in any way. That CA bundle is only guaranteed to verify a connection to the API server using the default service (`kubernetes.default.svc`).

## Custom signers

You can also introduce your own custom signer, which should have a similar prefixed name but using your own domain name. For example, if you represent an open source project that uses the domain `open-fictional.example` then you might use `issuer.open-fictional.example/service-mesh` as a signer name.

A custom signer uses the Kubernetes API to issue a certificate. See [API-based signers](#).

# Signing

## Control plane signer

The Kubernetes control plane implements each of the [Kubernetes signers](#), as part of the kube-controller-manager.

**Note:**

Prior to Kubernetes v1.18, the kube-controller-manager would sign any CSRs that were marked as approved.

**Note:**

The `spec.expirationSeconds` field was added in Kubernetes v1.22. Earlier versions of Kubernetes do not honor this field. Kubernetes API servers prior to v1.22 will silently drop this field when the object is created.

## API-based signers

Users of the REST API can sign CSRs by submitting an UPDATE request to the `status` subresource of the CSR to be signed.

As part of this request, the `status.certificate` field should be set to contain the signed certificate. This field contains one or more PEM-encoded certificates.

All PEM blocks must have the "CERTIFICATE" label, contain no headers, and the encoded data must be a BER-encoded ASN.1 Certificate structure as described in [section 4 of RFC5280](#).

Example certificate content:

```
-----BEGIN CERTIFICATE-----  
MIIDgjCCAmqgAwIBAgIUC1N1EJ4Qnsd322BhDPRwmg3b/oAwDQYJKoZIhvcNAQEL  
BQAwXDELMAkGA1UEBhMCeHgxCjAIBgNVBAgMAXgxCjAIBgNVBAcMAXgxCjAIBgNV  
BAoMAXgxCjAIBgNVBAsMAXgxCzAJBgNVBAMMAMhMRAwDgYJKoZIhvcNAQkBFgF4  
MB4XDTIwMDcwNjIyMDcwMFoXDTI1MDcwNTIyMDcwMFowNzEVMBMGA1UEChMMc31z  
dGVt0m5vZGVzMR4wHAYDVQQDExVzeXN0ZW06bm9kZToxMjcuMC4wLjEwggEiMA0G  
CSqGSIB3DQEBAQUAA4IBDwAwggEKAoIBAQDne5X2eQ1JcLzkKvhzCR4Hx19+ZmU3  
+e1zf0ywLdoQxrPi+o4hVsUH3q0y52BmA7u1yehHDRSAq9u62cmi5ekgXhXHzGmm  
kmW5n0itRECV3SFsSm2DSghRKf0mm6iTYHWDHzUXKdm9lPPWoSOxoR5oq0sm3JEh  
Q7Et13wrvTJqBMJo1GTwQuF+HY0ku0NF/DLqbZIcpI08yQKyrBgYz2u051/oNp8a  
sTCsV40UfyHhx2BBLUo4g4SptHFySTBwlpRWBnSjZP0hmN74JcpTLB4J5f4iEeA7  
2QytZfADckG4wVkhH3C2EJUmRtFIBVirwDn39GXkSG1nvnMgF3uLZ6zNAgMBAAGj  
YTBfMA4GA1UdDwEB/wQEAvIFoDATBgnVHSUEDAKBgggrBgfFBQcDAjAMBgNVHRMB  
Af8EAjAAMB0GA1UdDgQWBBTRE12hW541kQBDeVCcd2f2VS1B1DALBgNVHREEBDAC  
ggAwDQYJKoZIhvcNAQELBQADggEBABpZjuIKTq8pCaX8dMEGPWtAykgLsTcD2jYr  
L0/TCrqmuaaliUa42jQTt20VsVP/L8offFunj/KjpQU0bvKJPLMRKtmxbhXuQCQi1  
qCRkp8o93mHvEz3mTUN+D1cfQ2fpsBENLnpS0F4G/JyY2Vrh19/X8+mImMEK5e0y  
o0BMby7byUj98WmcUvNCiXbC6F45QTmkwEhMqWns0JZQY+/XeDhEcg+1Jvz9Eyo2  
aGgPseye1o3DpyXnyfJWAWh0z7cikS5X2adesbgI86PhEHBXPIJ1v13ZdfCExmdd  
M1fLPhLyR54fGaY+7/X8P9AZzPefAkwizeXwe9ii6/a08vWoIE4=  
-----END CERTIFICATE-----
```

Non-PEM content may appear before or after the CERTIFICATE PEM blocks and is unvalidated, to allow for explanatory text as described in [section 5.2 of RFC7468](#).

When encoded in JSON or YAML, this field is base-64 encoded. A CertificateSigningRequest containing the example certificate above would look like this:

```
apiVersion: certificates.k8s.io/v1  
kind: CertificateSigningRequest  
...  
status:  
certificate: "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JS..."
```

## Approval or rejection

Before a [signer](#) issues a certificate based on a CertificateSigningRequest, the signer typically checks that the issuance for that CSR has been *approved*.

### Control plane automated approval

The kube-controller-manager ships with a built-in approver for certificates with a signerName of `kubernetes.io/kube-apiserver-client-kubelet` that delegates various permissions on CSRs for node credentials to authorization. The kube-controller-manager POSTs SubjectAccessReview resources to the API server in order to check authorization for certificate approval.

### Approval or rejection using `kubectl`

A Kubernetes administrator (with appropriate permissions) can manually approve (or deny) CertificateSigningRequests by using the `kubectl certificate approve` and `kubectl certificate deny` commands.

To approve a CSR with kubectl:

```
kubectl certificate approve <certificate-signing-request-name>
```

Likewise, to deny a CSR:

```
kubectl certificate deny <certificate-signing-request-name>
```

## Approval or rejection using the Kubernetes API

Users of the REST API can approve CSRs by submitting an UPDATE request to the `approval` subresource of the CSR to be approved. For example, you could write an `operator` that watches for a particular kind of CSR and then sends an UPDATE to approve them.

When you make an approval or rejection request, set either the `Approved` or `Denied` status condition based on the state you determine:

For Approved CSRs:

```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
...
status:
  conditions:
  - lastUpdateTime: "2020-02-08T11:37:35Z"
    lastTransitionTime: "2020-02-08T11:37:35Z"
    message: Approved by my custom approver controller
    reason: ApprovedByMyPolicy # You can set this to any string
    type: Approved
```

For Denied CSRs:

```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
...
status:
  conditions:
  - lastUpdateTime: "2020-02-08T11:37:35Z"
    lastTransitionTime: "2020-02-08T11:37:35Z"
    message: Denied by my custom approver controller
    reason: DeniedByMyPolicy # You can set this to any string
    type: Denied
```

It's usual to set `status.conditions.reason` to a machine-friendly reason code using TitleCase; this is a convention but you can set it to anything you like. If you want to add a note for human consumption, use the `status.conditions.message` field.

## Cluster trust bundles

ⓘ **FEATURE STATE:** Kubernetes v1.27 [alpha]

**Note:**

In Kubernetes 1.31, you must enable the [ClusterTrustBundle feature gate](#) and the [certificates.k8s.io/v1alpha1](#) API group in order to use this API.

A ClusterTrustBundles is a cluster-scoped object for distributing X.509 trust anchors (root certificates) to workloads within the cluster. They're designed to work well with the [signer](#) concept from CertificateSigningRequests.

ClusterTrustBundles can be used in two modes: [signer-linked](#) and [signer-unlinked](#).

## Common properties and validation

All ClusterTrustBundle objects have strong validation on the contents of their `trustBundle` field. That field must contain one or more X.509 certificates, DER-serialized, each wrapped in a PEM `CERTIFICATE` block. The certificates must parse as valid X.509 certificates.

Esoteric PEM features like inter-block data and intra-block headers are either rejected during object validation, or can be ignored by consumers of the object. Additionally, consumers are allowed to reorder the certificates in the bundle with their own arbitrary but stable ordering.

ClusterTrustBundle objects should be considered world-readable within the cluster. If your cluster uses [RBAC](#) authorization, all ServiceAccounts have a default grant that allows them to **get**, **list**, and **watch** all ClusterTrustBundle objects. If you use your own authorization mechanism and you have enabled ClusterTrustBundles in your cluster, you should set up an equivalent rule to make these objects public within the cluster, so that they work as intended.

If you do not have permission to list cluster trust bundles by default in your cluster, you can impersonate a service account you have access to in order to see available ClusterTrustBundles:

```
kubectl get clustertrustbundles --as='system:serviceaccount:mynamepsa'
```

## Signer-linked ClusterTrustBundles

Signer-linked ClusterTrustBundles are associated with a *signer name*, like this:

```
apiVersion: certificates.k8s.io/v1alpha1
kind: ClusterTrustBundle
metadata:
  name: example.com:mysigner:foo
spec:
  signerName: example.com/mysigner
  trustBundle: "<... PEM data ...>"
```

These ClusterTrustBundles are intended to be maintained by a signer-specific controller in the cluster, so they have several security features:

- To create or update a signer-linked ClusterTrustBundle, you must be permitted to **attest** on the signer (custom authorization verb `attest`, API group `certificates.k8s.io`; resource path `signers`). You can configure authorization for the specific resource name `<signerNameDomain>/<signerNamePath>` or match a pattern such as

- ```
<signerNameDomain>* .
```
- Signer-linked ClusterTrustBundles **must** be named with a prefix derived from their `spec.signerName` field. Slashes ( / ) are replaced with colons ( : ), and a final colon is appended. This is followed by an arbitrary name. For example, the signer `example.com/mysigner` can be linked to a ClusterTrustBundle  
`example.com:mysigner:<arbitrary-name> .`

Signer-linked ClusterTrustBundles will typically be consumed in workloads by a combination of a [field selector](#) on the signer name, and a separate [label selector](#).

## Signer-unlinked ClusterTrustBundles

Signer-unlinked ClusterTrustBundles have an empty `spec.signerName` field, like this:

```
apiVersion: certificates.k8s.io/v1alpha1
kind: ClusterTrustBundle
metadata:
  name: foo
spec:
  # no signerName specified, so the field is blank
  trustBundle: "<... PEM data ...>"
```

They are primarily intended for cluster configuration use cases. Each signer-unlinked ClusterTrustBundle is an independent object, in contrast to the customary grouping behavior of signer-linked ClusterTrustBundles.

Signer-unlinked ClusterTrustBundles have no `attest` verb requirement. Instead, you control access to them directly using the usual mechanisms, such as role-based access control.

To distinguish them from signer-linked ClusterTrustBundles, the names of signer-unlinked ClusterTrustBundles **must not** contain a colon ( : ).

## Accessing ClusterTrustBundles from pods

ⓘ **FEATURE STATE:** Kubernetes v1.29 [alpha]

The contents of ClusterTrustBundles can be injected into the container filesystem, similar to ConfigMaps and Secrets. See the [clusterTrustBundle projected volume source](#) for more details.

## How to issue a certificate for a user

A few steps are required in order to get a normal user to be able to authenticate and invoke an API. First, this user must have a certificate issued by the Kubernetes cluster, and then present that certificate to the Kubernetes API.

### Create private key

The following scripts show how to generate PKI private key and CSR. It is important to set CN and O attribute of the CSR. CN is the name of the user and O is the group that this user will belong to. You can refer to

[RBAC](#) for standard groups.

```
openssl genrsa -out myuser.key 2048
openssl req -new -key myuser.key -out myuser.csr -subj "/CN=myuser"
```

## Create a CertificateSigningRequest

Create a [CertificateSigningRequest](#) and submit it to a Kubernetes Cluster via kubectl. Below is a script to generate the CertificateSigningRequest.

```
cat <<EOF | kubectl apply -f -
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: myuser
spec:
  request: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBDRVNULS0tLS0KTU1JQ1ZqQ0
  signerName: kubernetes.io/kube-apiserver-client
  expirationSeconds: 86400 # one day
  usages:
  - client auth
EOF
```

Some points to note:

- usages has to be ' client auth '
- expirationSeconds could be made longer (i.e. 864000 for ten days) or shorter (i.e. 3600 for one hour)
- request is the base64 encoded value of the CSR file content. You can get the content using this command:

```
cat myuser.csr | base64 | tr -d "\n"
```

## Approve the CertificateSigningRequest

Use kubectl to create a CSR and approve it.

Get the list of CSRs:

```
kubectl get csr
```

Approve the CSR:

```
kubectl certificate approve myuser
```

## Get the certificate

Retrieve the certificate from the CSR:

```
kubectl get csr/myuser -o yaml
```

The certificate value is in Base64-encoded format under `status.certificate` .

Export the issued certificate from the `CertificateSigningRequest`.

```
kubectl get csr myuser -o jsonpath='{.status.certificate}' | base64 -d
```

## Create Role and RoleBinding

With the certificate created it is time to define the Role and RoleBinding for this user to access Kubernetes cluster resources.

This is a sample command to create a Role for this new user:

```
kubectl create role developer --verb=create --verb=get --verb=list --
```

This is a sample command to create a RoleBinding for this new user:

```
kubectl create rolebinding developer-binding-myuser --role=developer
```

## Add to kubeconfig

The last step is to add this user into the `kubeconfig` file.

First, you need to add new credentials:

```
kubectl config set-credentials myuser --client-key=myuser.key --client
```

Then, you need to add the context:

```
kubectl config set-context myuser --cluster=kubernetes --user=myuser
```

To test it, change the context to `myuser` :

```
kubectl config use-context myuser
```

## What's next

- Read [Manage TLS Certificates in a Cluster](#)
- View the source code for the `kube-controller-manager` built in [signer](#)
- View the source code for the `kube-controller-manager` built in [approver](#)
- For details of X.509 itself, refer to [RFC 5280](#) section 3.1
- For information on the syntax of PKCS#10 certificate signing requests, refer to [RFC 2986](#)
- Read about the `ClusterTrustBundle` API:
  - [ClusterTrustBundle API reference](#)

# 3.12 - Mapping PodSecurityPolicies to Pod Security Standards

The tables below enumerate the configuration parameters on `PodSecurityPolicy` objects, whether the field mutates and/or validates pods, and how the configuration values map to the [Pod Security Standards](#).

For each applicable parameter, the allowed values for the [Baseline](#) and [Restricted](#) profiles are listed. Anything outside the allowed values for those profiles would fall under the [Privileged](#) profile. "No opinion" means all values are allowed under all Pod Security Standards.

For a step-by-step migration guide, see [Migrate from PodSecurityPolicy to the Built-In PodSecurity Admission Controller](#).

## PodSecurityPolicy Spec

The fields enumerated in this table are part of the `PodSecurityPolicySpec`, which is specified under the `.spec` field path.

| PodSecurityPolicySpec                 | Type                  | Pod Security Standards Equivalent                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------------------------|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>privileged</code>               | Validating            | <b>Baseline &amp; Restricted:</b> <code>false</code> / undefined / nil                                                                                                                                                                                                                                                                                                                                                              |
| <code>defaultAddCapabilities</code>   | Mutating & Validating | Requirements match <code>allowedCapabilities</code> below.                                                                                                                                                                                                                                                                                                                                                                          |
| <code>allowedCapabilities</code>      | Validating            | <b>Baseline:</b> subset of <ul style="list-style-type: none"><li>• AUDIT_WRITE</li><li>• CHOWN</li><li>• DAC_OVERRIDE</li><li>• FOWNER</li><li>• FSETID</li><li>• KILL</li><li>• MKNOD</li><li>• NET_BIND_SERVICE</li><li>• SETFCAP</li><li>• SETGID</li><li>• SETPCAP</li><li>• SETUID</li><li>• SYS_CHROOT</li></ul><br><b>Restricted:</b> empty / undefined / nil OR a list containing <i>only</i> <code>NET_BIND_SERVICE</code> |
| <code>requiredDropCapabilities</code> | Mutating & Validating | <b>Baseline:</b> no opinion<br><b>Restricted:</b> must include <code>ALL</code>                                                                                                                                                                                                                                                                                                                                                     |

|                    |                                   |                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| volumes            | Validating                        | <p><b>Baseline:</b> anything except</p> <ul style="list-style-type: none"><li>• hostPath</li><li>• *</li></ul> <p><b>Restricted:</b> subset of</p> <ul style="list-style-type: none"><li>• configMap</li><li>• csi</li><li>• downwardAPI</li><li>• emptyDir</li><li>• ephemeral</li><li>• persistentVolumeClaim</li><li>• projected</li><li>• secret</li></ul>                                            |
| hostNetwork        | Validating                        | <p><b>Baseline &amp; Restricted:</b> false / undefined / nil</p>                                                                                                                                                                                                                                                                                                                                          |
| hostPorts          | Validating                        | <p><b>Baseline &amp; Restricted:</b> undefined / nil / empty</p>                                                                                                                                                                                                                                                                                                                                          |
| hostPID            | Validating                        | <p><b>Baseline &amp; Restricted:</b> false / undefined / nil</p>                                                                                                                                                                                                                                                                                                                                          |
| hostIPC            | Validating                        | <p><b>Baseline &amp; Restricted:</b> false / undefined / nil</p>                                                                                                                                                                                                                                                                                                                                          |
| seLinux            | Mutating & Validating             | <p><b>Baseline &amp; Restricted:</b> seLinux.rule is MustRunAs , with the following options</p> <ul style="list-style-type: none"><li>• user is unset ( "" / undefined / nil)</li><li>• role is unset ( "" / undefined / nil)</li><li>• type is unset or one of:<br/>    container_t,<br/>    container_init_t,<br/>    container_kvm_t,<br/>    container_engine_t</li><li>• level is anything</li></ul> |
| runAsUser          | Mutating & Validating             | <p><b>Baseline:</b> Anything</p> <p><b>Restricted:</b> rule is MustRunAsNonRoot</p>                                                                                                                                                                                                                                                                                                                       |
| runAsGroup         | Mutating (MustRunAs) & Validating | <p><i>No opinion</i></p>                                                                                                                                                                                                                                                                                                                                                                                  |
| supplementalGroups | Mutating & Validating             | <p><i>No opinion</i></p>                                                                                                                                                                                                                                                                                                                                                                                  |
| fsGroup            | Mutating & Validating             | <p><i>No opinion</i></p>                                                                                                                                                                                                                                                                                                                                                                                  |

|                                           |                       |                                                                                                 |
|-------------------------------------------|-----------------------|-------------------------------------------------------------------------------------------------|
| readOnlyRootFilesystem                    | Mutating & Validating | <i>No opinion</i>                                                                               |
| defaultAllowPrivilegeEscalation           | Mutating              | <i>No opinion (non-validating)</i>                                                              |
| allowPrivilegeEscalation                  | Mutating & Validating | <i>Only mutating if set to false</i><br><b>Baseline:</b> No opinion<br><b>Restricted:</b> false |
| allowedHostPaths                          | Validating            | <i>No opinion (volumes takes precedence)</i>                                                    |
| allowedFlexVolumes                        | Validating            | <i>No opinion (volumes takes precedence)</i>                                                    |
| allowedCSIDrivers                         | Validating            | <i>No opinion (volumes takes precedence)</i>                                                    |
| allowedUnsafeSysctls                      | Validating            | <b>Baseline &amp; Restricted:</b> undefined / nil / empty                                       |
| forbiddenSysctls                          | Validating            | <i>No opinion</i>                                                                               |
| allowedProcMountTypes<br>(alpha feature)  | Validating            | <b>Baseline &amp; Restricted:</b> ["Default"] OR undefined / nil / empty                        |
| runtimeClass<br>.defaultRuntimeClassName  | Mutating              | <i>No opinion</i>                                                                               |
| runtimeClass<br>.allowedRuntimeClassNames | Validating            | <i>No opinion</i>                                                                               |

## PodSecurityPolicy annotations

The [annotations](#) enumerated in this table can be specified under `.metadata.annotations` on the PodSecurityPolicy object.

| PSP Annotation                                                        | Type       | Pod Security Standards Equivalent                                                                                                                                                                                           |
|-----------------------------------------------------------------------|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>seccomp.security.alpha.kubernetes.io/defaultProfileName</code>  | Mutating   | <i>No opinion</i>                                                                                                                                                                                                           |
| <code>seccomp.security.alpha.kubernetes.io/allowedProfileNames</code> | Validating | <b>Baseline:</b> "runtime/default," (Trailing comma to allow unset)<br><br><b>Restricted:</b> "runtime/default" (No trailing comma)<br><br><i>localhost/* values are also permitted for both Baseline &amp; Restricted.</i> |

---

|                                                              |            |                                                                                                                                                                                                                                                                       |
|--------------------------------------------------------------|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| apparmor.security.beta.kubernetes.io<br>/defaultProfileName  | Mutating   | <i>No opinion</i>                                                                                                                                                                                                                                                     |
| apparmor.security.beta.kubernetes.io<br>/allowedProfileNames | Validating | <b>Baseline:</b> "runtime/<br>default," ( <i>Trailing comma<br/>to allow unset</i> )<br><br><b>Restricted:</b> "runtime/<br>default" ( <i>No trailing<br/>comma</i> )<br><br><i>localhost/* values are<br/>also permitted for both<br/>Baseline &amp; Restricted.</i> |

# 3.13 - Kubelet authentication/authorization

## Overview

A kubelet's HTTPS endpoint exposes APIs which give access to data of varying sensitivity, and allow you to perform operations with varying levels of power on the node and within containers.

This document describes how to authenticate and authorize access to the kubelet's HTTPS endpoint.

## Kubelet authentication

By default, requests to the kubelet's HTTPS endpoint that are not rejected by other configured authentication methods are treated as anonymous requests, and given a username of `system:anonymous` and a group of `system:unauthenticated`.

To disable anonymous access and send `401 Unauthorized` responses to unauthenticated requests:

- start the kubelet with the `--anonymous-auth=false` flag

To enable X509 client certificate authentication to the kubelet's HTTPS endpoint:

- start the kubelet with the `--client-ca-file` flag, providing a CA bundle to verify client certificates with
- start the apiserver with `--kubelet-client-certificate` and `--kubelet-client-key` flags
- see the [apiserver authentication documentation](#) for more details

To enable API bearer tokens (including service account tokens) to be used to authenticate to the kubelet's HTTPS endpoint:

- ensure the `authentication.k8s.io/v1beta1` API group is enabled in the API server
- start the kubelet with the `--authentication-token-webhook` and `--kubeconfig` flags
- the kubelet calls the `TokenReview` API on the configured API server to determine user information from bearer tokens

## Kubelet authorization

Any request that is successfully authenticated (including an anonymous request) is then authorized. The default authorization mode is `AlwaysAllow`, which allows all requests.

There are many possible reasons to subdivide access to the kubelet API:

- anonymous auth is enabled, but anonymous users' ability to call the kubelet API should be limited
- bearer token auth is enabled, but arbitrary API users' (like service accounts) ability to call the kubelet API should be limited
- client certificate auth is enabled, but only some of the client certificates signed by the configured CA should be allowed to use the kubelet API

To subdivide access to the kubelet API, delegate authorization to the API server:

- ensure the `authorization.k8s.io/v1beta1` API group is enabled in the API server
- start the kubelet with the `--authorization-mode=Webhook` and the `--kubeconfig` flags
- the kubelet calls the `SubjectAccessReview` API on the configured API server to determine whether each request is authorized

The kubelet authorizes API requests using the same [request attributes](#) approach as the apiserver.

The verb is determined from the incoming request's HTTP verb:

---

| <b>HTTP verb</b> | <b>request verb</b> |
|------------------|---------------------|
|------------------|---------------------|

---

|      |        |
|------|--------|
| POST | create |
|------|--------|

---

|           |     |
|-----------|-----|
| GET, HEAD | get |
|-----------|-----|

---

|     |        |
|-----|--------|
| PUT | update |
|-----|--------|

---

|       |       |
|-------|-------|
| PATCH | patch |
|-------|-------|

---

|        |        |
|--------|--------|
| DELETE | delete |
|--------|--------|

The resource and subresource is determined from the incoming request's path:

---

| <b>Kubelet API</b> | <b>resource</b> | <b>subresource</b> |
|--------------------|-----------------|--------------------|
|--------------------|-----------------|--------------------|

---

|          |       |       |
|----------|-------|-------|
| /stats/* | nodes | stats |
|----------|-------|-------|

---

|            |       |         |
|------------|-------|---------|
| /metrics/* | nodes | metrics |
|------------|-------|---------|

---

|         |       |     |
|---------|-------|-----|
| /logs/* | nodes | log |
|---------|-------|-----|

---

|         |       |      |
|---------|-------|------|
| /spec/* | nodes | spec |
|---------|-------|------|

---

|                   |       |       |
|-------------------|-------|-------|
| <i>all others</i> | nodes | proxy |
|-------------------|-------|-------|

The namespace and API group attributes are always an empty string, and the resource name is always the name of the kubelet's `Node` API object.

When running in this mode, ensure the user identified by the `--kubelet-client-certificate` and `--kubelet-client-key` flags passed to the apiserver is authorized for the following attributes:

- `verb=*, resource=nodes, subresource=proxy`
- `verb=*, resource=nodes, subresource=stats`
- `verb=*, resource=nodes, subresource=log`
- `verb=*, resource=nodes, subresource=spec`
- `verb=*, resource=nodes, subresource=metrics`

## 3.14 - TLS bootstrapping

In a Kubernetes cluster, the components on the worker nodes - kubelet and kube-proxy - need to communicate with Kubernetes control plane components, specifically kube-apiserver. In order to ensure that communication is kept private, not interfered with, and ensure that each component of the cluster is talking to another trusted component, we strongly recommend using client TLS certificates on nodes.

The normal process of bootstrapping these components, especially worker nodes that need certificates so they can communicate safely with kube-apiserver, can be a challenging process as it is often outside of the scope of Kubernetes and requires significant additional work. This in turn, can make it challenging to initialize or scale a cluster.

In order to simplify the process, beginning in version 1.4, Kubernetes introduced a certificate request and signing API. The proposal can be found [here](#).

This document describes the process of node initialization, how to set up TLS client certificate bootstrapping for kubelets, and how it works.

## Initialization process

When a worker node starts up, the kubelet does the following:

1. Look for its `kubeconfig` file
2. Retrieve the URL of the API server and credentials, normally a TLS key and signed certificate from the `kubeconfig` file
3. Attempt to communicate with the API server using the credentials.

Assuming that the kube-apiserver successfully validates the kubelet's credentials, it will treat the kubelet as a valid node, and begin to assign pods to it.

Note that the above process depends upon:

- Existence of a key and certificate on the local host in the `kubeconfig`
- The certificate having been signed by a Certificate Authority (CA) trusted by the kube-apiserver

All of the following are responsibilities of whoever sets up and manages the cluster:

1. Creating the CA key and certificate
2. Distributing the CA certificate to the control plane nodes, where kube-apiserver is running
3. Creating a key and certificate for each kubelet; strongly recommended to have a unique one, with a unique CN, for each kubelet
4. Signing the kubelet certificate using the CA key
5. Distributing the kubelet key and signed certificate to the specific node on which the kubelet is running

The TLS Bootstrapping described in this document is intended to simplify, and partially or even completely automate, steps 3 onwards, as these are the most common when initializing or scaling a cluster.

## Bootstrap initialization

In the bootstrap initialization process, the following occurs:

1. kubelet begins
2. kubelet sees that it does *not* have a `kubeconfig` file
3. kubelet searches for and finds a `bootstrap-kubeconfig` file
4. kubelet reads its bootstrap file, retrieving the URL of the API server and a limited usage "token"
5. kubelet connects to the API server, authenticates using the token
6. kubelet now has limited credentials to create and retrieve a certificate signing request (CSR)
7. kubelet creates a CSR for itself with the signerName set to `kubernetes.io/kube-apiserver-client-kubelet`
8. CSR is approved in one of two ways:
  - If configured, kube-controller-manager automatically approves the CSR
  - If configured, an outside process, possibly a person, approves the CSR using the Kubernetes API or via `kubectl`
9. Certificate is created for the kubelet
10. Certificate is issued to the kubelet
11. kubelet retrieves the certificate
12. kubelet creates a proper `kubeconfig` with the key and signed certificate
13. kubelet begins normal operation
14. Optional: if configured, kubelet automatically requests renewal of the certificate when it is close to expiry
15. The renewed certificate is approved and issued, either automatically or manually, depending on configuration.

The rest of this document describes the necessary steps to configure TLS Bootstrapping, and its limitations.

## Configuration

To configure for TLS bootstrapping and optional automatic approval, you must configure options on the following components:

- `kube-apiserver`
- `kube-controller-manager`
- `kubelet`
- in-cluster resources: `ClusterRoleBinding` and potentially `ClusterRole`

In addition, you need your Kubernetes Certificate Authority (CA).

## Certificate Authority

As without bootstrapping, you will need a Certificate Authority (CA) key and certificate. As without bootstrapping, these will be used to sign the kubelet certificate. As before, it is your responsibility to distribute them to control plane nodes.

For the purposes of this document, we will assume these have been distributed to control plane nodes at `/var/lib/kubernetes/ca.pem` (certificate) and `/var/lib/kubernetes/ca-key.pem` (key). We will refer to these as "Kubernetes CA certificate and key".

All Kubernetes components that use these certificates - kubelet, kube-apiserver, kube-controller-manager - assume the key and certificate to be PEM-encoded.

# kube-apiserver configuration

The kube-apiserver has several requirements to enable TLS bootstrapping:

- Recognizing CA that signs the client certificate
- Authenticating the bootstrapping kubelet to the `system:bootstrappers` group
- Authorize the bootstrapping kubelet to create a certificate signing request (CSR)

## Recognizing client certificates

This is normal for all client certificate authentication. If not already set, add the `--client-ca-file=FILENAME` flag to the kube-apiserver command to enable client certificate authentication, referencing a certificate authority bundle containing the signing certificate, for example `--client-ca-file=/var/lib/kubernetes/ca.pem`.

## Initial bootstrap authentication

In order for the bootstrapping kubelet to connect to kube-apiserver and request a certificate, it must first authenticate to the server. You can use any [authenticator](#) that can authenticate the kubelet.

While any authentication strategy can be used for the kubelet's initial bootstrap credentials, the following two authenticators are recommended for ease of provisioning.

1. [Bootstrap Tokens](#)
2. [Token authentication file](#)

Using bootstrap tokens is a simpler and more easily managed method to authenticate kubelets, and does not require any additional flags when starting kube-apiserver.

Whichever method you choose, the requirement is that the kubelet be able to authenticate as a user with the rights to:

1. create and retrieve CSRs
2. be automatically approved to request node client certificates, if automatic approval is enabled.

A kubelet authenticating using bootstrap tokens is authenticated as a user in the group `system:bootstrappers`, which is the standard method to use.

As this feature matures, you should ensure tokens are bound to a Role Based Access Control (RBAC) policy which limits requests (using the [bootstrap token](#)) strictly to client requests related to certificate provisioning. With RBAC in place, scoping the tokens to a group allows for great flexibility. For example, you could disable a particular bootstrap group's access when you are done provisioning the nodes.

## Bootstrap tokens

Bootstrap tokens are described in detail [here](#). These are tokens that are stored as secrets in the Kubernetes cluster, and then issued to the individual kubelet. You can use a single token for an entire cluster, or issue one per worker node.

The process is two-fold:

1. Create a Kubernetes secret with the token ID, secret and scope(s).
2. Issue the token to the kubelet

From the kubelet's perspective, one token is like another and has no special meaning. From the kube-apiserver's perspective, however, the bootstrap token is special. Due to its `type`, `namespace` and `name`, kube-apiserver recognizes it as a special token, and grants anyone authenticating with that token special bootstrap rights, notably treating them as a member of the `system:bootstrappers` group. This fulfills a basic requirement for TLS bootstrapping.

The details for creating the secret are available [here](#).

If you want to use bootstrap tokens, you must enable it on kube-apiserver with the flag:

```
--enable-bootstrap-token-auth=true
```

## Token authentication file

kube-apiserver has the ability to accept tokens as authentication. These tokens are arbitrary but should represent at least 128 bits of entropy derived from a secure random number generator (such as `/dev/urandom` on most modern Linux systems). There are multiple ways you can generate a token. For example:

```
head -c 16 /dev/urandom | od -An -t x | tr -d ' '
```

This will generate tokens that look like `02b50b05283e98dd0fd71db496ef01e8`.

The token file should look like the following example, where the first three values can be anything and the quoted group name should be as depicted:

```
02b50b05283e98dd0fd71db496ef01e8,kubelet-bootstrap,10001,"system:boot
```

Add the `--token-auth-file=FILENAME` flag to the kube-apiserver command (in your systemd unit file perhaps) to enable the token file. See docs [here](#) for further details.

## Authorize kubelet to create CSR

Now that the bootstrapping node is *authenticated* as part of the `system:bootstrappers` group, it needs to be *authorized* to create a certificate signing request (CSR) as well as retrieve it when done. Fortunately, Kubernetes ships with a `ClusterRole` with precisely these (and only these) permissions, `system:node-bootstrapper`.

To do this, you only need to create a `ClusterRoleBinding` that binds the `system:bootstrappers` group to the cluster role `system:node-bootstrapper`.

```
# enable bootstrapping nodes to create CSR
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: create-csrs-for-bootstrapping
subjects:
- kind: Group
  name: system:bootstrappers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: system:node-bootstrapper
  apiGroup: rbac.authorization.k8s.io
```

## kube-controller-manager configuration

While the apiserver receives the requests for certificates from the kubelet and authenticates those requests, the controller-manager is responsible for issuing actual signed certificates.

The controller-manager performs this function via a certificate-issuing control loop. This takes the form of a [cfssl](#) local signer using assets on disk. Currently, all certificates issued have one year validity and a default set of key usages.

In order for the controller-manager to sign certificates, it needs the following:

- access to the "Kubernetes CA key and certificate" that you created and distributed
- enabling CSR signing

### Access to key and certificate

As described earlier, you need to create a Kubernetes CA key and certificate, and distribute it to the control plane nodes. These will be used by the controller-manager to sign the kubelet certificates.

Since these signed certificates will, in turn, be used by the kubelet to authenticate as a regular kubelet to kube-apiserver, it is important that the CA provided to the controller-manager at this stage also be trusted by kube-apiserver for authentication. This is provided to kube-apiserver with the flag `--client-ca-file=FILENAME` (for example, `--client-ca-file=/var/lib/kubernetes/ca.pem`), as described in the kube-apiserver configuration section.

To provide the Kubernetes CA key and certificate to kube-controller-manager, use the following flags:

```
--cluster-signing-cert-file="/etc/path/to/kubernetes/ca/ca.crt" --clu
```

For example:

```
--cluster-signing-cert-file="/var/lib/kubernetes/ca.pem" --cluster-si
```

The validity duration of signed certificates can be configured with flag:

```
--cluster-signing-duration
```

## Approval

In order to approve CSRs, you need to tell the controller-manager that it is acceptable to approve them. This is done by granting RBAC permissions to the correct group.

There are two distinct sets of permissions:

- `nodeclient` : If a node is creating a new certificate for a node, then it does not have a certificate yet. It is authenticating using one of the tokens listed above, and thus is part of the group `system:bootstrappers` .
- `selfnodeclient` : If a node is renewing its certificate, then it already has a certificate (by definition), which it uses continuously to authenticate as part of the group `system:nodes` .

To enable the kubelet to request and receive a new certificate, create a `ClusterRoleBinding` that binds the group in which the bootstrapping node is a member `system:bootstrappers` to the `clusterRole` that grants it permission,

```
system:certificates.k8s.io:certificatesigningrequests:nodeclient :
```

```
# Approve all CSRs for the group "system:bootstrappers"
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: auto-approve-csrs-for-group
subjects:
- kind: Group
  name: system:bootstrappers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: system:certificates.k8s.io:certificatesigningrequests:nodeclient
  apiGroup: rbac.authorization.k8s.io
```

To enable the kubelet to renew its own client certificate, create a `ClusterRoleBinding` that binds the group in which the fully functioning node is a member `system:nodes` to the `clusterRole` that grants it permission,

```
system:certificates.k8s.io:certificatesigningrequests:selfnodeclient :
```

```
# Approve renewal CSRs for the group "system:nodes"
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: auto-approve-renewals-for-nodes
subjects:
- kind: Group
  name: system:nodes
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: system:certificates.k8s.io:certificatesigningrequests:selfnode
  apiGroup: rbac.authorization.k8s.io
```

The `csrapproving` controller that ships as part of [kube-controller-manager](#) and is enabled by default. The controller uses the [SubjectAccessReview API](#) to determine if a given user is authorized to request a CSR, then approves based on the authorization outcome. To prevent conflicts with other approvers, the built-in approver doesn't explicitly deny CSRs. It only ignores unauthorized requests. The controller also prunes expired certificates as part of garbage collection.

## kubelet configuration

Finally, with the control plane nodes properly set up and all of the necessary authentication and authorization in place, we can configure the kubelet.

The kubelet requires the following configuration to bootstrap:

- A path to store the key and certificate it generates (optional, can use default)
- A path to a `kubeconfig` file that does not yet exist; it will place the bootstrapped config file here
- A path to a bootstrap `kubeconfig` file to provide the URL for the server and bootstrap credentials, e.g. a bootstrap token
- Optional: instructions to rotate certificates

The bootstrap `kubeconfig` should be in a path available to the kubelet, for example `/var/lib/kubelet/bootstrap-kubeconfig`.

Its format is identical to a normal `kubeconfig` file. A sample file might look as follows:

```
apiVersion: v1
kind: Config
clusters:
- cluster:
  certificate-authority: /var/lib/kubernetes/ca.pem
  server: https://my.server.example.com:6443
  name: bootstrap
contexts:
- context:
  cluster: bootstrap
  user: kubelet-bootstrap
  name: bootstrap
current-context: bootstrap
preferences: {}
users:
- name: kubelet-bootstrap
  user:
    token: 07401b.f395accd246ae52d
```

The important elements to note are:

- `certificate-authority` : path to a CA file, used to validate the server certificate presented by kube-apiserver
- `server` : URL to kube-apiserver
- `token` : the token to use

The format of the token does not matter, as long as it matches what kube-apiserver expects. In the above example, we used a bootstrap token. As stated earlier, *any* valid authentication method can be used, not only tokens.

Because the bootstrap `kubeconfig` is a standard `kubeconfig`, you can use `kubectl` to generate it. To create the above example file:

```
kubectl config --kubeconfig=/var/lib/kubelet/bootstrap-kubeconfig set
kubectl config --kubeconfig=/var/lib/kubelet/bootstrap-kubeconfig set
kubectl config --kubeconfig=/var/lib/kubelet/bootstrap-kubeconfig set
kubectl config --kubeconfig=/var/lib/kubelet/bootstrap-kubeconfig use
```

To indicate to the kubelet to use the bootstrap `kubeconfig`, use the following kubelet flag:

```
--bootstrap-kubeconfig="/var/lib/kubelet/bootstrap-kubeconfig" --kube
```

When starting the kubelet, if the file specified via `--kubeconfig` does not exist, the bootstrap kubeconfig specified via `--bootstrap-kubeconfig` is used to request a client certificate from the API server. On approval of the certificate request and receipt back by the kubelet, a kubeconfig file referencing the generated key and obtained certificate is written to the path specified by `--kubeconfig`. The certificate and key file will be placed in the directory specified by `--cert-dir`.

## Client and serving certificates

All of the above relate to kubelet *client* certificates, specifically, the certificates a kubelet uses to authenticate to kube-apiserver.

A kubelet also can use *serving* certificates. The kubelet itself exposes an https endpoint for certain features. To secure these, the kubelet can do one of:

- use provided key and certificate, via the `--tls-private-key-file` and `--tls-cert-file` flags
- create self-signed key and certificate, if a key and certificate are not provided
- request serving certificates from the cluster server, via the CSR API

The client certificate provided by TLS bootstrapping is signed, by default, for `client auth` only, and thus cannot be used as serving certificates, or `server auth`.

However, you *can* enable its server certificate, at least partially, via certificate rotation.

## Certificate rotation

Kubernetes v1.8 and higher kubelet implements features for enabling rotation of its client and/or serving certificates. Note, rotation of serving certificate is a **beta** feature and requires the `RotateKubeletServerCertificate` feature flag on the kubelet (enabled by default).

You can configure the kubelet to rotate its client certificates by creating new CSRs as its existing credentials expire. To enable this feature, use the `rotateCertificates` field of [kubelet configuration file](#) or pass the following command line argument to the kubelet (deprecated):

```
--rotate-certificates
```

Enabling `RotateKubeletServerCertificate` causes the kubelet **both** to request a serving certificate after bootstrapping its client credentials **and** to rotate that certificate. To enable this behavior, use the field `serverTLSBootstrap` of the [kubelet configuration file](#) or pass the following command line argument to the kubelet (deprecated):

```
--rotate-server-certificates
```

### Note:

The CSR approving controllers implemented in core Kubernetes do not approve node *serving* certificates for [security reasons](#). To use `RotateKubeletServerCertificate` operators need to run a custom approving controller, or manually approve the serving certificate requests.

A deployment-specific approval process for kubelet serving certificates should typically only approve CSRs which:

1. are requested by nodes (ensure the `spec.username` field is of the form `system:node:<nodeName>` and `spec.groups` contains `system:nodes`)
2. request usages for a serving certificate (ensure `spec.usages` contains `server auth`, optionally contains `digital signature` and `key encipherment`, and contains no other usages)
3. only have IP and DNS `subjectAltNames` that belong to the requesting node, and have no URI and Email `subjectAltNames` (parse the x509 Certificate Signing Request in `spec.request` to verify `subjectAltNames`)

## Other authenticating components

All of TLS bootstrapping described in this document relates to the kubelet. However, other components may need to communicate directly with kube-apiserver. Notable is kube-proxy, which is part of the Kubernetes node components and runs on every node, but may also include other components such as monitoring or networking.

Like the kubelet, these other components also require a method of authenticating to kube-apiserver. You have several options for generating these credentials:

- The old way: Create and distribute certificates the same way you did for kubelet before TLS bootstrapping
- DaemonSet: Since the kubelet itself is loaded on each node, and is sufficient to start base services, you can run kube-proxy and other node-specific services not as a standalone process, but rather as a daemonset in the `kube-system` namespace. Since it will be in-cluster, you can give it a proper service account with appropriate permissions to perform its activities. This may be the simplest way to configure such services.

## kubectl approval

CSRs can be approved outside of the approval flows built into the controller manager.

The signing controller does not immediately sign all certificate requests. Instead, it waits until they have been flagged with an "Approved" status by an appropriately-privileged user. This flow is intended to allow for automated approval handled by an external approval controller or the approval controller implemented in the core controller-manager.

However cluster administrators can also manually approve certificate requests using kubectl. An administrator can list CSRs with `kubectl get csr` and describe one in detail with `kubectl describe csr <name>`. An administrator can approve or deny a CSR with `kubectl certificate approve <name>` and `kubectl certificate deny <name>`.

# 3.15 - Validating Admission Policy

ⓘ **FEATURE STATE:** Kubernetes v1.30 [stable]

This page provides an overview of Validating Admission Policy.

## What is Validating Admission Policy?

Validating admission policies offer a declarative, in-process alternative to validating admission webhooks.

Validating admission policies use the Common Expression Language (CEL) to declare the validation rules of a policy. Validation admission policies are highly configurable, enabling policy authors to define policies that can be parameterized and scoped to resources as needed by cluster administrators.

## What Resources Make a Policy

A policy is generally made up of three resources:

- The `ValidatingAdmissionPolicy` describes the abstract logic of a policy (think: "this policy makes sure a particular label is set to a particular value").
- A `ValidatingAdmissionPolicyBinding` links the above resources together and provides scoping. If you only want to require an `owner` label to be set for `Pods`, the binding is where you would specify this restriction.
- A parameter resource provides information to a `ValidatingAdmissionPolicy` to make it a concrete statement (think "the `owner` label must be set to something that ends in `.company.com`"). A native type such as `ConfigMap` or a CRD defines the schema of a parameter resource. `ValidatingAdmissionPolicy` objects specify what `Kind` they are expecting for their parameter resource.

At least a `ValidatingAdmissionPolicy` and a corresponding `ValidatingAdmissionPolicyBinding` must be defined for a policy to have an effect.

If a `ValidatingAdmissionPolicy` does not need to be configured via parameters, simply leave `spec.paramKind` in `ValidatingAdmissionPolicy` not specified.

## Getting Started with Validating Admission Policy

Validating Admission Policy is part of the cluster control-plane. You should write and deploy them with great caution. The following describes how to quickly experiment with Validating Admission Policy.

## Creating a ValidatingAdmissionPolicy

The following is an example of a ValidatingAdmissionPolicy.

[validatingadmissionpolicy/basic-example-policy.yaml](#) 

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "demo-policy.example.com"
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups: ["apps"]
        apiVersions: ["v1"]
        operations: ["CREATE", "UPDATE"]
        resources: ["deployments"]
  validations:
    - expression: "object.spec.replicas <= 5"
```

`spec.validations` contains CEL expressions which use the [Common Expression Language \(CEL\)](#) to validate the request. If an expression evaluates to false, the validation check is enforced according to the `spec.failurePolicy` field.

**Note:**

You can quickly test CEL expressions in [CEL Playground](#).

To configure a validating admission policy for use in a cluster, a binding is required. The following is an example of a ValidatingAdmissionPolicyBinding.:

[validatingadmissionpolicy/basic-example-binding.yaml](#) 

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
  name: "demo-binding-test.example.com"
spec:
  policyName: "demo-policy.example.com"
  validationActions: [Deny]
  matchResources:
    namespaceSelector:
      matchLabels:
        environment: test
```

When trying to create a deployment with replicas set not satisfying the validation expression, an error will return containing message:

```
ValidatingAdmissionPolicy 'demo-policy.example.com' with binding 'dem
```

The above provides a simple example of using ValidatingAdmissionPolicy

without a parameter configured.

## Validation actions

Each `ValidatingAdmissionPolicyBinding` must specify one or more `validationActions` to declare how `validations` of a policy are enforced.

The supported `validationActions` are:

- `Deny` : Validation failure results in a denied request.
- `Warn` : Validation failure is reported to the request client as a [warning](#).
- `Audit` : Validation failure is included in the audit event for the API request.

For example, to both warn clients about a validation failure and to audit the validation failures, use:

```
validationActions: [Warn, Audit]
```

`Deny` and `Warn` may not be used together since this combination needlessly duplicates the validation failure both in the API response body and the HTTP warning headers.

A `validation` that evaluates to false is always enforced according to these actions. Failures defined by the `failurePolicy` are enforced according to these actions only if the `failurePolicy` is set to `Fail` (or not specified), otherwise the failures are ignored.

See [Audit Annotations: validation failures](#) for more details about the validation failure audit annotation.

## Parameter resources

Parameter resources allow a policy configuration to be separate from its definition. A policy can define `paramKind`, which outlines GVK of the parameter resource, and then a policy binding ties a policy by name (via `policyName`) to a particular parameter resource via `paramRef`.

If parameter configuration is needed, the following is an example of a `ValidatingAdmissionPolicy` with parameter configuration.

[validatingadmissionpolicy/policy-with-param.yaml](#) 

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "replicalimit-policy.example.com"
spec:
  failurePolicy: Fail
  paramKind:
    apiVersion: rules.example.com/v1
    kind: ReplicaLimit
  matchConstraints:
    resourceRules:
      - apiGroups: ["apps"]
        apiVersions: ["v1"]
        operations: ["CREATE", "UPDATE"]
        resources: ["deployments"]
  validations:
    - expression: "object.spec.replicas <= params.maxReplicas"
      reason: Invalid
```

The `spec.paramKind` field of the `ValidatingAdmissionPolicy` specifies the kind of resources used to parameterize this policy. For this example, it is configured by `ReplicaLimit` custom resources. Note in this example how the CEL expression references the parameters via the CEL `params` variable, e.g. `params.maxReplicas`. `spec.matchConstraints` specifies what resources this policy is designed to validate. Note that the native types such like `ConfigMap` could also be used as parameter reference.

The `spec.validations` fields contain CEL expressions. If an expression evaluates to false, the validation check is enforced according to the `spec.failurePolicy` field.

The validating admission policy author is responsible for providing the `ReplicaLimit` parameter CRD.

To configure an validating admission policy for use in a cluster, a binding and parameter resource are created. The following is an example of a `ValidatingAdmissionPolicyBinding` that uses a **cluster-wide** param - the same param will be used to validate every resource request that matches the binding:

[validatingadmissionpolicy/binding-with-param.yaml](#) 

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
  name: "replicalimit-binding-test.example.com"
spec:
  policyName: "replicalimit-policy.example.com"
  validationActions: [Deny]
  paramRef:
    name: "replica-limit-test.example.com"
    namespace: "default"
  matchResources:
    namespaceSelector:
      matchLabels:
        environment: test
```

Notice this binding applies a parameter to the policy for all resources which are in the `test` environment.

The parameter resource could be as following:

[validatingadmissionpolicy/replicalimit-param.yaml](#) 

```
apiVersion: rules.example.com/v1
kind: ReplicaLimit
metadata:
  name: "replica-limit-test.example.com"
  namespace: "default"
maxReplicas: 3
```

This policy parameter resource limits deployments to a max of 3 replicas.

An admission policy may have multiple bindings. To bind all other environments to have a `maxReplicas` limit of 100, create another `ValidatingAdmissionPolicyBinding`:

[validatingadmissionpolicy/binding-with-param-prod.yaml](#) 

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
  name: "replicalimit-binding-nontest"
spec:
  policyName: "replicalimit-policy.example.com"
  validationActions: [Deny]
  paramRef:
    name: "replica-limit-prod.example.com"
    namespace: "default"
  matchResources:
    namespaceSelector:
      matchExpressions:
        - key: environment
          operator: NotIn
          values:
            - test
```

Notice this binding applies a different parameter to resources which are not in the `test` environment.

And have a parameter resource:

[validatingadmissionpolicy/replicalimit-param-prod.yaml](#) 

```
apiVersion: rules.example.com/v1
kind: ReplicaLimit
metadata:
  name: "replica-limit-prod.example.com"
maxReplicas: 100
```

For each admission request, the API server evaluates CEL expressions of each (policy, binding, param) combination that match the request. For a request to be admitted it must pass **all** evaluations.

If multiple bindings match the request, the policy will be evaluated for each, and they must all pass evaluation for the policy to be considered passed.

If multiple parameters match a single binding, the policy rules will be evaluated for each param, and they too must all pass for the binding to be considered passed. Bindings can have overlapping match criteria. The policy is evaluated for each matching binding-parameter combination. A policy may even be evaluated multiple times if multiple bindings match it, or a single binding that matches multiple parameters.

The `params` object representing a parameter resource will not be set if a parameter resource has not been bound, so for policies requiring a parameter resource, it can be useful to add a check to ensure one has been bound. A parameter resource will not be bound and `params` will be null if `paramKind` of the policy, or `paramRef` of the binding are not specified.

For the use cases require parameter configuration, we recommend to add a param check in `spec.validations[0].expression` :

```
- expression: "params != null"
  message: "params missing but required to bind to this policy"
```

## Optional parameters

It can be convenient to be able to have optional parameters as part of a parameter resource, and only validate them if present. CEL provides `has()`, which checks if the key passed to it exists. CEL also implements Boolean short-circuiting. If the first half of a logical OR evaluates to true, it won't evaluate the other half (since the result of the entire OR will be true regardless).

Combining the two, we can provide a way to validate optional parameters:

```
!has(params.optionalNumber) || (params.optionalNumber >= 5 &&
params.optionalNumber <= 10)
```

Here, we first check that the optional parameter is present with `!` `has(params.optionalNumber)` .

- If `optionalNumber` hasn't been defined, then the expression short-circuits since `!has(params.optionalNumber)` will evaluate to true.
- If `optionalNumber` has been defined, then the latter half of the CEL expression will be evaluated, and `optionalNumber` will be checked to ensure that it contains a value between 5 and 10 inclusive.

## Per-namespace Parameters

As the author of a `ValidatingAdmissionPolicy` and its `ValidatingAdmissionPolicyBinding`, you can choose to specify cluster-wide, or per-namespace parameters. If you specify a `namespace` for the binding's `paramRef`, the control plane only searches for parameters in that namespace.

However, if `namespace` is not specified in the `ValidatingAdmissionPolicyBinding`, the API server can search for relevant parameters in the namespace that a request is against. For example, if

you make a request to modify a ConfigMap in the `default` namespace and there is a relevant `ValidatingAdmissionPolicyBinding` with no `namespace` set, then the API server looks for a parameter object in `default`. This design enables policy configuration that depends on the namespace of the resource being manipulated, for more fine-tuned control.

## Parameter selector

In addition to specify a parameter in a binding by `name`, you may choose instead to specify label selector, such that all resources of the policy's `paramKind`, and the param's `namespace` (if applicable) that match the label selector are selected for evaluation. See [selector](#) for more information on how label selectors match resources.

If multiple parameters are found to meet the condition, the policy's rules are evaluated for each parameter found and the results will be ANDed together.

If `namespace` is provided, only objects of the `paramKind` in the provided namespace are eligible for selection. Otherwise, when `namespace` is empty and `paramKind` is namespace-scoped, the `namespace` used in the request being admitted will be used.

## Authorization checks

We introduced the authorization check for parameter resources. User is expected to have `read` access to the resources referenced by `paramKind` in `ValidatingAdmissionPolicy` and `paramRef` in `ValidatingAdmissionPolicyBinding`.

Note that if a resource in `paramKind` fails resolving via the restmapper, `read` access to all resources of groups is required.

## Failure Policy

`failurePolicy` defines how mis-configurations and CEL expressions evaluating to error from the admission policy are handled. Allowed values are `Ignore` or `Fail`.

- `Ignore` means that an error calling the `ValidatingAdmissionPolicy` is ignored and the API request is allowed to continue.
- `Fail` means that an error calling the `ValidatingAdmissionPolicy` causes the admission to fail and the API request to be rejected.

Note that the `failurePolicy` is defined inside `ValidatingAdmissionPolicy`:

[validatingadmissionpolicy/failure-policy-ignore.yaml](#) 

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
spec:
...
failurePolicy: Ignore # The default is "Fail"
validations:
- expression: "object.spec.xyz == params.x"
```

## Validation Expression

`spec.validations[i].expression` represents the expression which will be evaluated by CEL. To learn more, see the [CEL language specification](#) CEL expressions have access to the contents of the Admission request/response, organized into CEL variables as well as some other useful variables:

- 'object' - The object from the incoming request. The value is null for DELETE requests.
- 'oldObject' - The existing object. The value is null for CREATE requests.
- 'request' - Attributes of the [admission request](#).
- 'params' - Parameter resource referred to by the policy binding being evaluated. The value is null if `ParamKind` is not specified.
- `namespaceObject` - The namespace, as a Kubernetes resource, that the incoming object belongs to. The value is null if the incoming object is cluster-scoped.
- `authorizer` - A CEL Authorizer. May be used to perform authorization checks for the principal (authenticated user) of the request. See [AuthzSelectors](#) and [Authz](#) in the Kubernetes CEL library documentation for more details.
- `authorizer.requestResource` - A shortcut for an authorization check configured with the request resource (group, resource, (subresource), namespace, name).

The `apiVersion`, `kind`, `metadata.name` and `metadata.generateName` are always accessible from the root of the object. No other metadata properties are accessible.

Equality on arrays with list type of 'set' or 'map' ignores element order, i.e. `[1, 2] == [2, 1]`. Concatenation on arrays with x-kubernetes-list-type use the semantics of the list type:

- 'set': `x + y` performs a union where the array positions of all elements in `x` are preserved and non-intersecting elements in `y` are appended, retaining their partial order.
- 'map': `x + y` performs a merge where the array positions of all keys in `x` are preserved but the values are overwritten by values in `y` when the key sets of `x` and `y` intersect. Elements in `y` with non-intersecting keys are appended, retaining their partial order.

## Validation expression examples

| Expression                                                                                                | Purpose                                                                    |
|-----------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| <code>object.minReplicas &lt;= object.replicas &amp;&amp; object.replicas &lt;= object.maxReplicas</code> | Validate that the three fields defining replicas are ordered appropriately |
| <code>'Available' in object.stateCounts</code>                                                            | Validate that an entry with the 'Available' key exists in a map            |
| <code>(size(object.list1) == 0) != (size(object.list2) == 0)</code>                                       | Validate that one of two lists is non-empty, but not both                  |
| <code>!('MY_KEY' in object.map1)    object['MY_KEY'].matches('^[a-zA-Z]*\$')</code>                       | Validate the value of a map for a specific key, if it is in the map        |

| Expression                                                                             | Purpose                                                                                        |
|----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| object.envars.filter(e, e.name == 'MY_ENV').all(e, e.value.matches('^[a-zA-Z]*\$'))    | Validate the 'value' field of a listMap entry where key field 'name' is 'MY_ENV'               |
| has(object.expired) && object.created + object.ttl < object.expired                    | Validate that 'expired' date is after a 'create' date plus a 'ttl' duration                    |
| object.health.startsWith('ok')                                                         | Validate a 'health' string field has the prefix 'ok'                                           |
| object.widgets.exists(w, w.key == 'x' && w.foo < 10)                                   | Validate that the 'foo' property of a listMap item with a key 'x' is less than 10              |
| type(object) == string ? object == '100%' : object == 1000                             | Validate an int-or-string field for both the int and string cases                              |
| object.metadata.name.startsWith(object.prefix)                                         | Validate that an object's name has the prefix of another field value                           |
| object.set1.all(e, !(e in object.set2))                                                | Validate that two listSets are disjoint                                                        |
| size(object.names) == size(object.details) && object.names.all(n, n in object.details) | Validate the 'details' map is keyed by the items in the 'names' listSet                        |
| size(object.clusters.filter(c, c.name == object.primary)) == 1                         | Validate that the 'primary' property has one and only one occurrence in the 'clusters' listMap |

Read [Supported evaluation on CEL](#) for more information about CEL rules.

`spec.validation[i].reason` represents a machine-readable description of why this validation failed. If this is the first validation in the list to fail, this reason, as well as the corresponding HTTP response code, are used in the HTTP response to the client. The currently supported reasons are: `Unauthorized`, `Forbidden`, `Invalid`, `RequestEntityTooLarge`. If not set, `StatusReasonInvalid` is used in the response to the client.

## Matching requests: `matchConditions`

You can define *match conditions* for a `ValidatingAdmissionPolicy` if you need fine-grained request filtering. These conditions are useful if you find that match rules, `objectSelectors` and `namespaceSelectors` still doesn't provide the filtering you want. Match conditions are [CEL expressions](#). All match conditions must evaluate to true for the resource to be evaluated.

Here is an example illustrating a few different uses for match conditions:

[access/validating-admission-policy-match-conditions.yaml](#) 

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "demo-policy.example.com"
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups: ["*"]
        apiVersions: ["*"]
        operations: ["CREATE", "UPDATE"]
        resources: ["*"]
  matchConditions:
    - name: 'exclude-leases' # Each match condition must have a unique name
      expression: '!request.resource.group == "coordination.k8s.io"'
    - name: 'exclude-kubelet-requests'
      expression: '!("system:nodes" in request.userInfo.groups)' # Match conditions can use any valid CEL expression
    - name: 'rbac' # Skip RBAC requests.
      expression: 'request.resource.group != "rbac.authorization.k8s.io"'
  validations:
    - expression: '!object.metadata.name.contains("demo") || object.name == "demo"'
```

Match conditions have access to the same CEL variables as validation expressions.

In the event of an error evaluating a match condition the policy is not evaluated. Whether to reject the request is determined as follows:

1. If **any** match condition evaluated to `false` (regardless of other errors), the API server skips the policy.
2. Otherwise:
  - o for `failurePolicy: Fail`, reject the request (without evaluating the policy).
  - o for `failurePolicy: Ignore`, proceed with the request but skip the policy.

## Audit annotations

`auditAnnotations` may be used to include audit annotations in the audit event of the API request.

For example, here is an admission policy with an audit annotation:

[access/validating-admission-policy-audit-annotation.yaml](#) 

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "demo-policy.example.com"
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups: ["apps"]
        apiVersions: ["v1"]
        operations: ["CREATE", "UPDATE"]
        resources: ["deployments"]
  validations:
    - expression: "object.spec.replicas > 50"
      messageExpression: "'Deployment spec.replicas set to ' + string(object.spec.replicas)"
  auditAnnotations:
    - key: "high-replica-count"
      valueExpression: "'Deployment spec.replicas set to ' + string(object.spec.replicas)"
```

When an API request is validated with this admission policy, the resulting audit event will look like:

```
# the audit event recorded
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "annotations": {
    "demo-policy.example.com/high-replica-count": "Deployment spec.replicas set to 60"
    # other annotations
    ...
  }
  # other fields
  ...
}
```

In this example the annotation will only be included if the `spec.replicas` of the Deployment is more than 50, otherwise the CEL expression evaluates to null and the annotation will not be included.

Note that audit annotation keys are prefixed by the name of the `ValidatingAdmissionWebhook` and a `/`. If another admission controller, such as an admission webhook, uses the exact same audit annotation key, the value of the first admission controller to include the audit annotation will be included in the audit event and all other values will be ignored.

## Message expression

To return a more friendly message when the policy rejects a request, we can use a CEL expression to composite a message with `spec.validations[i].messageExpression`. Similar to the validation expression, a message expression has access to `object`, `oldObject`, `request`, `params`, and `namespaceObject`. Unlike validations, message expression must evaluate to a string.

For example, to better inform the user of the reason of denial when the policy refers to a parameter, we can have the following validation:

[access/deployment-replicas-policy.yaml](#) 

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "deploy-replica-policy.example.com"
spec:
  paramKind:
    apiVersion: rules.example.com/v1
    kind: ReplicaLimit
  matchConstraints:
    resourceRules:
      - apiGroups: ["apps"]
        apiVersions: ["v1"]
        operations: ["CREATE", "UPDATE"]
        resources: ["deployments"]
  validations:
    - expression: "object.spec.replicas <= params.maxReplicas"
      messageExpression: "'object.spec.replicas must be no greater than"
      reason: Invalid
```

After creating a params object that limits the replicas to 3 and setting up the binding, when we try to create a deployment with 5 replicas, we will receive the following message.

```
$ kubectl create deploy --image=nginx nginx --replicas=5
error: failed to create deployment: deployments.apps "nginx" is forbi
```

This is more informative than a static message of "too many replicas".

The message expression takes precedence over the static message defined in `spec.validations[i].message` if both are defined. However, if the message expression fails to evaluate, the static message will be used instead. Additionally, if the message expression evaluates to a multi-line string, the evaluation result will be discarded and the static message will be used if present. Note that static message is validated against multi-line strings.

## Type checking

When a policy definition is created or updated, the validation process parses the expressions it contains and reports any syntax errors, rejecting the definition if any errors are found. Afterward, the referred variables are checked for type errors, including missing fields and type confusion, against the matched types of `spec.matchConstraints`. The result of type checking can be retrieved from `status.typeChecking`. The presence of `status.typeChecking` indicates the completion of type checking, and an empty `status.typeChecking` means that no errors were detected.

For example, given the following policy definition:

[validatingadmissionpolicy/typechecking.yaml](#) 

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "deploy-replica-policy.example.com"
spec:
  matchConstraints:
    resourceRules:
      - apiGroups: ["apps"]
        apiVersions: ["v1"]
        operations: ["CREATE", "UPDATE"]
        resources: ["deployments"]
  validations:
    - expression: "object.replicas > 1" # should be "object.spec.replicas"
      message: "must be replicated"
      reason: Invalid
```

The status will yield the following information:

```
status:
  typeChecking:
    expressionWarnings:
      - fieldRef: spec.validations[0].expression
        warning: |-  
          apps/v1, Kind=Deployment: ERROR: <input>:1:7: undefined field  
          / object.replicas > 1  
          / .....^
```

If multiple resources are matched in `spec.matchConstraints` , all of matched resources will be checked against. For example, the following policy definition

[validatingadmissionpolicy/typechecking-multiple-match.yaml](#) 

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "replica-policy.example.com"
spec:
  matchConstraints:
    resourceRules:
      - apiGroups: ["apps"]
        apiVersions: ["v1"]
        operations: ["CREATE", "UPDATE"]
        resources: ["deployments", "replicasets"]
  validations:
    - expression: "object.replicas > 1" # should be "object.spec.replicas"
      message: "must be replicated"
      reason: Invalid
```

will have multiple types and type checking result of each type in the warning message.

```
status:
  typeChecking:
    expressionWarnings:
      - fieldRef: spec.validations[0].expression
        warning: |-
          apps/v1, Kind=Deployment: ERROR: <input>:1:7: undefined field
          | object.replicas > 1
          | .....
          apps/v1, Kind=ReplicaSet: ERROR: <input>:1:7: undefined field
          | object.replicas > 1
          | .....
```

Type Checking has the following limitation:

- No wildcard matching. If `spec.matchConstraints.resourceRules` contains `"*"` in any of `apiGroups`, `apiVersions` or `resources`, the types that `"*"` matches will not be checked.
- The number of matched types is limited to 10. This is to prevent a policy that manually specifying too many types. to consume excessive computing resources. In the order of ascending group, version, and then resource, 11th combination and beyond are ignored.
- Type Checking does not affect the policy behavior in any way. Even if the type checking detects errors, the policy will continue to evaluate. If errors do occur during evaluate, the failure policy will decide its outcome.
- Type Checking does not apply to CRDs, including matched CRD types and reference of `paramKind`. The support for CRDs will come in future release.

## Variable composition

If an expression grows too complicated, or part of the expression is reusable and computationally expensive to evaluate, you can extract some part of the expressions into variables. A variable is a named expression that can be referred later in `variables` in other expressions.

```
spec:
  variables:
    - name: foo
      expression: "'foo' in object.spec.metadata.labels ? object.spec
  validations:
    - expression: variables.foo == 'bar'
```

A variable is lazily evaluated when it is first referred. Any error that occurs during the evaluation will be reported during the evaluation of the referring expression. Both the result and potential error are memorized and count only once towards the runtime cost.

The order of variables are important because a variable can refer to other variables that are defined before it. This ordering prevents circular references.

The following is a more complex example of enforcing that image repository names match the environment defined in its namespace.

[access/image-matches-namespace-environment.policy.yaml](#) 

```
# This policy enforces that all containers of a deployment has the image
# Except for "exempt" deployments, or any containers that do not belong to
# For example, if the namespace has a label of {"environment": "staging"}, then
# or do not contain "example.com" at all, unless the deployment has {"image": "example.com"}
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "image-matches-namespace-environment.policy.example.com"
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups: ["apps"]
        apiVersions: ["v1"]
        operations: ["CREATE", "UPDATE"]
        resources: ["deployments"]
  variables:
    - name: environment
      expression: "'environment' in namespaceObject.metadata.labels ? namespaceObject.metadata.labels['environment'] : null"
    - name: exempt
      expression: "'exempt' in object.metadata.labels && object.metadata.labels['exempt'] === 'true'"
    - name: containers
      expression: "object.spec.template.spec.containers"
    - name: containersToCheck
      expression: "variables.containers.filter(c, c.image.contains('example.com'))"
  validations:
    - expression: "variables.exempt || variables.containersToCheck.all(c => c.image.startsWith(variables.environment))"
      messageExpression: "'only ' + variables.environment + ' images are allowed' + (variables.exempt ? '' : ' but exempt is true')"
```

With the policy bound to the namespace `default`, which is labeled `environment: prod`, the following attempt to create a deployment would be rejected.

```
kubectl create deploy --image=dev.example.com/nginx invalid
```

The error message is similar to this.

```
error: failed to create deployment: deployments.apps "invalid" is forbidden
```

# 4 - Well-Known Labels, Annotations and Taints

Kubernetes reserves all labels and annotations in the `kubernetes.io` and `k8s.io` namespaces.

This document serves both as a reference to the values and as a coordination point for assigning values.

## Labels, annotations and taints used on API objects

### `apf.kubernetes.io/autoupdate-spec`

Type: Annotation

Example: `apf.kubernetes.io/autoupdate-spec: "true"`

Used on: [FlowSchema and PriorityLevelConfiguration Objects](#)

If this annotation is set to true on a FlowSchema or PriorityLevelConfiguration, the `spec` for that object is managed by the kube-apiserver. If the API server does not recognize an APF object, and you annotate it for automatic update, the API server deletes the entire object. Otherwise, the API server does not manage the object spec. For more details, read [Maintenance of the Mandatory and Suggested Configuration Objects](#).

### `app.kubernetes.io/component`

Type: Label

Example: `app.kubernetes.io/component: "database"`

Used on: All Objects (typically used on [workload resources](#)).

The component within the application architecture.

One of the [recommended labels](#).

### `app.kubernetes.io/created-by` (deprecated)

Type: Label

Example: `app.kubernetes.io/created-by: "controller-manager"`

Used on: All Objects (typically used on [workload resources](#)).

The controller/user who created this resource.

**Note:**

Starting from v1.9, this label is deprecated.

### `app.kubernetes.io/instance`

Type: Label

Example: `app.kubernetes.io/instance: "mysql-abcxyz"`

Used on: All Objects (typically used on [workload resources](#)).

A unique name identifying the instance of an application. To assign a

non-unique name, use [app.kubernetes.io/name](#).

One of the [recommended labels](#).

## app.kubernetes.io/managed-by

Type: Label

Example: app.kubernetes.io/managed-by: "helm"

Used on: All Objects (typically used on [workload resources](#)).

The tool being used to manage the operation of an application.

One of the [recommended labels](#).

## app.kubernetes.io/name

Type: Label

Example: app.kubernetes.io/name: "mysql"

Used on: All Objects (typically used on [workload resources](#)).

The name of the application.

One of the [recommended labels](#).

## app.kubernetes.io/part-of

Type: Label

Example: app.kubernetes.io/part-of: "wordpress"

Used on: All Objects (typically used on [workload resources](#)).

The name of a higher-level application this object is part of.

One of the [recommended labels](#).

## app.kubernetes.io/version

Type: Label

Example: app.kubernetes.io/version: "5.7.21"

Used on: All Objects (typically used on [workload resources](#)).

The current version of the application.

Common forms of values include:

- [semantic version](#)
- the Git [revision hash](#) for the source code.

One of the [recommended labels](#).

## applyset.kubernetes.io/additional-namespaces (alpha)

Type: Annotation

Example: applyset.kubernetes.io/additional-namespaces:

"namespace1,namespace2"

Used on: Objects being used as ApplySet parents.

Use of this annotation is Alpha. For Kubernetes version 1.31, you can use this annotation on Secrets, ConfigMaps, or custom resources if the

CustomResourceDefinition defining them has the `applyset.kubernetes.io/is-parent-type` label.

Part of the specification used to implement [ApplySet-based pruning in kubectl](#). This annotation is applied to the parent object used to track an ApplySet to extend the scope of the ApplySet beyond the parent object's own namespace (if any). The value is a comma-separated list of the names of namespaces other than the parent's namespace in which objects are found.

## applyset.kubernetes.io/contains-group-kinds (alpha)

Type: Annotation

Example: `applyset.kubernetes.io/contains-group-kinds: "certificates.cert-manager.io,configmaps,deployments.apps,secrets,services"`

Used on: Objects being used as ApplySet parents.

Use of this annotation is Alpha. For Kubernetes version 1.31, you can use this annotation on Secrets, ConfigMaps, or custom resources if the CustomResourceDefinition defining them has the `applyset.kubernetes.io/is-parent-type` label.

Part of the specification used to implement [ApplySet-based pruning in kubectl](#). This annotation is applied to the parent object used to track an ApplySet to optimize listing of ApplySet member objects. It is optional in the ApplySet specification, as tools can perform discovery or use a different optimization. However, as of Kubernetes version 1.31, it is required by kubectl. When present, the value of this annotation must be a comma separated list of the group-kinds, in the fully-qualified name format, i.e. `<resource>.<group>`.

## applyset.kubernetes.io/contains-group-resources (deprecated)

Type: Annotation

Example: `applyset.kubernetes.io/contains-group-resources: "certificates.cert-manager.io,configmaps,deployments.apps,secrets,services"`

Used on: Objects being used as ApplySet parents.

For Kubernetes version 1.31, you can use this annotation on Secrets, ConfigMaps, or custom resources if the CustomResourceDefinition defining them has the `applyset.kubernetes.io/is-parent-type` label.

Part of the specification used to implement [ApplySet-based pruning in kubectl](#). This annotation is applied to the parent object used to track an ApplySet to optimize listing of ApplySet member objects. It is optional in the ApplySet specification, as tools can perform discovery or use a different optimization. However, in Kubernetes version 1.31, it is required by kubectl. When present, the value of this annotation must be a comma separated list of the group-kinds, in the fully-qualified name format, i.e. `<resource>.<group>`.

### Note:

This annotation is currently deprecated and replaced by [applyset.kubernetes.io/contains-group-kinds](#), support for this will be removed in applyset beta or GA.

## applyset.kubernetes.io/id (alpha)

Type: Label

Example: `applyset.kubernetes.io/id:`

`"applyset-0eFHV8ySqp7XoShsGvyWFQD3s96yqwHmzc4e0HR1dsY-v1"`

Used on: Objects being used as ApplySet parents.

Use of this label is Alpha. For Kubernetes version 1.31, you can use this label on Secrets, ConfigMaps, or custom resources if the CustomResourceDefinition defining them has the `applyset.kubernetes.io/is-parent-type` label.

Part of the specification used to implement [ApplySet-based pruning in kubectl](#). This label is what makes an object an ApplySet parent object. Its value is the unique ID of the ApplySet, which is derived from the identity of the parent object itself. This ID **must** be the base64 encoding (using the URL safe encoding of RFC4648) of the hash of the group-kind-name-namespace of the object it is on, in the form:

`<base64(sha256(<name>.<namespace>.<kind>.<group>))>` . There is no relation between the value of this label and object UID.

## applyset.kubernetes.io/is-parent-type (alpha)

Type: Label

Example: `applyset.kubernetes.io/is-parent-type: "true"`

Used on: Custom Resource Definition (CRD)

Use of this label is Alpha. Part of the specification used to implement [ApplySet-based pruning in kubectl](#). You can set this label on a CustomResourceDefinition (CRD) to identify the custom resource type it defines (not the CRD itself) as an allowed parent for an ApplySet. The only permitted value for this label is `"true"` ; if you want to mark a CRD as not being a valid parent for ApplySets, omit this label.

## applyset.kubernetes.io/part-of (alpha)

Type: Label

Example: `applyset.kubernetes.io/part-of:`

`"applyset-0eFHV8ySqp7XoShsGvyWFQD3s96yqwHmzc4e0HR1dsY-v1"`

Used on: All objects.

Use of this label is Alpha. Part of the specification used to implement [ApplySet-based pruning in kubectl](#). This label is what makes an object a member of an ApplySet. The value of the label **must** match the value of the `applyset.kubernetes.io/id` label on the parent object.

## applyset.kubernetes.io/tooling (alpha)

Type: Annotation

Example: `applyset.kubernetes.io/tooling: "kubectl/v1.31"`

Used on: Objects being used as ApplySet parents.

Use of this annotation is Alpha. For Kubernetes version 1.31, you can use this annotation on Secrets, ConfigMaps, or custom resources if the CustomResourceDefinition defining them has the `applyset.kubernetes.io/is-parent-type` label.

Part of the specification used to implement [ApplySet-based pruning in](#)

[kubectl](#). This annotation is applied to the parent object used to track an ApplySet to indicate which tooling manages that ApplySet. Tooling should refuse to mutate ApplySets belonging to other tools. The value must be in the format `<toolname>/<semver>`.

## apps.kubernetes.io/pod-index (beta)

Type: Label

Example: `apps.kubernetes.io/pod-index: "0"`

Used on: Pod

When a StatefulSet controller creates a Pod for the StatefulSet, it sets this label on that Pod. The value of the label is the ordinal index of the pod being created.

See [Pod Index Label](#) in the StatefulSet topic for more details. Note the [PodIndexLabel](#) feature gate must be enabled for this label to be added to pods.

## cluster-autoscaler.kubernetes.io/safe-to-evict

Type: Annotation

Example: `cluster-autoscaler.kubernetes.io/safe-to-evict: "true"`

Used on: Pod

When this annotation is set to `"true"`, the cluster autoscaler is allowed to evict a Pod even if other rules would normally prevent that. The cluster autoscaler never evicts Pods that have this annotation explicitly set to `"false"`; you could set that on an important Pod that you want to keep running. If this annotation is not set then the cluster autoscaler follows its Pod-level behavior.

## config.kubernetes.io/local-config

Type: Annotation

Example: `config.kubernetes.io/local-config: "true"`

Used on: All objects

This annotation is used in manifests to mark an object as local configuration that should not be submitted to the Kubernetes API.

A value of `"true"` for this annotation declares that the object is only consumed by client-side tooling and should not be submitted to the API server.

A value of `"false"` can be used to declare that the object should be submitted to the API server even when it would otherwise be assumed to be local.

This annotation is part of the Kubernetes Resource Model (KRM) Functions Specification, which is used by Kustomize and similar third-party tools. For example, Kustomize removes objects with this annotation from its final build output.

## container.apparmor.security.beta.kubernetes.io/\* (deprecated)

Type: Annotation

Example: `container.apparmor.security.beta.kubernetes.io/my-container:my-custom-profile`

Used on: Pods

This annotation allows you to specify the AppArmor security profile for a container within a Kubernetes pod. As of Kubernetes v1.30, this should be set with the `appArmorProfile` field instead. To learn more, see the [AppArmor](#) tutorial. The tutorial illustrates using AppArmor to restrict a container's abilities and access.

The profile specified dictates the set of rules and restrictions that the containerized process must adhere to. This helps enforce security policies and isolation for your containers.

## internal.config.kubernetes.io/\* (reserved prefix)

Type: Annotation

Used on: All objects

This prefix is reserved for internal use by tools that act as orchestrators in accordance with the Kubernetes Resource Model (KRM) Functions Specification. Annotations with this prefix are internal to the orchestration process and are not persisted to the manifests on the filesystem. In other words, the orchestrator tool should set these annotations when reading files from the local filesystem and remove them when writing the output of functions back to the filesystem.

A KRM function **must not** modify annotations with this prefix, unless otherwise specified for a given annotation. This enables orchestrator tools to add additional internal annotations, without requiring changes to existing functions.

## internal.config.kubernetes.io/path

Type: Annotation

Example: `internal.config.kubernetes.io/path: "relative/file/path.yaml"`

Used on: All objects

This annotation records the slash-delimited, OS-agnostic, relative path to the manifest file the object was loaded from. The path is relative to a fixed location on the filesystem, determined by the orchestrator tool.

This annotation is part of the Kubernetes Resource Model (KRM) Functions Specification, which is used by Kustomize and similar third-party tools.

A KRM Function **should not** modify this annotation on input objects unless it is modifying the referenced files. A KRM Function **may** include this annotation on objects it generates.

## internal.config.kubernetes.io/index

Type: Annotation

Example: `internal.config.kubernetes.io/index: "2"`

Used on: All objects

This annotation records the zero-indexed position of the YAML document that contains the object within the manifest file the object was loaded from. Note that YAML documents are separated by three dashes ( `---` ) and can each contain one object. When this annotation is not specified, a

value of 0 is implied.

This annotation is part of the Kubernetes Resource Model (KRM) Functions Specification, which is used by Kustomize and similar third-party tools.

A KRM Function **should not** modify this annotation on input objects unless it is modifying the referenced files. A KRM Function **may** include this annotation on objects it generates.

## kubernetes.io/arch

Type: Label

Example: `kubernetes.io/arch: "amd64"`

Used on: Node

The Kubelet populates this with `runtime.GOARCH` as defined by Go. This can be handy if you are mixing ARM and x86 nodes.

## kubernetes.io/os

Type: Label

Example: `kubernetes.io/os: "linux"`

Used on: Node, Pod

For nodes, the kubelet populates this with `runtime.GOOS` as defined by Go. This can be handy if you are mixing operating systems in your cluster (for example: mixing Linux and Windows nodes).

You can also set this label on a Pod. Kubernetes allows you to set any value for this label; if you use this label, you should nevertheless set it to the Go `runtime.GOOS` string for the operating system that this Pod actually works with.

When the `kubernetes.io/os` label value for a Pod does not match the label value on a Node, the kubelet on the node will not admit the Pod. However, this is not taken into account by the kube-scheduler. Alternatively, the kubelet refuses to run a Pod where you have specified a Pod OS, if this isn't the same as the operating system for the node where that kubelet is running. Just look for [Pods OS](#) for more details.

## kubernetes.io/metadata.name

Type: Label

Example: `kubernetes.io/metadata.name: "mynamespace"`

Used on: Namespaces

The Kubernetes API server (part of the [control plane](#)) sets this label on all namespaces. The label value is set to the name of the namespace. You can't change this label's value.

This is useful if you want to target a specific namespace with a label selector.

## kubernetes.io/limit-ranger

Type: Annotation

Example: `kubernetes.io/limit-ranger: "LimitRanger plugin set: cpu, memory request for container nginx; cpu, memory limit for container`

nginx"

Used on: Pod

Kubernetes by default doesn't provide any resource limit, that means unless you explicitly define limits, your container can consume unlimited CPU and memory. You can define a default request or default limit for pods. You do this by creating a LimitRange in the relevant namespace. Pods deployed after you define a LimitRange will have these limits applied to them. The annotation `kubernetes.io/limit-ranger` records that resource defaults were specified for the Pod, and they were applied successfully. For more details, read about [LimitRanges](#).

## kubernetes.io/config.hash

Type: Annotation

Example: `kubernetes.io/config.hash: "df7cc47f8477b6b1226d7d23a904867b"`

Used on: Pod

When the kubelet creates a static Pod based on a given manifest, it attaches this annotation to the static Pod. The value of the annotation is the UID of the Pod. Note that the kubelet also sets the `.spec.nodeName` to the current node name as if the Pod was scheduled to the node.

## kubernetes.io/config.mirror

Type: Annotation

Example: `kubernetes.io/config.mirror: "df7cc47f8477b6b1226d7d23a904867b"`

Used on: Pod

For a static Pod created by the kubelet on a node, a mirror Pod is created on the API server. The kubelet adds an annotation to indicate that this Pod is actually a mirror Pod. The annotation value is copied from the [kubernetes.io/config.hash](#) annotation, which is the UID of the Pod.

When updating a Pod with this annotation set, the annotation cannot be changed or removed. If a Pod doesn't have this annotation, it cannot be added during a Pod update.

## kubernetes.io/config.source

Type: Annotation

Example: `kubernetes.io/config.source: "file"`

Used on: Pod

This annotation is added by the kubelet to indicate where the Pod comes from. For static Pods, the annotation value could be one of `file` or `http` depending on where the Pod manifest is located. For a Pod created on the API server and then scheduled to the current node, the annotation value is `api`.

## kubernetes.io/config.seen

Type: Annotation

Example: `kubernetes.io/config.seen: "2023-10-27T04:04:56.011314488Z"`

Used on: Pod

When the kubelet sees a Pod for the first time, it may add this annotation to the Pod with a value of current timestamp in the RFC3339 format.

## addonmanager.kubernetes.io/mode

Type: Label

Example: `addonmanager.kubernetes.io/mode: "Reconcile"`

Used on: All objects

To specify how an add-on should be managed, you can use the `addonmanager.kubernetes.io/mode` label. This label can have one of three values: `Reconcile`, `EnsureExists`, or `Ignore`.

- `Reconcile` : Addon resources will be periodically reconciled with the expected state. If there are any differences, the add-on manager will recreate, reconfigure or delete the resources as needed. This is the default mode if no label is specified.
- `EnsureExists` : Addon resources will be checked for existence only but will not be modified after creation. The add-on manager will create or re-create the resources when there is no instance of the resource with that name.
- `Ignore` : Addon resources will be ignored. This mode is useful for add-ons that are not compatible with the add-on manager or that are managed by another controller.

For more details, see [Addon-manager](#).

## beta.kubernetes.io/arch (deprecated)

Type: Label

This label has been deprecated. Please use [kubernetes.io/arch](#) instead.

## beta.kubernetes.io/os (deprecated)

Type: Label

This label has been deprecated. Please use [kubernetes.io/os](#) instead.

## kube-aggregator.kubernetes.io/automanaged

Type: Label

Example: `kube-aggregator.kubernetes.io/automanaged: "onstart"`

Used on: APIService

The `kube-apiserver` sets this label on any APIService object that the API server has created automatically. The label marks how the control plane should manage that APIService. You should not add, modify, or remove this label by yourself.

### Note:

Automanaged APIService objects are deleted by kube-apiserver when it has no built-in or custom resource API corresponding to the API group/version of the APIService.

There are two possible values:

- `onstart` : The APIService should be reconciled when an API server starts up, but not otherwise.

- **true** : The API server should reconcile this APIService continuously.

## service.alpha.kubernetes.io/tolerate-unready-endpoints (deprecated)

Type: Annotation

Used on: StatefulSet

This annotation on a Service denotes if the Endpoints controller should go ahead and create Endpoints for unready Pods. Endpoints of these Services retain their DNS records and continue receiving traffic for the Service from the moment the kubelet starts all containers in the pod and marks it *Running*, til the kubelet stops all containers and deletes the pod from the API server.

## kubernetes.io/hostname

Type: Label

Example: `kubernetes.io/hostname: "ip-172-20-114-199.ec2.internal"`

Used on: Node

The Kubelet populates this label with the hostname of the node. Note that the hostname can be changed from the "actual" hostname by passing the `--hostname-override` flag to the `kubelet`.

This label is also used as part of the topology hierarchy. See [topology.kubernetes.io/zone](#) for more information.

## kubernetes.io/change-cause

Type: Annotation

Example: `kubernetes.io/change-cause: "kubectl edit --record deployment foo"`

Used on: All Objects

This annotation is a best guess at why something was changed.

It is populated when adding `--record` to a `kubectl` command that may change an object.

## kubernetes.io/description

Type: Annotation

Example: `kubernetes.io/description: "Description of K8s object."`

Used on: All Objects

This annotation is used for describing specific behaviour of given object.

## kubernetes.io/enforce-mountable-secrets

Type: Annotation

Example: `kubernetes.io/enforce-mountable-secrets: "true"`

Used on: ServiceAccount

The value for this annotation must be **true** to take effect. When you set this annotation to "true", Kubernetes enforces the following rules for Pods running as this ServiceAccount:

1. Secrets mounted as volumes must be listed in the ServiceAccount's `secrets` field.
2. Secrets referenced in `envFrom` for containers (including sidecar containers and init containers) must also be listed in the ServiceAccount's `secrets` field. If any container in a Pod references a Secret not listed in the ServiceAccount's `secrets` field (and even if the reference is marked as `optional`), then the Pod will fail to start, and an error indicating the non-compliant secret reference will be generated.
3. Secrets referenced in a Pod's `imagePullSecrets` must be present in the ServiceAccount's `imagePullSecrets` field, the Pod will fail to start, and an error indicating the non-compliant image pull secret reference will be generated.

When you create or update a Pod, these rules are checked. If a Pod doesn't follow them, it won't start and you'll see an error message. If a Pod is already running and you change the `kubernetes.io/enforce-mountable-secrets` annotation to true, or you edit the associated ServiceAccount to remove the reference to a Secret that the Pod is already using, the Pod continues to run.

## node.kubernetes.io/exclude-from-external-load-balancers

Type: Label

Example: `node.kubernetes.io/exclude-from-external-load-balancers`

Used on: Node

You can add labels to particular worker nodes to exclude them from the list of backend servers used by external load balancers. The following command can be used to exclude a worker node from the list of backend servers in a backend set:

```
kubectl label nodes <node-name> node.kubernetes.io/exclude-from-exter
```

## controller.kubernetes.io/pod-deletion-cost

Type: Annotation

Example: `controller.kubernetes.io/pod-deletion-cost: "10"`

Used on: Pod

This annotation is used to set [Pod Deletion Cost](#) which allows users to influence ReplicaSet downscaling order. The annotation value parses into an `int32` type.

## cluster-autoscaler.kubernetes.io/enable-ds-eviction

Type: Annotation

Example: `cluster-autoscaler.kubernetes.io/enable-ds-eviction: "true"`

Used on: Pod

This annotation controls whether a DaemonSet pod should be evicted by a ClusterAutoscaler. This annotation needs to be specified on DaemonSet pods in a DaemonSet manifest. When this annotation is set to `"true"`, the ClusterAutoscaler is allowed to evict a DaemonSet Pod, even if other rules would normally prevent that. To disallow the ClusterAutoscaler from evicting DaemonSet pods, you can set this annotation to `"false"`

for important DaemonSet pods. If this annotation is not set, then the ClusterAutoscaler follows its overall behavior (i.e evict the DaemonSets based on its configuration).

**Note:**

This annotation only impacts DaemonSet Pods.

## kubernetes.io/ingress-bandwidth

Type: Annotation

Example: `kubernetes.io/ingress-bandwidth: 10M`

Used on: Pod

You can apply quality-of-service traffic shaping to a pod and effectively limit its available bandwidth. Ingress traffic to a Pod is handled by shaping queued packets to effectively handle data. To limit the bandwidth on a Pod, write an object definition JSON file and specify the data traffic speed using `kubernetes.io/ingress-bandwidth` annotation. The unit used for specifying ingress rate is bits per second, as a [Quantity](#). For example, `10M` means 10 megabits per second.

**Note:**

Ingress traffic shaping annotation is an experimental feature. If you want to enable traffic shaping support, you must add the `bandwidth` plugin to your CNI configuration file (default `/etc/cni/net.d`) and ensure that the binary is included in your CNI bin dir (default `/opt/cni/bin`).

## kubernetes.io/egress-bandwidth

Type: Annotation

Example: `kubernetes.io/egress-bandwidth: 10M`

Used on: Pod

Egress traffic from a Pod is handled by policing, which simply drops packets in excess of the configured rate. The limits you place on a Pod do not affect the bandwidth of other Pods. To limit the bandwidth on a Pod, write an object definition JSON file and specify the data traffic speed using `kubernetes.io/egress-bandwidth` annotation. The unit used for specifying egress rate is bits per second, as a [Quantity](#). For example, `10M` means 10 megabits per second.

**Note:**

Egress traffic shaping annotation is an experimental feature. If you want to enable traffic shaping support, you must add the `bandwidth` plugin to your CNI configuration file (default `/etc/cni/net.d`) and ensure that the binary is included in your CNI bin dir (default `/opt/cni/bin`).

## beta.kubernetes.io/instance-type (deprecated)

Type: Label

**Note:**

Starting in v1.17, this label is deprecated in favor of

## [node.kubernetes.io/instance-type](https://kubernetes.io/docs/reference/labels-annotations-selectors/label-selectors/#node.kubernetes.io/instance-type).

### node.kubernetes.io/instance-type

Type: Label

Example: `node.kubernetes.io/instance-type: "m3.medium"`

Used on: Node

The Kubelet populates this with the instance type as defined by the cloud provider. This will be set only if you are using a cloud provider. This setting is handy if you want to target certain workloads to certain instance types, but typically you want to rely on the Kubernetes scheduler to perform resource-based scheduling. You should aim to schedule based on properties rather than on instance types (for example: require a GPU, instead of requiring a `g2.2xlarge` ).

### failure-domain.beta.kubernetes.io/region (deprecated)

Type: Label

#### **Note:**

Starting in v1.17, this label is deprecated in favor of [topology.kubernetes.io/region](https://kubernetes.io/docs/reference/labels-annotations-selectors/label-selectors/#topology.kubernetes.io/region).

### failure-domain.beta.kubernetes.io/zone (deprecated)

Type: Label

#### **Note:**

Starting in v1.17, this label is deprecated in favor of [topology.kubernetes.io/zone](https://kubernetes.io/docs/reference/labels-annotations-selectors/label-selectors/#topology.kubernetes.io/zone).

### pv.kubernetes.io/bind-completed

Type: Annotation

Example: `pv.kubernetes.io/bind-completed: "yes"`

Used on: PersistentVolumeClaim

When this annotation is set on a PersistentVolumeClaim (PVC), that indicates that the lifecycle of the PVC has passed through initial binding setup. When present, that information changes how the control plane interprets the state of PVC objects. The value of this annotation does not matter to Kubernetes.

### pv.kubernetes.io/bound-by-controller

Type: Annotation

Example: `pv.kubernetes.io/bound-by-controller: "yes"`

Used on: PersistentVolume, PersistentVolumeClaim

If this annotation is set on a PersistentVolume or PersistentVolumeClaim, it indicates that a storage binding (PersistentVolume → PersistentVolumeClaim, or PersistentVolumeClaim → PersistentVolume) was installed by the controller. If the annotation isn't set, and there is a storage binding in place, the absence of that annotation means that the

binding was done manually. The value of this annotation does not matter.

## pv.kubernetes.io/provisioned-by

Type: Annotation

Example: `pv.kubernetes.io/provisioned-by: "kubernetes.io/rbd"`

Used on: PersistentVolume

This annotation is added to a PersistentVolume(PV) that has been dynamically provisioned by Kubernetes. Its value is the name of volume plugin that created the volume. It serves both users (to show where a PV comes from) and Kubernetes (to recognize dynamically provisioned PVs in its decisions).

## pv.kubernetes.io/migrated-to

Type: Annotation

Example: `pv.kubernetes.io/migrated-to: pd.csi.storage.gke.io`

Used on: PersistentVolume, PersistentVolumeClaim

It is added to a PersistentVolume(PV) and PersistentVolumeClaim(PVC) that is supposed to be dynamically provisioned/deleted by its corresponding CSI driver through the `csIMigration` feature gate. When this annotation is set, the Kubernetes components will "stand-down" and the `external-provisioner` will act on the objects.

## statefulset.kubernetes.io/pod-name

Type: Label

Example: `statefulset.kubernetes.io/pod-name: "mystatefulset-7"`

Used on: Pod

When a StatefulSet controller creates a Pod for the StatefulSet, the control plane sets this label on that Pod. The value of the label is the name of the Pod being created.

See [Pod Name Label](#) in the StatefulSet topic for more details.

## scheduler.alpha.kubernetes.io/node-selector

Type: Annotation

Example: `scheduler.alpha.kubernetes.io/node-selector: "name-of-node-selector"`

Used on: Namespace

The [PodNodeSelector](#) uses this annotation key to assign node selectors to pods in namespaces.

## topology.kubernetes.io/region

Type: Label

Example: `topology.kubernetes.io/region: "us-east-1"`

Used on: Node, PersistentVolume

See [topology.kubernetes.io/zone](#).

## topology.kubernetes.io/zone

Type: Label

Example: `topology.kubernetes.io/zone: "us-east-1c"`

Used on: Node, PersistentVolume

**On Node:** The `kubelet` or the external `cloud-controller-manager` populates this with the information from the cloud provider. This will be set only if you are using a cloud provider. However, you can consider setting this on nodes if it makes sense in your topology.

**On PersistentVolume:** topology-aware volume provisoners will automatically set node affinity constraints on a `PersistentVolume`.

A zone represents a logical failure domain. It is common for Kubernetes clusters to span multiple zones for increased availability. While the exact definition of a zone is left to infrastructure implementations, common properties of a zone include very low network latency within a zone, no-cost network traffic within a zone, and failure independence from other zones. For example, nodes within a zone might share a network switch, but nodes in different zones should not.

A region represents a larger domain, made up of one or more zones. It is uncommon for Kubernetes clusters to span multiple regions. While the exact definition of a zone or region is left to infrastructure implementations, common properties of a region include higher network latency between them than within them, non-zero cost for network traffic between them, and failure independence from other zones or regions. For example, nodes within a region might share power infrastructure (e.g. a UPS or generator), but nodes in different regions typically would not.

Kubernetes makes a few assumptions about the structure of zones and regions:

1. regions and zones are hierarchical: zones are strict subsets of regions and no zone can be in 2 regions
2. zone names are unique across regions; for example region "africa-east-1" might be comprised of zones "africa-east-1a" and "africa-east-1b"

It should be safe to assume that topology labels do not change. Even though labels are strictly mutable, consumers of them can assume that a given node is not going to be moved between zones without being destroyed and recreated.

Kubernetes can use this information in various ways. For example, the scheduler automatically tries to spread the Pods in a `ReplicaSet` across nodes in a single-zone cluster (to reduce the impact of node failures, see [kubernetes.io/hostname](#)). With multiple-zone clusters, this spreading behavior also applies to zones (to reduce the impact of zone failures). This is achieved via `SelectorSpreadPriority`.

`SelectorSpreadPriority` is a best effort placement. If the zones in your cluster are heterogeneous (for example: different numbers of nodes, different types of nodes, or different pod resource requirements), this placement might prevent equal spreading of your Pods across zones. If desired, you can use homogeneous zones (same number and types of nodes) to reduce the probability of unequal spreading.

The scheduler (through the `VolumeZonePredicate` predicate) also will ensure that Pods, that claim a given volume, are only placed into the same zone as that volume. Volumes cannot be attached across zones.

If `PersistentVolumeLabel` does not support automatic labeling of your PersistentVolumes, you should consider adding the labels manually (or adding support for `PersistentVolumeLabel`). With `PersistentVolumeLabel`, the scheduler prevents Pods from mounting volumes in a different zone. If your infrastructure doesn't have this constraint, you don't need to add the zone labels to the volumes at all.

## volume.beta.kubernetes.io/storage-provisioner (deprecated)

Type: Annotation

Example: `volume.beta.kubernetes.io/storage-provisioner: "k8s.io/minikube-hostpath"`

Used on: PersistentVolumeClaim

This annotation has been deprecated since v1.23. See [volume.kubernetes.io/storage-provisioner](#).

## volume.beta.kubernetes.io/storage-class (deprecated)

Type: Annotation

Example: `volume.beta.kubernetes.io/storage-class: "example-class"`

Used on: PersistentVolume, PersistentVolumeClaim

This annotation can be used for PersistentVolume(PV) or PersistentVolumeClaim(PVC) to specify the name of [StorageClass](#). When both the `storageClassName` attribute and the `volume.beta.kubernetes.io/storage-class` annotation are specified, the annotation `volume.beta.kubernetes.io/storage-class` takes precedence over the `storageClassName` attribute.

This annotation has been deprecated. Instead, set the [storageClassName field](#) for the PersistentVolumeClaim or PersistentVolume.

## volume.beta.kubernetes.io/mount-options (deprecated)

Type: Annotation

Example: `volume.beta.kubernetes.io/mount-options: "ro,soft"`

Used on: PersistentVolume

A Kubernetes administrator can specify additional [mount options](#) for when a PersistentVolume is mounted on a node.

## volume.kubernetes.io/storage-provisioner

Type: Annotation

Used on: PersistentVolumeClaim

This annotation is added to a PVC that is supposed to be dynamically provisioned. Its value is the name of a volume plugin that is supposed to provision a volume for this PVC.

## volume.kubernetes.io/selected-node

Type: Annotation

Used on: PersistentVolumeClaim

This annotation is added to a PVC that is triggered by a scheduler to be dynamically provisioned. Its value is the name of the selected node.

## `volumes.kubernetes.io/controller-managed-attach-detach`

Type: Annotation

Used on: Node

If a node has the annotation `volumes.kubernetes.io/controller-managed-attach-detach`, its storage attach and detach operations are being managed by the [volume attach/detach controller](#).

The value of the annotation isn't important.

## `node.kubernetes.io/windows-build`

Type: Label

Example: `node.kubernetes.io/windows-build: "10.0.17763"`

Used on: Node

When the kubelet is running on Microsoft Windows, it automatically labels its Node to record the version of Windows Server in use.

The label's value is in the format "MajorVersion.MinorVersion.BuildNumber".

## `storage.alpha.kubernetes.io/migrated-plugins`

Type: Annotation

Example: `storage.alpha.kubernetes.io/migrated-plugins: "kubernetes.io/cinder"`

Used on: CSINode (an extension API)

This annotation is automatically added for the CSINode object that maps to a node that installs CSIDriver. This annotation shows the in-tree plugin name of the migrated plugin. Its value depends on your cluster's in-tree cloud provider storage type.

For example, if the in-tree cloud provider storage type is `CSIMigrationvSphere`, the CSINodes instance for the node should be updated with: `storage.alpha.kubernetes.io/migrated-plugins: "kubernetes.io/vsphere-volume"`

## `service.kubernetes.io/headless`

Type: Label

Example: `service.kubernetes.io/headless: ""`

Used on: Endpoints

The control plane adds this label to an Endpoints object when the owning Service is headless. To learn more, read [Headless Services](#).

## `service.kubernetes.io/topology-aware-hints` (deprecated)

Example: `service.kubernetes.io/topology-aware-hints: "Auto"`

Used on: Service

This annotation was used for enabling *topology aware hints* on Services.

Topology aware hints have since been renamed: the concept is now called [topology aware routing](#). Setting the annotation to `Auto`, on a Service, configured the Kubernetes control plane to add topology hints on EndpointSlices associated with that Service. You can also explicitly set the annotation to `Disabled`.

If you are running a version of Kubernetes older than 1.31, check the documentation for that Kubernetes version to see how topology aware routing works in that release.

There are no other valid values for this annotation. If you don't want topology aware hints for a Service, don't add this annotation.

## service.kubernetes.io/topology-mode

Type: Annotation

Example: `service.kubernetes.io/topology-mode: Auto`

Used on: Service

This annotation provides a way to define how Services handle network topology; for example, you can configure a Service so that Kubernetes prefers keeping traffic between a client and server within a single topology zone. In some cases this can help reduce costs or improve network performance.

See [Topology Aware Routing](#) for more details.

## kubernetes.io/service-name

Type: Label

Example: `kubernetes.io/service-name: "my-website"`

Used on: EndpointSlice

Kubernetes associates [EndpointSlices](#) with [Services](#) using this label.

This label records the name of the Service that the EndpointSlice is backing. All EndpointSlices should have this label set to the name of their associated Service.

## kubernetes.io/service-account.name

Type: Annotation

Example: `kubernetes.io/service-account.name: "sa-name"`

Used on: Secret

This annotation records the name of the ServiceAccount that the token (stored in the Secret of type `kubernetes.io/service-account-token`) represents.

## kubernetes.io/service-account.uid

Type: Annotation

Example: `kubernetes.io/service-account.uid: da68f9c6-9d26-11e7-b84e-002dc52800da`

Used on: Secret

This annotation records the unique ID of the ServiceAccount that the token (stored in the Secret of type `kubernetes.io/service-account-token`)

represents.

## kubernetes.io/legacy-token-last-used

Type: Label

Example: `kubernetes.io/legacy-token-last-used: 2022-10-24`

Used on: Secret

The control plane only adds this label to Secrets that have the type `kubernetes.io/service-account-token`. The value of this label records the date (ISO 8601 format, UTC time zone) when the control plane last saw a request where the client authenticated using the service account token.

If a legacy token was last used before the cluster gained the feature (added in Kubernetes v1.26), then the label isn't set.

## kubernetes.io/legacy-token-invalid-since

Type: Label

Example: `kubernetes.io/legacy-token-invalid-since: 2023-10-27`

Used on: Secret

The control plane automatically adds this label to auto-generated Secrets that have the type `kubernetes.io/service-account-token`. This label marks the Secret-based token as invalid for authentication. The value of this label records the date (ISO 8601 format, UTC time zone) when the control plane detects that the auto-generated Secret has not been used for a specified duration (defaults to one year).

## endpointslice.kubernetes.io/managed-by

Type: Label

Example: `endpointslice.kubernetes.io/managed-by: endpointslice-controller.k8s.io`

Used on: EndpointSlices

The label is used to indicate the controller or entity that manages the EndpointSlice. This label aims to enable different EndpointSlice objects to be managed by different controllers or entities within the same cluster.

## endpointslice.kubernetes.io/skip-mirror

Type: Label

Example: `endpointslice.kubernetes.io/skip-mirror: "true"`

Used on: Endpoints

The label can be set to `"true"` on an Endpoints resource to indicate that the EndpointSliceMirroring controller should not mirror this resource with EndpointSlices.

## service.kubernetes.io/service-proxy-name

Type: Label

Example: `service.kubernetes.io/service-proxy-name: "foo-bar"`

Used on: Service

The kube-proxy has this label for custom proxy, which delegates service control to custom proxy.

## experimental.windows.kubernetes.io/isolation-type (deprecated)

Type: Annotation

Example: `experimental.windows.kubernetes.io/isolation-type: "hyperv"`

Used on: Pod

The annotation is used to run Windows containers with Hyper-V isolation.

**Note:**

Starting from v1.20, this annotation is deprecated. Experimental Hyper-V support was removed in 1.21.

## ingressclass.kubernetes.io/is-default-class

Type: Annotation

Example: `ingressclass.kubernetes.io/is-default-class: "true"`

Used on: IngressClass

When a IngressClass resource has this annotation set to "true", new Ingress resource without a class specified will be assigned this default class.

## nginx.ingress.kubernetes.io/configuration-snippet

Type: Annotation

Example: `nginx.ingress.kubernetes.io/configuration-snippet: "more_set_headers \"Request-Id: $req_id\";\\nmore_set_headers \"Example: 42\";\\n"`

Used on: Ingress

You can use this annotation to set extra configuration on an Ingress that uses the [NGINX Ingress Controller](#). The `configuration-snippet` annotation is ignored by default since version 1.9.0 of the ingress controller. The NGINX ingress controller setting `allow-snippet-annotations` has to be explicitly enabled to use this annotation. Enabling the annotation can be dangerous in a multi-tenant cluster, as it can lead people with otherwise limited permissions being able to retrieve all Secrets in the cluster.

## kubernetes.io/ingress.class (deprecated)

Type: Annotation

Used on: Ingress

**Note:**

Starting in v1.18, this annotation is deprecated in favor of `spec.ingressClassName`.

## storageclass.kubernetes.io/is-default-class

Type: Annotation

Example: `storageclass.kubernetes.io/is-default-class: "true"`

Used on: StorageClass

When a single StorageClass resource has this annotation set to "true", new PersistentVolumeClaim resource without a class specified will be assigned this default class.

## alpha.kubernetes.io/provided-node-ip (alpha)

Type: Annotation

Example: `alpha.kubernetes.io/provided-node-ip: "10.0.0.1"`

Used on: Node

The kubelet can set this annotation on a Node to denote its configured IPv4 and/or IPv6 address.

When kubelet is started with the `--cloud-provider` flag set to any value (includes both external and legacy in-tree cloud providers), it sets this annotation on the Node to denote an IP address set from the command line flag (`--node-ip`). This IP is verified with the cloud provider as valid by the cloud-controller-manager.

## batch.kubernetes.io/job-completion-index

Type: Annotation, Label

Example: `batch.kubernetes.io/job-completion-index: "3"`

Used on: Pod

The Job controller in the kube-controller-manager sets this as a label and annotation for Pods created with Indexed [completion mode](#).

Note the [PodIndexLabel](#) feature gate must be enabled for this to be added as a pod **label**, otherwise it will just be an annotation.

## batch.kubernetes.io/cronjob-scheduled-timestamp

Type: Annotation

Example: `batch.kubernetes.io/cronjob-scheduled-timestamp: "2016-05-19T03:00:00-07:00"`

Used on: Jobs and Pods controlled by CronJobs

This annotation is used to record the original (expected) creation timestamp for a Job, when that Job is part of a CronJob. The control plane sets the value to that timestamp in RFC3339 format. If the Job belongs to a CronJob with a timezone specified, then the timestamp is in that timezone. Otherwise, the timestamp is in controller-manager's local time.

## kubectl.kubernetes.io/default-container

Type: Annotation

Example: `kubectl.kubernetes.io/default-container: "front-end-app"`

The value of the annotation is the container name that is default for this Pod. For example, `kubectl logs` or `kubectl exec` without `-c` or `--container` flag will use this default container.

## kubectl.kubernetes.io/default-logs-container (deprecated)

Type: Annotation

Example: `kubectl.kubernetes.io/default-logs-container: "front-end-app"`

The value of the annotation is the container name that is the default logging container for this Pod. For example, `kubectl logs` without `-c` or `--container` flag will use this default container.

**Note:**

This annotation is deprecated. You should use the [kubectl.kubernetes.io/default-container](#) annotation instead. Kubernetes versions 1.25 and newer ignore this annotation.

## kubectl.kubernetes.io/last-applied-configuration

Type: Annotation

Example: *see following snippet*

```
kubectl.kubernetes.io/last-applied-configuration: >
  {"apiVersion": "apps/v1", "kind": "Deployment", "metadata": {"annotation
```

Used on: all objects

The `kubectl` command line tool uses this annotation as a legacy mechanism to track changes. That mechanism has been superseded by [Server-side apply](#).

## kubectl.kubernetes.io/restartedAt

Type: Annotation

Example: `kubectl.kubernetes.io/restartedAt: "2024-06-21T17:27:41Z"`

Used on: Deployment, ReplicaSet, StatefulSet, DaemonSet, Pod

This annotation contains the latest restart time of a resource (Deployment, ReplicaSet, StatefulSet or DaemonSet), where `kubectl` triggered a rollout in order to force creation of new Pods. The command `kubectl rollout restart <RESOURCE>` triggers a restart by patching the template metadata of all the pods of resource with this annotation. In above example the latest restart time is shown as 21st June 2024 at 17:27:41 UTC.

You should not assume that this annotation represents the date / time of the most recent update; a separate change could have been made since the last manually triggered rollout.

If you manually set this annotation on a Pod, nothing happens. The restarting side effect comes from how workload management and Pod templating works.

## endpoints.kubernetes.io/over-capacity

Type: Annotation

Example: `endpoints.kubernetes.io/over-capacity:truncated`

Used on: Endpoints

The control plane adds this annotation to an [Endpoints](#) object if the associated [Service](#) has more than 1000 backing endpoints. The annotation indicates that the Endpoints object is over capacity and the number of endpoints has been truncated to 1000.

If the number of backend endpoints falls below 1000, the control plane removes this annotation.

## endpoints.kubernetes.io/last-change-trigger-time

Type: Annotation

Example: `endpoints.kubernetes.io/last-change-trigger-time: "2023-07-20T04:45:21Z"`

Used on: Endpoints

This annotation set to an [Endpoints](#) object that represents the timestamp (The timestamp is stored in RFC 3339 date-time string format. For example, '2018-10-22T19:32:52.1Z'). This is timestamp of the last change in some Pod or Service object, that triggered the change to the Endpoints object.

## control-plane.alpha.kubernetes.io/leader (deprecated)

Type: Annotation

Example: `control-plane.alpha.kubernetes.io/leader={"holderIdentity":"controller-0","leaseDurationSeconds":15,"acquireTime":"2023-01-19T13:12:57Z","renewTime":"2023-01-19T13:13:54Z","leaderTransitions":1}`

Used on: Endpoints

The control plane previously set annotation on an [Endpoints](#) object. This annotation provided the following detail:

- Who is the current leader.
- The time when the current leadership was acquired.
- The duration of the lease (of the leadership) in seconds.
- The time the current lease (the current leadership) should be renewed.
- The number of leadership transitions that happened in the past.

Kubernetes now uses [Leases](#) to manage leader assignment for the Kubernetes control plane.

## batch.kubernetes.io/job-tracking (deprecated)

Type: Annotation

Example: `batch.kubernetes.io/job-tracking: ""`

Used on: Jobs

The presence of this annotation on a Job used to indicate that the control plane is [tracking the Job status using finalizers](#). Adding or removing this annotation no longer has an effect (Kubernetes v1.27 and later) All Jobs are tracked with finalizers.

## job-name (deprecated)

Type: Label

Example: `job-name: "pi"`

Used on: Jobs and Pods controlled by Jobs

**Note:**

Starting from Kubernetes 1.27, this label is deprecated. Kubernetes 1.27 and newer ignore this label and use the prefixed `job-name` label.

## controller-uid (deprecated)

Type: Label

Example: `controller-uid: "$UID"`

Used on: Jobs and Pods controlled by Jobs

**Note:**

Starting from Kubernetes 1.27, this label is deprecated. Kubernetes 1.27 and newer ignore this label and use the prefixed `controller-uid` label.

## batch.kubernetes.io/job-name

Type: Label

Example: `batch.kubernetes.io/job-name: "pi"`

Used on: Jobs and Pods controlled by Jobs

This label is used as a user-friendly way to get Pods corresponding to a Job. The `job-name` comes from the `name` of the Job and allows for an easy way to get Pods corresponding to the Job.

## batch.kubernetes.io/controller-uid

Type: Label

Example: `batch.kubernetes.io/controller-uid: "$UID"`

Used on: Jobs and Pods controlled by Jobs

This label is used as a programmatic way to get all Pods corresponding to a Job.

The `controller-uid` is a unique identifier that gets set in the `selector` field so the Job controller can get all the corresponding Pods.

## scheduler.alpha.kubernetes.io/defaultTolerations

Type: Annotation

Example: `scheduler.alpha.kubernetes.io/defaultTolerations:`

```
'[{"operator": "Equal", "value": "value1", "effect": "NoSchedule",  
"key": "dedicated-node"}]'
```

Used on: Namespace

This annotation requires the [PodTolerationRestriction](#) admission controller to be enabled. This annotation key allows assigning tolerations to a namespace and any new pods created in this namespace would get these tolerations added.

## scheduler.alpha.kubernetes.io/tolerationsWhitelist

Type: Annotation

Example: `scheduler.alpha.kubernetes.io/tolerationsWhitelist: '[{"operator": "Exists", "effect": "NoSchedule", "key": "dedicated-node"}]'`

Used on: Namespace

This annotation is only useful when the (Alpha) [PodTolerationRestriction](#) admission controller is enabled. The annotation value is a JSON document that defines a list of allowed tolerations for the namespace it annotates. When you create a Pod or modify its tolerations, the API server checks the tolerations to see if they are mentioned in the allow list. The pod is admitted only if the check succeeds.

## `scheduler.alpha.kubernetes.io/preferAvoidPods` (deprecated)

Type: Annotation

Used on: Node

This annotation requires the [NodePreferAvoidPods scheduling plugin](#) to be enabled. The plugin is deprecated since Kubernetes 1.22. Use [Taints and Toleration](#)s instead.

## `node.kubernetes.io/not-ready`

Type: Taint

Example: `node.kubernetes.io/not-ready: "NoExecute"`

Used on: Node

The Node controller detects whether a Node is ready by monitoring its health and adds or removes this taint accordingly.

## `node.kubernetes.io/unreachable`

Type: Taint

Example: `node.kubernetes.io/unreachable: "NoExecute"`

Used on: Node

The Node controller adds the taint to a Node corresponding to the [NodeCondition](#) Ready being Unknown .

## `node.kubernetes.io/unschedulable`

Type: Taint

Example: `node.kubernetes.io/unschedulable: "NoSchedule"`

Used on: Node

The taint will be added to a node when initializing the node to avoid race condition.

## `node.kubernetes.io/memory-pressure`

Type: Taint

Example: `node.kubernetes.io/memory-pressure: "NoSchedule"`

Used on: Node

The kubelet detects memory pressure based on `memory.available` and `allocatableMemory.available` observed on a Node. The observed values are then compared to the corresponding thresholds that can be set on the kubelet to determine if the Node condition and taint should be added/removed.

## node.kubernetes.io/disk-pressure

Type: Taint

Example: `node.kubernetes.io/disk-pressure : "NoSchedule"`

Used on: Node

The kubelet detects disk pressure based on `imagefs.available`, `imagefs.inodesFree`, `nodefs.available` and `nodefs.inodesFree` (Linux only) observed on a Node. The observed values are then compared to the corresponding thresholds that can be set on the kubelet to determine if the Node condition and taint should be added/removed.

## node.kubernetes.io/network-unavailable

Type: Taint

Example: `node.kubernetes.io/network-unavailable: "NoSchedule"`

Used on: Node

This is initially set by the kubelet when the cloud provider used indicates a requirement for additional network configuration. Only when the route on the cloud is configured properly will the taint be removed by the cloud provider.

## node.kubernetes.io/pid-pressure

Type: Taint

Example: `node.kubernetes.io/pid-pressure: "NoSchedule"`

Used on: Node

The kubelet checks D-value of the size of `/proc/sys/kernel/pid_max` and the PIDs consumed by Kubernetes on a node to get the number of available PIDs that referred to as the `pid.available` metric. The metric is then compared to the corresponding threshold that can be set on the kubelet to determine if the node condition and taint should be added/removed.

## node.kubernetes.io/out-of-service

Type: Taint

Example: `node.kubernetes.io/out-of-service:NoExecute`

Used on: Node

A user can manually add the taint to a Node marking it out-of-service. If the `NodeOutOfServiceVolumeDetach` [feature gate](#) is enabled on `kube-controller-manager`, and a Node is marked out-of-service with this taint, the Pods on the node will be forcefully deleted if there are no matching tolerations on it and volume detach operations for the Pods terminating on the node will happen immediately. This allows the Pods on the out-of-service node to recover quickly on a different node.

**Caution:**

Refer to [Non-graceful node shutdown](#) for further details about when and how to use this taint.

## node.cloudprovider.kubernetes.io/uninitialized

Type: Taint

Example: `node.cloudprovider.kubernetes.io/uninitialized: "NoSchedule"`

Used on: Node

Sets this taint on a Node to mark it as unusable, when kubelet is started with the "external" cloud provider, until a controller from the cloud-controller-manager initializes this Node, and then removes the taint.

## node.cloudprovider.kubernetes.io/shutdown

Type: Taint

Example: `node.cloudprovider.kubernetes.io/shutdown: "NoSchedule"`

Used on: Node

If a Node is in a cloud provider specified shutdown state, the Node gets tainted accordingly with `node.cloudprovider.kubernetes.io/shutdown` and the taint effect of `NoSchedule`.

## feature.node.kubernetes.io/\*

Type: Label

Example: `feature.node.kubernetes.io/network-sriov.capable: "true"`

Used on: Node

These labels are used by the Node Feature Discovery (NFD) component to advertise features on a node. All built-in labels use the `feature.node.kubernetes.io` label namespace and have the format `feature.node.kubernetes.io/<feature-name>: "true"`. NFD has many extension points for creating vendor and application-specific labels. For details, see the [customization guide](#).

## nfd.node.kubernetes.io/master.version

Type: Annotation

Example: `nfd.node.kubernetes.io/master.version: "v0.6.0"`

Used on: Node

For node(s) where the Node Feature Discovery (NFD) [master](#) is scheduled, this annotation records the version of the NFD master. It is used for informative use only.

## nfd.node.kubernetes.io/worker.version

Type: Annotation

Example: `nfd.node.kubernetes.io/worker.version: "v0.4.0"`

Used on: Nodes

This annotation records the version for a Node Feature Discovery's [worker](#) if there is one running on a node. It's used for informative use

only.

## nfd.node.kubernetes.io/feature-labels

Type: Annotation

Example: `nfd.node.kubernetes.io/feature-labels: "cpu-cpuid.ADX,cpu-cpuid.AESNI,cpu-hardware_multithreading,kernel-version.full"`

Used on: Nodes

This annotation records a comma-separated list of node feature labels managed by [Node Feature Discovery](#) (NFD). NFD uses this for an internal mechanism. You should not edit this annotation yourself.

## nfd.node.kubernetes.io/extended-resources

Type: Annotation

Example: `nfd.node.kubernetes.io/extended-resources: "accelerator.acme.example/q500,example.com/coprocessor-fx5"`

Used on: Nodes

This annotation records a comma-separated list of [extended resources](#) managed by [Node Feature Discovery](#) (NFD). NFD uses this for an internal mechanism. You should not edit this annotation yourself.

## nfd.node.kubernetes.io/node-name

Type: Label

Example: `nfd.node.kubernetes.io/node-name: node-1`

Used on: Nodes

It specifies which node the NodeFeature object is targeting. Creators of NodeFeature objects must set this label and consumers of the objects are supposed to use the label for filtering features designated for a certain node.

**Note:**

These Node Feature Discovery (NFD) labels or annotations only apply to the nodes where NFD is running. To learn more about NFD and its components go to its official [documentation](#).

## service.beta.kubernetes.io/aws-load-balancer-access-log-emit-interval (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-access-log-emit-interval: "5"`

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures the load balancer for a Service based on this annotation. The value determines how often the load balancer writes log entries. For example, if you set the value to 5, the log writes occur 5 seconds apart.

## service.beta.kubernetes.io/aws-load-balancer-access-log-enabled (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-access-log-`

```
enabled: "false"
```

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures the load balancer for a Service based on this annotation. Access logging is enabled if you set the annotation to "true".

## service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-name (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-access-log-enabled: example`

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures the load balancer for a Service based on this annotation. The load balancer writes logs to an S3 bucket with the name you specify.

## service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-prefix (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-access-log-enabled: "/example"`

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures the load balancer for a Service based on this annotation. The load balancer writes log objects with the prefix that you specify.

## service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags: "Environment=demo,Project=example"`

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures tags (an AWS concept) for a load balancer based on the comma-separated key/value pairs in the value of this annotation.

## service.beta.kubernetes.io/aws-load-balancer-alpn-policy (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-alpn-policy: HTTP2Optional`

Used on: Service

The [AWS load balancer controller](#) uses this annotation. See [annotations](#) in the AWS load balancer controller documentation.

## service.beta.kubernetes.io/aws-load-balancer-attributes (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-attributes: "deletion_protection.enabled=true"`

Used on: Service

The [AWS load balancer controller](#) uses this annotation. See [annotations](#)

in the AWS load balancer controller documentation.

## service.beta.kubernetes.io/aws-load-balancer-backend-protocol (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp`

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures the load balancer listener based on the value of this annotation.

## service.beta.kubernetes.io/aws-load-balancer-connection-draining-enabled (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-connection-draining-enabled: "false"`

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures the load balancer based on this annotation. The load balancer's connection draining setting depends on the value you set.

## service.beta.kubernetes.io/aws-load-balancer-connection-draining-timeout (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-connection-draining-timeout: "60"`

Used on: Service

If you configure [connection draining](#) for a Service of `type: LoadBalancer`, and you use the AWS cloud, the integration configures the draining period based on this annotation. The value you set determines the draining timeout in seconds.

## service.beta.kubernetes.io/aws-load-balancer-ip-address-type (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-ip-address-type: ipv4`

Used on: Service

The [AWS load balancer controller](#) uses this annotation. See [annotations](#) in the AWS load balancer controller documentation.

## service.beta.kubernetes.io/aws-load-balancer-connection-idle-timeout (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-connection-idle-timeout: "60"`

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures a load balancer based on this annotation. The load balancer has a configured idle timeout period (in seconds) that applies to its connections. If no data has been sent or received by the time that the idle timeout period elapses, the load balancer closes the connection.

## service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing-enabled (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing-enabled: "true"`

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures a load balancer based on this annotation. If you set this annotation to "true", each load balancer node distributes requests evenly across the registered targets in all enabled [availability zones](#). If you disable cross-zone load balancing, each load balancer node distributes requests evenly across the registered targets in its availability zone only.

## service.beta.kubernetes.io/aws-load-balancer-eip-allocations (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-eip-allocations: "eipalloc-01bcdef23bcdef456,eipalloc-def1234abc4567890"`

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures a load balancer based on this annotation. The value is a comma-separated list of elastic IP address allocation IDs.

This annotation is only relevant for Services of `type: LoadBalancer`, where the load balancer is an AWS Network Load Balancer.

## service.beta.kubernetes.io/aws-load-balancer-extra-security-groups (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-extra-security-groups: "sg-12abcd3456,sg-34dcba6543"`

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures a load balancer based on this annotation. The annotation value is a comma-separated list of extra AWS VPC security groups to configure for the load balancer.

## service.beta.kubernetes.io/aws-load-balancer-healthcheck-healthy-threshold (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-healthcheck-healthy-threshold: "3"`

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures a load balancer based on this annotation. The annotation value specifies the number of successive successful health checks required for a backend to be considered healthy for traffic.

## service.beta.kubernetes.io/aws-load-balancer-healthcheck-interval (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-healthcheck-interval: "30"`

#### Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures a load balancer based on this annotation. The annotation value specifies the interval, in seconds, between health check probes made by the load balancer.

### service.beta.kubernetes.io/aws-load-balancer-healthcheck-path (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-healthcheck-path: /healthcheck`

#### Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures a load balancer based on this annotation. The annotation value determines the path part of the URL that is used for HTTP health checks.

### service.beta.kubernetes.io/aws-load-balancer-healthcheck-port (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-healthcheck-port: "24"`

#### Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures a load balancer based on this annotation. The annotation value determines which port the load balancer connects to when performing health checks.

### service.beta.kubernetes.io/aws-load-balancer-healthcheck-protocol (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-healthcheck-protocol: TCP`

#### Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures a load balancer based on this annotation. The annotation value determines how the load balancer checks the health of backend targets.

### service.beta.kubernetes.io/aws-load-balancer-healthcheck-timeout (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-healthcheck-timeout: "3"`

#### Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures a load balancer based on this annotation. The annotation value specifies the number of seconds before a probe that hasn't yet succeeded is automatically treated as having failed.

### service.beta.kubernetes.io/aws-load-balancer-healthcheck-unhealthy-threshold (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-healthcheck-`

```
unhealthy-threshold: "3"
```

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures a load balancer based on this annotation. The annotation value specifies the number of successive unsuccessful health checks required for a backend to be considered unhealthy for traffic.

## service.beta.kubernetes.io/aws-load-balancer-internal (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-internal: "true"`

Used on: Service

The cloud controller manager integration with AWS elastic load balancing configures a load balancer based on this annotation. When you set this annotation to "true", the integration configures an internal load balancer.

If you use the [AWS load balancer controller](#), see [service.beta.kubernetes.io/aws-load-balancer-scheme](#) .

## service.beta.kubernetes.io/aws-load-balancer-manage-backend-security-group-rules (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-manage-backend-security-group-rules: "true"`

Used on: Service

The [AWS load balancer controller](#) uses this annotation. See [annotations](#) in the AWS load balancer controller documentation.

## service.beta.kubernetes.io/aws-load-balancer-name (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-name: my-elb`

Used on: Service

If you set this annotation on a Service, and you also annotate that Service with `service.beta.kubernetes.io/aws-load-balancer-type: "external"` , and you use the [AWS load balancer controller](#) in your cluster, then the AWS load balancer controller sets the name of that load balancer to the value you set for *this* annotation.

See [annotations](#) in the AWS load balancer controller documentation.

## service.beta.kubernetes.io/aws-load-balancer-nlb-target-type (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "true"`

Used on: Service

The [AWS load balancer controller](#) uses this annotation. See [annotations](#) in the AWS load balancer controller documentation.

## service.beta.kubernetes.io/aws-load-balancer-private-ipv4-addresses (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-private-ipv4-addresses: "198.51.100.0,198.51.100.64"`

Used on: Service

The [AWS load balancer controller](#) uses this annotation. See [annotations](#) in the AWS load balancer controller documentation.

## service.beta.kubernetes.io/aws-load-balancer-proxy-protocol (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-proxy-protocol:`

`""`

Used on: Service

The official Kubernetes integration with AWS elastic load balancing configures a load balancer based on this annotation. The only permitted value is `"+"`, which indicates that the load balancer should wrap TCP connections to the backend Pod with the PROXY protocol.

## service.beta.kubernetes.io/aws-load-balancer-scheme (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-scheme: internal`

Used on: Service

The [AWS load balancer controller](#) uses this annotation. See [annotations](#) in the AWS load balancer controller documentation.

## service.beta.kubernetes.io/aws-load-balancer-security-groups (deprecated)

Example: `service.beta.kubernetes.io/aws-load-balancer-security-groups:`  
`"sg-53fae93f,sg-8725gr62r"`

Used on: Service

The AWS load balancer controller uses this annotation to specify a comma separated list of security groups you want to attach to an AWS load balancer. Both name and ID of security are supported where name matches a `Name` tag, not the `groupName` attribute.

When this annotation is added to a Service, the load-balancer controller attaches the security groups referenced by the annotation to the load balancer. If you omit this annotation, the AWS load balancer controller automatically creates a new security group and attaches it to the load balancer.

### Note:

Kubernetes v1.27 and later do not directly set or read this annotation. However, the AWS load balancer controller (part of the Kubernetes project) does still use the `service.beta.kubernetes.io/aws-load-balancer-security-groups` annotation.

## service.beta.kubernetes.io/load-balancer-source-ranges (deprecated)

Example: `service.beta.kubernetes.io/load-balancer-source-ranges:`  
`"192.0.2.0/25"`

Used on: Service

The [AWS load balancer controller](#) uses this annotation. You should set

.spec.loadBalancerSourceRanges for the Service instead.

## service.beta.kubernetes.io/aws-load-balancer-ssl-cert (beta)

Example: service.beta.kubernetes.io/aws-load-balancer-ssl-cert:

```
"arn:aws:acm:us-  
east-1:123456789012:certificate/12345678-1234-1234-1234-123456789012"
```

Used on: Service

The official integration with AWS elastic load balancing configures TLS for a Service of `type: LoadBalancer` based on this annotation. The value of the annotation is the AWS Resource Name (ARN) of the X.509 certificate that the load balancer listener should use.

(The TLS protocol is based on an older technology that abbreviates to SSL.)

## service.beta.kubernetes.io/aws-load-balancer-ssl-negotiation-policy (beta)

Example: service.beta.kubernetes.io/aws-load-balancer-ssl-negotiation-policy: ELBSecurityPolicy-TLS-1-2-2017-01

The official integration with AWS elastic load balancing configures TLS for a Service of `type: LoadBalancer` based on this annotation. The value of the annotation is the name of an AWS policy for negotiating TLS with a client peer.

## service.beta.kubernetes.io/aws-load-balancer-ssl-ports (beta)

Example: service.beta.kubernetes.io/aws-load-balancer-ssl-ports: "\*"

The official integration with AWS elastic load balancing configures TLS for a Service of `type: LoadBalancer` based on this annotation. The value of the annotation is either "\*", which means that all the load balancer's ports should use TLS, or it is a comma separated list of port numbers.

## service.beta.kubernetes.io/aws-load-balancer-subnets (beta)

Example: service.beta.kubernetes.io/aws-load-balancer-subnets:  
"private-a,private-b"

Kubernetes' official integration with AWS uses this annotation to configure a load balancer and determine in which AWS availability zones to deploy the managed load balancing service. The value is either a comma separated list of subnet names, or a comma separated list of subnet IDs.

## service.beta.kubernetes.io/aws-load-balancer-target-group-attributes (beta)

Example: service.beta.kubernetes.io/aws-load-balancer-target-group-attributes: "stickiness.enabled=true,stickiness.type=source\_ip"

Used on: Service

The [AWS load balancer controller](#) uses this annotation. See [annotations](#) in the AWS load balancer controller documentation.

## service.beta.kubernetes.io/aws-load-balancer-target-node-labels (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-target-node-labels: "kubernetes.io/oss=Linux,topology.kubernetes.io/region=us-east-2"`

Kubernetes' official integration with AWS uses this annotation to determine which nodes in your cluster should be considered as valid targets for the load balancer.

## service.beta.kubernetes.io/aws-load-balancer-type (beta)

Example: `service.beta.kubernetes.io/aws-load-balancer-type: external`

Kubernetes' official integrations with AWS use this annotation to determine whether the AWS cloud provider integration should manage a Service of `type: LoadBalancer`.

There are two permitted values:

### **nlb**

the cloud controller manager configures a Network Load Balancer

### **external**

the cloud controller manager does not configure any load balancer

If you deploy a Service of `type: LoadBalancer` on AWS, and you don't set any `service.beta.kubernetes.io/aws-load-balancer-type` annotation, the AWS integration deploys a classic Elastic Load Balancer. This behavior, with no annotation present, is the default unless you specify otherwise.

When you set this annotation to `external` on a Service of `type: LoadBalancer`, and your cluster has a working deployment of the AWS Load Balancer controller, then the AWS Load Balancer controller attempts to deploy a load balancer based on the Service specification.

### **Caution:**

Do not modify or add the `service.beta.kubernetes.io/aws-load-balancer-type` annotation on an existing Service object. See the AWS documentation on this topic for more details.

## service.beta.kubernetes.io/azure-load-balancer-disable-tcp-reset (deprecated)

Example: `service.beta.kubernetes.io/azure-load-balancer-disable-tcp-reset: "false"`

Used on: Service

This annotation only works for Azure standard load balancer backed service. This annotation is used on the Service to specify whether the load balancer should disable or enable TCP reset on idle timeout. If enabled, it helps applications to behave more predictably, to detect the termination of a connection, remove expired connections and initiate new connections. You can set the value to be either true or false.

See [Load Balancer TCP Reset](#) for more information.

### **Note:**

This annotation is deprecated.

## pod-security.kubernetes.io/enforce

Type: Label

Example: `pod-security.kubernetes.io/enforce: "baseline"`

Used on: Namespace

Value **must** be one of `privileged`, `baseline`, or `restricted` which correspond to [Pod Security Standard](#) levels. Specifically, the `enforce` label *prohibits* the creation of any Pod in the labeled Namespace which does not meet the requirements outlined in the indicated level.

See [Enforcing Pod Security at the Namespace Level](#) for more information.

## pod-security.kubernetes.io/enforce-version

Type: Label

Example: `pod-security.kubernetes.io/enforce-version: "1.31"`

Used on: Namespace

Value **must** be `latest` or a valid Kubernetes version in the format `v<major>.<minor>`. This determines the version of the [Pod Security Standard](#) policies to apply when validating a Pod.

See [Enforcing Pod Security at the Namespace Level](#) for more information.

## pod-security.kubernetes.io/audit

Type: Label

Example: `pod-security.kubernetes.io/audit: "baseline"`

Used on: Namespace

Value **must** be one of `privileged`, `baseline`, or `restricted` which correspond to [Pod Security Standard](#) levels. Specifically, the `audit` label does not prevent the creation of a Pod in the labeled Namespace which does not meet the requirements outlined in the indicated level, but adds an this annotation to the Pod.

See [Enforcing Pod Security at the Namespace Level](#) for more information.

## pod-security.kubernetes.io/audit-version

Type: Label

Example: `pod-security.kubernetes.io/audit-version: "1.31"`

Used on: Namespace

Value **must** be `latest` or a valid Kubernetes version in the format `v<major>.<minor>`. This determines the version of the [Pod Security Standard](#) policies to apply when validating a Pod.

See [Enforcing Pod Security at the Namespace Level](#) for more information.

## pod-security.kubernetes.io/warn

Type: Label

Example: `pod-security.kubernetes.io/warn: "baseline"`

Used on: Namespace

Value **must** be one of `privileged`, `baseline`, or `restricted` which correspond to [Pod Security Standard](#) levels. Specifically, the `warn` label does not prevent the creation of a Pod in the labeled Namespace which does not meet the requirements outlined in the indicated level, but returns a warning to the user after doing so. Note that warnings are also displayed when creating or updating objects that contain Pod templates, such as Deployments, Jobs, StatefulSets, etc.

See [Enforcing Pod Security at the Namespace Level](#) for more information.

## pod-security.kubernetes.io/warn-version

Type: Label

Example: `pod-security.kubernetes.io/warn-version: "1.31"`

Used on: Namespace

Value **must** be `latest` or a valid Kubernetes version in the format `v<major>.<minor>`. This determines the version of the [Pod Security Standard](#) policies to apply when validating a submitted Pod. Note that warnings are also displayed when creating or updating objects that contain Pod templates, such as Deployments, Jobs, StatefulSets, etc.

See [Enforcing Pod Security at the Namespace Level](#) for more information.

## rbac.authorization.kubernetes.io/autoupdate

Type: Annotation

Example: `rbac.authorization.kubernetes.io/autoupdate: "false"`

Used on: ClusterRole, ClusterRoleBinding, Role, RoleBinding

When this annotation is set to `"true"` on default RBAC objects created by the API server, they are automatically updated at server start to add missing permissions and subjects (extra permissions and subjects are left in place). To prevent autoupdating a particular role or rolebinding, set this annotation to `"false"`. If you create your own RBAC objects and set this annotation to `"false"`, `kubectl auth reconcile` (which allows reconciling arbitrary RBAC objects in a [manifest](#)) respects this annotation and does not automatically add missing permissions and subjects.

## kubernetes.io/psp (deprecated)

Type: Annotation

Example: `kubernetes.io/psp: restricted`

Used on: Pod

This annotation was only relevant if you were using [PodSecurityPolicy](#) objects. Kubernetes v1.31 does not support the PodSecurityPolicy API.

When the PodSecurityPolicy admission controller admitted a Pod, the admission controller modified the Pod to have this annotation. The value of the annotation was the name of the PodSecurityPolicy that was used for validation.

## seccomp.security.alpha.kubernetes.io/pod (non-functional)

Type: Annotation

Used on: Pod

Kubernetes before v1.25 allowed you to configure seccomp behavior using this annotation. See [Restrict a Container's Syscalls with seccomp](#) to learn the supported way to specify seccomp restrictions for a Pod.

## container.seccomp.security.alpha.kubernetes.io/[NAME] (non-functional)

Type: Annotation

Used on: Pod

Kubernetes before v1.25 allowed you to configure seccomp behavior using this annotation. See [Restrict a Container's Syscalls with seccomp](#) to learn the supported way to specify seccomp restrictions for a Pod.

## snapshot.storage.kubernetes.io/allow-volume-mode-change

Type: Annotation

Example: `snapshot.storage.kubernetes.io/allow-volume-mode-change: "true"`

Used on: VolumeSnapshotContent

Value can either be `true` or `false`. This determines whether a user can modify the mode of the source volume when a PersistentVolumeClaim is being created from a VolumeSnapshot.

Refer to [Converting the volume mode of a Snapshot](#) and the [Kubernetes CSI Developer Documentation](#) for more information.

## scheduler.alpha.kubernetes.io/critical-pod (deprecated)

Type: Annotation

Example: `scheduler.alpha.kubernetes.io/critical-pod: ""`

Used on: Pod

This annotation lets Kubernetes control plane know about a Pod being a critical Pod so that the descheduler will not remove this Pod.

### Note:

Starting in v1.16, this annotation was removed in favor of [Pod Priority](#).

## Annotations used for audit

- [authorization.k8s.io/decision](#)
- [authorization.k8s.io/reason](#)
- [insecure-sha1.invalid-cert.kubernetes.io/\\$hostname](#)
- [missing-san.invalid-cert.kubernetes.io/\\$hostname](#)
- [pod-security.kubernetes.io/audit-violations](#)
- [pod-security.kubernetes.io/enforce-policy](#)
- [pod-security.kubernetes.io/exempt](#)
- [validation.policy.admission.k8s.io/validation\\_failure](#)

See more details on [Audit Annotations](#).

## kubeadm

### kubeadm.alpha.kubernetes.io/cri-socket

Type: Annotation

Example: `kubeadm.alpha.kubernetes.io/cri-socket: unix:///run/containerd/container.sock`

Used on: Node

Annotation that kubeadm uses to preserve the CRI socket information given to kubeadm at `init` / `join` time for later use. kubeadm annotates the Node object with this information. The annotation remains "alpha", since ideally this should be a field in KubeletConfiguration instead.

### kubeadm.kubernetes.io/etcđ.advertise-client-urls

Type: Annotation

Example: `kubeadm.kubernetes.io/etcđ.advertise-client-urls: https://172.17.0.18:2379`

Used on: Pod

Annotation that kubeadm places on locally managed etcd Pods to keep track of a list of URLs where etcd clients should connect to. This is used mainly for etcd cluster health check purposes.

### kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint

Type: Annotation

Example: `kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: https://172.17.0.18:6443`

Used on: Pod

Annotation that kubeadm places on locally managed `kube-apiserver` Pods to keep track of the exposed advertise address/port endpoint for that API server instance.

### kubeadm.kubernetes.io/component-config.hash

Type: Annotation

Example: `kubeadm.kubernetes.io/component-config.hash: 2c26b46b68ffc68ff99b453c1d30413413422d706483bfa0f98a5e886266e7ae`

Used on: ConfigMap

Annotation that kubeadm places on ConfigMaps that it manages for configuring components. It contains a hash (SHA-256) used to determine if the user has applied settings different from the kubeadm defaults for a particular component.

### node-role.kubernetes.io/control-plane

Type: Label

Used on: Node

A marker label to indicate that the node is used to run control plane components. The kubeadm tool applies this label to the control plane

nodes that it manages. Other cluster management tools typically also set this taint.

You can label control plane nodes with this label to make it easier to schedule Pods only onto these nodes, or to avoid running Pods on the control plane. If this label is set, the [EndpointSlice controller](#) ignores that node while calculating Topology Aware Hints.

## node-role.kubernetes.io/control-plane

Type: Taint

Example: node-role.kubernetes.io/control-plane:NoSchedule

Used on: Node

Taint that kubeadm applies on control plane nodes to restrict placing Pods and allow only specific pods to schedule on them.

If this Taint is applied, control plane nodes allow only critical workloads to be scheduled onto them. You can manually remove this taint with the following command on a specific node.

```
kubectl taint nodes <node-name> node-role.kubernetes.io/control-plane
```

## node-role.kubernetes.io/master (deprecated)

Type: Taint

Used on: Node

Example: node-role.kubernetes.io/master:NoSchedule

Taint that kubeadm previously applied on control plane nodes to allow only critical workloads to schedule on them. Replaced by the [node-role.kubernetes.io/control-plane](#) taint. kubeadm no longer sets or uses this deprecated taint.

## 4.1 - Audit Annotations

This page serves as a reference for the audit annotations of the kubernetes.io namespace. These annotations apply to `Event` object from API group `audit.k8s.io`.

### Note:

The following annotations are not used within the Kubernetes API. When you [enable auditing](#) in your cluster, audit event data is written using `Event` from API group `audit.k8s.io`. The annotations apply to audit events. Audit events are different from objects in the [Event API](#) (API group `events.k8s.io`).

### k8s.io/deprecated

Example: `k8s.io/deprecated: "true"`

Value **must** be "true" or "false". The value "true" indicates that the request used a deprecated API version.

### k8s.io/removed-release

Example: `k8s.io/removed-release: "1.22"`

Value **must** be in the format ". ". It is set to target the removal release on requests made to deprecated API versions with a target removal release.

### pod-security.kubernetes.io/exempt

Example: `pod-security.kubernetes.io/exempt: namespace`

Value **must** be one of `user` , `namespace` , or `runtimeClass` which correspond to [Pod Security Exemption](#) dimensions. This annotation indicates on which dimension was based the exemption from the PodSecurity enforcement.

### pod-security.kubernetes.io/enforce-policy

Example: `pod-security.kubernetes.io/enforce-policy: restricted:latest`

Value **must** be `privileged:<version>` , `baseline:<version>` , `restricted:<version>` which correspond to [Pod Security Standard](#) levels accompanied by a version which **must** be `latest` or a valid Kubernetes version in the format `v<MAJOR>.<MINOR>` . This annotation informs about the enforcement level that allowed or denied the pod during PodSecurity admission.

See [Pod Security Standards](#) for more information.

### pod-security.kubernetes.io/audit-violations

Example: `pod-security.kubernetes.io/audit-violations: would violate PodSecurity "restricted:latest": allowPrivilegeEscalation != false (container "example" must set securityContext.allowPrivilegeEscalation=false), ...`

This annotation details an audit policy violation, it contains the [Pod Security Standard](#) level that was transgressed as well as the specific policies on the fields that were violated from the PodSecurity enforcement.

See [Pod Security Standards](#) for more information.

## authorization.k8s.io/decision

Example: `authorization.k8s.io/decision: "forbid"`

This annotation indicates whether or not a request was authorized in Kubernetes audit logs.

See [Auditing](#) for more information.

## authorization.k8s.io/reason

Example: `authorization.k8s.io/reason: "Human-readable reason for the decision"`

This annotation gives reason for the [decision](#) in Kubernetes audit logs.

See [Auditing](#) for more information.

## missing-san.invalid-cert.kubernetes.io/\$hostname

Example: `missing-san.invalid-cert.kubernetes.io/example-svc.example-namespace.svc: "relies on a legacy Common Name field instead of the SAN extension for subject validation"`

Used by Kubernetes version v1.24 and later

This annotation indicates a webhook or aggregated API server is using an invalid certificate that is missing `subjectAltNames`. Support for these certificates was disabled by default in Kubernetes 1.19, and removed in Kubernetes 1.23.

Requests to endpoints using these certificates will fail. Services using these certificates should replace them as soon as possible to avoid disruption when running in Kubernetes 1.23+ environments.

There's more information about this in the Go documentation: [X.509 CommonName deprecation](#).

## insecure-sha1.invalid-cert.kubernetes.io/\$hostname

Example: `insecure-sha1.invalid-cert.kubernetes.io/example-svc.example-namespace.svc: "uses an insecure SHA-1 signature"`

Used by Kubernetes version v1.24 and later

This annotation indicates a webhook or aggregated API server is using an

insecure certificate signed with a SHA-1 hash. Support for these insecure certificates is disabled by default in Kubernetes 1.24, and will be removed in a future release.

Services using these certificates should replace them as soon as possible, to ensure connections are secured properly and to avoid disruption in future releases.

There's more information about this in the Go documentation: [Rejecting SHA-1 certificates](#).

## validation.policy.admission.k8s.io/validation\_failure

Example: validation.policy.admission.k8s.io/validation\_failure:

```
'[{"message": "Invalid value", "policy": "policy.example.com",  
"binding": "policybinding.example.com", {"expressionIndex": "1",  
"validationActions": ["Audit"]}]'
```

Used by Kubernetes version v1.27 and later.

This annotation indicates that a admission policy validation evaluated to false for an API request, or that the validation resulted in an error while the policy was configured with `failurePolicy: Fail`.

The value of the annotation is a JSON object. The `message` in the JSON provides the message about the validation failure.

The `policy`, `binding` and `expressionIndex` in the JSON identifies the name of the `ValidatingAdmissionPolicy`, the name of the `ValidatingAdmissionPolicyBinding` and the index in the policy `validations` of the CEL expressions that failed, respectively.

The `validationActions` shows what actions were taken for this validation failure. See [Validating Admission Policy](#) for more details about `validationActions`.

## 5 - Kubernetes API

Kubernetes' API is the application that serves Kubernetes functionality through a RESTful interface and stores the state of the cluster.

Kubernetes resources and "records of intent" are all stored as API objects, and modified via RESTful calls to the API. The API allows configuration to be managed in a declarative way. Users can interact with the Kubernetes API directly, or via tools like `kubectl`. The core Kubernetes API is flexible and can also be extended to support custom resources.

# 5.1 - Workload Resources

## 5.1.1 - Pod

Pod is a collection of containers that can run on a host.

```
apiVersion: v1

import "k8s.io/api/core/v1"
```

### Pod

Pod is a collection of containers that can run on a host. This resource is created by clients and scheduled onto hosts.

- **apiVersion**: v1

- **kind**: Pod

- **metadata** ([ObjectMeta](#))

Standard object's metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **spec** ([PodSpec](#))

Specification of the desired behavior of the pod. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

- **status** ([PodStatus](#))

Most recently observed status of the pod. This data may not be up to date. Populated by the system. Read-only. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

### PodSpec

PodSpec is a description of a pod.

#### Containers

- **containers** ([][Container](#)), required

*Patch strategy: merge on key name*

List of containers belonging to the pod. Containers cannot currently be added or removed. There must be at least one container in a Pod. Cannot be updated.

- **initContainers** ([][Container](#))

*Patch strategy: merge on key name*

List of initialization containers belonging to the pod. Init containers are executed in order prior to containers being started. If any init container fails, the pod is considered to have failed and is handled according to its `restartPolicy`. The name for an init container or normal container must be unique among all containers. Init

containers may not have Lifecycle actions, Readiness probes, Liveness probes, or Startup probes. The resourceRequirements of an init container are taken into account during scheduling by finding the highest request/limit for each resource type, and then using the max of of that value or the sum of the normal containers. Limits are applied to init containers in a similar fashion. Init containers cannot currently be added or removed. Cannot be updated. More info: <https://kubernetes.io/docs/concepts/workloads/pods/init-containers/>

- **ephemeralContainers** ([][EphemeralContainer](#))

*Patch strategy: merge on key name*

List of ephemeral containers run in this pod. Ephemeral containers may be run in an existing pod to perform user-initiated actions such as debugging. This list cannot be specified when creating a pod, and it cannot be modified by updating the pod spec. In order to add an ephemeral container to an existing pod, use the pod's ephemeralContainers subresource.

- **imagePullSecrets** ([][LocalObjectReference](#))

*Patch strategy: merge on key name*

ImagePullSecrets is an optional list of references to secrets in the same namespace to use for pulling any of the images used by this PodSpec. If specified, these secrets will be passed to individual puller implementations for them to use. More info: <https://kubernetes.io/docs/concepts/containers/images#specifying-imagepullsecrets-on-a-pod>

- **enableServiceLinks** (boolean)

EnableServiceLinks indicates whether information about services should be injected into pod's environment variables, matching the syntax of Docker links. Optional: Defaults to true.

- **os** (PodOS)

Specifies the OS of the containers in the pod. Some pod and container fields are restricted if this is set.

If the OS field is set to linux, the following fields must be unset: -  
securityContext.windowsOptions

If the OS field is set to windows, following fields must be unset: -  
spec.hostPID - spec.hostIPC - spec.hostUsers -  
spec.securityContext.seLinuxOptions -  
spec.securityContext.seccompProfile - spec.securityContext.fsGroup -  
spec.securityContext.fsGroupChangePolicy -  
spec.securityContext.sysctls - spec.shareProcessNamespace -  
spec.securityContext.runAsUser - spec.securityContext.runAsGroup -  
spec.securityContext.supplementalGroups -  
spec.containers[ ].securityContext.seLinuxOptions -  
spec.containers[ ].securityContext.seccompProfile -  
spec.containers[ ].securityContext.capabilities -  
spec.containers[ ].securityContext.readOnlyRootFilesystem -  
spec.containers[ ].securityContext.privileged -  
spec.containers[ ].securityContext.allowPrivilegeEscalation -  
spec.containers[ ].securityContext.procMount -  
spec.containers[ ].securityContext.runAsUser -  
spec.containers[ \* ].securityContext.runAsGroup

*PodOS defines the OS parameters of a pod.*

- o **os.name** (string), required

Name is the name of the operating system. The currently supported values are linux and windows. Additional value may be defined in future and can be one of: <https://github.com/opencontainers/runtime-spec/blob/master/config.md#platform-specific-configuration> Clients should expect to handle additional values and treat unrecognized values in this field as os: null

## Volumes

- **volumes** ([]Volume)

*Patch strategies: retainKeys, merge on key name*

List of volumes that can be mounted by containers belonging to the pod. More info: <https://kubernetes.io/docs/concepts/storage/volumes>

## Scheduling

- **nodeSelector** (map[string]string)

NodeSelector is a selector which must be true for the pod to fit on a node. Selector which must match a node's labels for the pod to be scheduled on that node. More info: <https://kubernetes.io/docs/concepts/configuration/assign-pod-node/>

- **nodeName** (string)

NodeName is a request to schedule this pod onto a specific node. If it is non-empty, the scheduler simply schedules this pod onto that node, assuming that it fits resource requirements.

- **affinity** (Affinity)

If specified, the pod's scheduling constraints

*Affinity is a group of affinity scheduling rules.*

- o **affinity.nodeAffinity** ([NodeAffinity](#))

Describes node affinity scheduling rules for the pod.

- o **affinity.podAffinity** ([PodAffinity](#))

Describes pod affinity scheduling rules (e.g. co-locate this pod in the same node, zone, etc. as some other pod(s)).

- o **affinity.podAntiAffinity** ([PodAntiAffinity](#))

Describes pod anti-affinity scheduling rules (e.g. avoid putting this pod in the same node, zone, etc. as some other pod(s)).

- **tolerations** ([]Toleration)

If specified, the pod's tolerations.

*The pod this Toleration is attached to tolerates any taint that matches the triple <key,value,effect> using the matching operator .*

- o **tolerations.key** (string)

Key is the taint key that the toleration applies to. Empty means match all taint keys. If the key is empty, operator must be Exists; this combination means to match all values and all keys.

- **tolerations.operator** (string)

Operator represents a key's relationship to the value. Valid operators are Exists and Equal. Defaults to Equal. Exists is equivalent to wildcard for value, so that a pod can tolerate all taints of a particular category.

- **tolerations.value** (string)

Value is the taint value the toleration matches to. If the operator is Exists, the value should be empty, otherwise just a regular string.

- **tolerations.effect** (string)

Effect indicates the taint effect to match. Empty means match all taint effects. When specified, allowed values are NoSchedule, PreferNoSchedule and NoExecute.

- **tolerations.tolerationSeconds** (int64)

TolerationSeconds represents the period of time the toleration (which must be of effect NoExecute, otherwise this field is ignored) tolerates the taint. By default, it is not set, which means tolerate the taint forever (do not evict). Zero and negative values will be treated as 0 (evict immediately) by the system.

- **schedulerName** (string)

If specified, the pod will be dispatched by specified scheduler. If not specified, the pod will be dispatched by default scheduler.

- **runtimeClassName** (string)

RuntimeClassName refers to a RuntimeClass object in the node.k8s.io group, which should be used to run this pod. If no RuntimeClass resource matches the named class, the pod will not be run. If unset or empty, the "legacy" RuntimeClass will be used, which is an implicit class with an empty definition that uses the default runtime handler. More info: <https://git.k8s.io/enhancements/keps/sig-node/585-runtime-class>

- **priorityClassName** (string)

If specified, indicates the pod's priority. "system-node-critical" and "system-cluster-critical" are two special keywords which indicate the highest priorities with the former being the highest priority. Any other name must be defined by creating a PriorityClass object with that name. If not specified, the pod priority will be default or zero if there is no default.

- **priority** (int32)

The priority value. Various system components use this field to find the priority of the pod. When Priority Admission Controller is enabled, it prevents users from setting this field. The admission controller populates this field from PriorityClassName. The higher the value, the higher the priority.

- **preemptionPolicy** (string)

PreemptionPolicy is the Policy for preempting pods with lower priority. One of Never, PreemptLowerPriority. Defaults to PreemptLowerPriority if unset.

- **topologySpreadConstraints** ([]TopologySpreadConstraint)

*Patch strategy: merge on key topologyKey*

*Map: unique values on keys topologyKey, whenUnsatisfiable will be kept during a merge*

TopologySpreadConstraints describes how a group of pods ought to spread across topology domains. Scheduler will schedule pods in a way which abides by the constraints. All topologySpreadConstraints are ANDed.

*TopologySpreadConstraint specifies how to spread matching pods among the given topology.*

- **topologySpreadConstraints.maxSkew** (int32), required

MaxSkew describes the degree to which pods may be unevenly distributed. When `whenUnsatisfiable=DoNotSchedule`, it is the maximum permitted difference between the number of matching pods in the target topology and the global minimum. The global minimum is the minimum number of matching pods in an eligible domain or zero if the number of eligible domains is less than MinDomains. For example, in a 3-zone cluster, MaxSkew is set to 1, and pods with the same labelSelector spread as 2/2/1: In this case, the global minimum is 1. | zone1 | zone2 | zone3 | | P P | P P | P | - if MaxSkew is 1, incoming pod can only be scheduled to zone3 to become 2/2/2; scheduling it onto zone1(zone2) would make the ActualSkew(3-1) on zone1(zone2) violate MaxSkew(1). - if MaxSkew is 2, incoming pod can be scheduled onto any zone. When `whenUnsatisfiable=ScheduleAnyway`, it is used to give higher precedence to topologies that satisfy it. It's a required field. Default value is 1 and 0 is not allowed.

- **topologySpreadConstraints.topologyKey** (string), required

TopologyKey is the key of node labels. Nodes that have a label with this key and identical values are considered to be in the same topology. We consider each `<key, value>` as a "bucket", and try to put balanced number of pods into each bucket. We define a domain as a particular instance of a topology. Also, we define an eligible domain as a domain whose nodes meet the requirements of nodeAffinityPolicy and nodeTaintsPolicy. e.g. If TopologyKey is "kubernetes.io/hostname", each Node is a domain of that topology. And, if TopologyKey is "topology.kubernetes.io/zone", each zone is a domain of that topology. It's a required field.

- **topologySpreadConstraints.whenUnsatisfiable** (string), required

WhenUnsatisfiable indicates how to deal with a pod if it doesn't satisfy the spread constraint. - `DoNotSchedule` (default) tells the scheduler not to schedule it. - `ScheduleAnyway` tells the scheduler to schedule the pod in any location, but giving higher precedence to topologies that would help reduce the skew. A constraint is considered "Unsatisfiable" for an incoming pod if and only if every possible node assignment for that pod would violate "MaxSkew" on some topology. For example, in a 3-zone cluster, MaxSkew is set to 1, and pods with the same labelSelector spread as 3/1/1: | zone1 | zone2 | zone3 | | P P P | P | P | If WhenUnsatisfiable is set to `DoNotSchedule`, incoming pod can only be scheduled to zone2(zone3) to become 3/2/1(3/1/2) as ActualSkew(2-1) on zone2(zone3) satisfies MaxSkew(1). In other words, the cluster can still be

imbalanced, but scheduler won't make it *more* imbalanced. It's a required field.

- **topologySpreadConstraints.labelSelector** ([LabelSelector](#))

LabelSelector is used to find matching pods. Pods that match this label selector are counted to determine the number of pods in their corresponding topology domain.

- **topologySpreadConstraints.matchLabelKeys** ([]string)

*Atomic: will be replaced during a merge*

MatchLabelKeys is a set of pod label keys to select the pods over which spreading will be calculated. The keys are used to lookup values from the incoming pod labels, those key-value labels are ANDed with labelSelector to select the group of existing pods over which spreading will be calculated for the incoming pod. The same key is forbidden to exist in both MatchLabelKeys and LabelSelector. MatchLabelKeys cannot be set when LabelSelector isn't set. Keys that don't exist in the incoming pod labels will be ignored. A null or empty list means only match against labelSelector.

This is a beta field and requires the MatchLabelKeysInPodTopologySpread feature gate to be enabled (enabled by default).

- **topologySpreadConstraints.minDomains** (int32)

MinDomains indicates a minimum number of eligible domains. When the number of eligible domains with matching topology keys is less than minDomains, Pod Topology Spread treats "global minimum" as 0, and then the calculation of Skew is performed. And when the number of eligible domains with matching topology keys equals or greater than minDomains, this value has no effect on scheduling. As a result, when the number of eligible domains is less than minDomains, scheduler won't schedule more than maxSkew Pods to those domains. If value is nil, the constraint behaves as if MinDomains is equal to 1. Valid values are integers greater than 0. When value is not nil, WhenUnsatisfiable must be DoNotSchedule.

For example, in a 3-zone cluster, MaxSkew is set to 2, MinDomains is set to 5 and pods with the same labelSelector spread as 2/2/2: | zone1 | zone2 | zone3 | | P P | P P | P P | The number of domains is less than 5(MinDomains), so "global minimum" is treated as 0. In this situation, new pod with the same labelSelector cannot be scheduled, because computed skew will be 3(3 - 0) if new Pod is scheduled to any of the three zones, it will violate MaxSkew.

This is a beta field and requires the MinDomainsInPodTopologySpread feature gate to be enabled (enabled by default).

- **topologySpreadConstraints.nodeAffinityPolicy** (string)

NodeAffinityPolicy indicates how we will treat Pod's nodeAffinity/nodeSelector when calculating pod topology spread skew. Options are: - Honor: only nodes matching nodeAffinity/nodeSelector are included in the calculations. - Ignore: nodeAffinity/nodeSelector are ignored. All nodes are included in the calculations.

If this value is nil, the behavior is equivalent to the Honor policy. This is a beta-level feature default enabled by the NodeInclusionPolicyInPodTopologySpread feature flag.

- **topologySpreadConstraints.nodeTaintsPolicy** (string)

NodeTaintsPolicy indicates how we will treat node taints when calculating pod topology spread skew. Options are: - Honor: nodes without taints, along with tainted nodes for which the incoming pod has a toleration, are included. - Ignore: node taints are ignored. All nodes are included.

If this value is nil, the behavior is equivalent to the Ignore policy. This is a beta-level feature default enabled by the NodeInclusionPolicyInPodTopologySpread feature flag.

- **overhead** (map[string][Quantity](#))

Overhead represents the resource overhead associated with running a pod for a given RuntimeClass. This field will be autopopulated at admission time by the RuntimeClass admission controller. If the RuntimeClass admission controller is enabled, overhead must not be set in Pod create requests. The RuntimeClass admission controller will reject Pod create requests which have the overhead already set. If RuntimeClass is configured and selected in the PodSpec, Overhead will be set to the value defined in the corresponding RuntimeClass, otherwise it will remain unset and treated as zero. More info: <https://git.k8s.io/enhancements/keps/sig-node/688-pod-overhead/README.md>

## Lifecycle

- **restartPolicy** (string)

Restart policy for all containers within the pod. One of Always, OnFailure, Never. In some contexts, only a subset of those values may be permitted. Default to Always. More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#restart-policy>

- **terminationGracePeriodSeconds** (int64)

Optional duration in seconds the pod needs to terminate gracefully. May be decreased in delete request. Value must be non-negative integer. The value zero indicates stop immediately via the kill signal (no opportunity to shut down). If this value is nil, the default grace period will be used instead. The grace period is the duration in seconds after the processes running in the pod are sent a termination signal and the time when the processes are forcibly halted with a kill signal. Set this value longer than the expected cleanup time for your process. Defaults to 30 seconds.

- **activeDeadlineSeconds** (int64)

Optional duration in seconds the pod may be active on the node relative to StartTime before the system will actively try to mark it failed and kill associated containers. Value must be a positive integer.

- **readinessGates** ([]PodReadinessGate)

If specified, all readiness gates will be evaluated for pod readiness. A pod is ready when all its containers are ready AND all conditions specified in the readiness gates have status equal to "True" More info: <https://git.k8s.io/enhancements/keps/sig-network/580-pod-readiness-gates>

## [readiness-gates](#)

*PodReadinessGate contains the reference to a pod condition*

- **readinessGates.conditionType** (string), required  
ConditionType refers to a condition in the pod's condition list with matching type.

## Hostname and Name resolution

- **hostname** (string)

Specifies the hostname of the Pod If not specified, the pod's hostname will be set to a system-defined value.

- **setHostnameAsFQDN** (boolean)

If true the pod's hostname will be configured as the pod's FQDN, rather than the leaf name (the default). In Linux containers, this means setting the FQDN in the hostname field of the kernel (the nodename field of struct utsname). In Windows containers, this means setting the registry value of hostname for the registry key HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters to FQDN. If a pod does not have FQDN, this has no effect. Default to false.

- **subdomain** (string)

If specified, the fully qualified Pod hostname will be "<hostname>.<subdomain>.<pod namespace>.svc.<cluster domain>". If not specified, the pod will not have a domainname at all.

- **hostAliases** ([]HostAlias)

*Patch strategy: merge on key ip*

HostAliases is an optional list of hosts and IPs that will be injected into the pod's hosts file if specified. This is only valid for non-hostNetwork pods.

*HostAlias holds the mapping between IP and hostnames that will be injected as an entry in the pod's hosts file.*

- **hostAliases.hostnames** ([]string)  
Hostnames for the above IP address.
- **hostAliases.ip** (string)  
IP address of the host file entry.

- **dnsConfig** (PodDNSConfig)

Specifies the DNS parameters of a pod. Parameters specified here will be merged to the generated DNS configuration based on DNSPolicy.

*PodDNSConfig defines the DNS parameters of a pod in addition to those generated from DNSPolicy.*

- **dnsConfig.nameservers** ([]string)  
A list of DNS name server IP addresses. This will be appended to the base nameservers generated from DNSPolicy.  
Duplicated nameservers will be removed.
- **dnsConfig.options** ([]PodDNSConfigOption)

A list of DNS resolver options. This will be merged with the base options generated from DNSPolicy. Duplicated entries will be removed. Resolution options given in Options will override those that appear in the base DNSPolicy.

*PodDNSConfigOption defines DNS resolver options of a pod.*

- **dnsConfig.options.name** (string)

Required.

- **dnsConfig.options.value** (string)

- **dnsConfig.searches** ([]string)

A list of DNS search domains for host-name lookup. This will be appended to the base search paths generated from DNSPolicy. Duplicated search paths will be removed.

- **dnsPolicy** (string)

Set DNS policy for the pod. Defaults to "ClusterFirst". Valid values are 'ClusterFirstWithHostNet', 'ClusterFirst', 'Default' or 'None'. DNS parameters given in DNSConfig will be merged with the policy selected with DNSPolicy. To have DNS options set along with hostNetwork, you have to specify DNS policy explicitly to 'ClusterFirstWithHostNet'.

## Hosts namespaces

- **hostNetwork** (boolean)

Host networking requested for this pod. Use the host's network namespace. If this option is set, the ports that will be used must be specified. Default to false.

- **hostPID** (boolean)

Use the host's pid namespace. Optional: Default to false.

- **hostIPC** (boolean)

Use the host's ipc namespace. Optional: Default to false.

- **shareProcessNamespace** (boolean)

Share a single process namespace between all of the containers in a pod. When this is set containers will be able to view and signal processes from other containers in the same pod, and the first process in each container will not be assigned PID 1. HostPID and ShareProcessNamespace cannot both be set. Optional: Default to false.

## Service account

- **serviceAccountName** (string)

ServiceAccountName is the name of the ServiceAccount to use to run this pod. More info: <https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/>

- **automountServiceAccountToken** (boolean)

AutomountServiceAccountToken indicates whether a service account token should be automatically mounted.

## Security context

- **securityContext** (PodSecurityContext)

SecurityContext holds pod-level security attributes and common container settings. Optional: Defaults to empty. See type description for default values of each field.

*PodSecurityContext holds pod-level security attributes and common container settings. Some fields are also present in container.securityContext. Field values of container.securityContext take precedence over field values of PodSecurityContext.*

- **securityContext.runAsUser** (int64)

The UID to run the entrypoint of the container process. Defaults to user specified in image metadata if unspecified. May also be set in SecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence for that container. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.runAsNonRoot** (boolean)

Indicates that the container must run as a non-root user. If true, the Kubelet will validate the image at runtime to ensure that it does not run as UID 0 (root) and fail to start the container if it does. If unset or false, no such validation will be performed. May also be set in SecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.

- **securityContext.runAsGroup** (int64)

The GID to run the entrypoint of the container process. Uses runtime default if unset. May also be set in SecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence for that container. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.supplementalGroups** ([]int64)

A list of groups applied to the first process run in each container, in addition to the container's primary GID, the fsGroup (if specified), and group memberships defined in the container image for the uid of the container process. If unspecified, no additional groups are added to any container. Note that group memberships defined in the container image for the uid of the container process are still effective, even if they are not included in this list. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.fsGroup** (int64)

A special supplemental group that applies to all containers in a pod. Some volume types allow the Kubelet to change the ownership of that volume to be owned by the pod:

1. The owning GID will be the FSGroup
2. The setgid bit is set (new files created in the volume will be owned by FSGroup)
3. The permission bits are OR'd with rw-rw----

If unset, the Kubelet will not modify the ownership and permissions of any volume. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.fsGroupChangePolicy** (string)

`fsGroupChangePolicy` defines behavior of changing ownership and permission of the volume before being exposed inside Pod. This field will only apply to volume types which support `fsGroup` based ownership (and permissions). It will have no effect on ephemeral volume types such as: `secret`, `configmaps` and `emptydir`. Valid values are `"OnRootMismatch"` and `"Always"`. If not specified, `"Always"` is used. Note that this field cannot be set when `spec.os.name` is `windows`.

- **securityContext.seccompProfile** (`SeccompProfile`)

The `seccomp` options to use by the containers in this pod. Note that this field cannot be set when `spec.os.name` is `windows`.

*SeccompProfile defines a pod/container's seccomp profile settings. Only one profile source may be set.*

- **securityContext.seccompProfile.type** (string), required

`type` indicates which kind of `seccomp` profile will be applied. Valid options are:

`Localhost` - a profile defined in a file on the node should be used. `RuntimeDefault` - the container runtime default profile should be used. `Unconfined` - no profile should be applied.

- **securityContext.seccompProfile.localhostProfile** (string)

`localhostProfile` indicates a profile defined in a file on the node should be used. The profile must be preconfigured on the node to work. Must be a descending path, relative to the kubelet's configured `seccomp` profile location. Must be set if `type` is `"Localhost"`. Must NOT be set for any other type.

- **securityContext.seLinuxOptions** (`SELinuxOptions`)

The `SELinux` context to be applied to all containers. If unspecified, the container runtime will allocate a random `SELinux` context for each container. May also be set in `SecurityContext`. If set in both `SecurityContext` and `PodSecurityContext`, the value specified in `SecurityContext` takes precedence for that container. Note that this field cannot be set when `spec.os.name` is `windows`.

*SELinuxOptions are the labels to be applied to the container*

- **securityContext.seLinuxOptions.level** (string)

`Level` is `SELinux` level label that applies to the container.

- **securityContext.seLinuxOptions.role** (string)

`Role` is a `SELinux` role label that applies to the container.

- **securityContext.seLinuxOptions.type** (string)

`Type` is a `SELinux` type label that applies to the container.

- **securityContext.seLinuxOptions.user** (string)

`User` is a `SELinux` user label that applies to the container.

- **securityContext.sysctls** (`[]Sysctl`)

`Sysctls` hold a list of namespaced `sysctls` used for the pod.

Pods with unsupported sysctls (by the container runtime) might fail to launch. Note that this field cannot be set when spec.os.name is windows.

*Sysctl defines a kernel parameter to be set*

- **securityContext.sysctls.name** (string), required

Name of a property to set

- **securityContext.sysctls.value** (string), required

Value of a property to set

- **securityContext.windowsOptions**

(WindowsSecurityContextOptions)

The Windows specific settings applied to all containers. If unspecified, the options within a container's SecurityContext will be used. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence. Note that this field cannot be set when spec.os.name is linux.

*WindowsSecurityContextOptions contain Windows-specific options and credentials.*

- **securityContext.windowsOptions.gmsaCredentialSpec** (string)

GMSACredentialSpec is where the GMSA admission webhook (<https://github.com/kubernetes-sigs/windows-gmsa>) inlines the contents of the GMSA credential spec named by the GMSACredentialSpecName field.

- **securityContext.windowsOptions.gmsaCredentialSpecName** (string)

GMSACredentialSpecName is the name of the GMSA credential spec to use.

- **securityContext.windowsOptions.hostProcess** (boolean)

HostProcess determines if a container should be run as a 'Host Process' container. All of a Pod's containers must have the same effective HostProcess value (it is not allowed to have a mix of HostProcess containers and non-HostProcess containers). In addition, if HostProcess is true then HostNetwork must also be set to true.

- **securityContext.windowsOptions.runAsUserName** (string)

The UserName in Windows to run the entrypoint of the container process. Defaults to the user specified in image metadata if unspecified. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.

## Alpha level

- **hostUsers** (boolean)

Use the host's user namespace. Optional: Default to true. If set to true or not present, the pod will be run in the host user namespace,

useful for when the pod needs a feature only available to the host user namespace, such as loading a kernel module with CAP\_SYS\_MODULE. When set to false, a new userns is created for the pod. Setting false is useful for mitigating container breakout vulnerabilities even allowing users to run their containers as root without actually having root privileges on the host. This field is alpha-level and is only honored by servers that enable the UserNamespacesSupport feature.

- **resourceClaims** ([]PodResourceClaim)

*Patch strategies: retainKeys, merge on key name*

*Map: unique values on key name will be kept during a merge*

ResourceClaims defines which ResourceClaims must be allocated and reserved before the Pod is allowed to start. The resources will be made available to those containers which consume them by name.

This is an alpha field and requires enabling the DynamicResourceAllocation feature gate.

This field is immutable.

*PodResourceClaim references exactly one ResourceClaim through a ClaimSource. It adds a name to it that uniquely identifies the ResourceClaim inside the Pod. Containers that need access to the ResourceClaim reference it with this name.*

- **resourceClaims.name** (string), required

Name uniquely identifies this resource claim inside the pod. This must be a DNS\_LABEL.

- **resourceClaims.source** (ClaimSource)

Source describes where to find the ResourceClaim.

*\*ClaimSource describes a reference to a ResourceClaim.*

Exactly one of these fields should be set. Consumers of this type must treat an empty object as if it has an unknown value.\*

- **resourceClaims.source.resourceClaimName** (string)

ResourceClaimName is the name of a ResourceClaim object in the same namespace as this pod.

- **resourceClaims.source.resourceClaimTemplateName** (string)

ResourceClaimTemplateName is the name of a ResourceClaimTemplate object in the same namespace as this pod.

The template will be used to create a new ResourceClaim, which will be bound to this pod. When this pod is deleted, the ResourceClaim will also be deleted. The pod name and resource name, along with a generated component, will be used to form a unique name for the ResourceClaim, which will be recorded in pod.status.resourceClaimStatuses.

This field is immutable and no changes will be made to the corresponding ResourceClaim by the control plane after creating the ResourceClaim.

- **schedulingGates** ([]PodSchedulingGate)

*Patch strategy: merge on key name*

*Map: unique values on key name will be kept during a merge*

SchedulingGates is an opaque list of values that if specified will block scheduling the pod. If schedulingGates is not empty, the pod will stay in the SchedulingGated state and the scheduler will not attempt to schedule the pod.

SchedulingGates can only be set at pod creation time, and be removed only afterwards.

This is a beta feature enabled by the PodSchedulingReadiness feature gate.

*PodSchedulingGate is associated to a Pod to guard its scheduling.*

- **schedulingGates.name** (string), required

Name of the scheduling gate. Each scheduling gate must have a unique name field.

## Deprecated

- **serviceAccount** (string)

DeprecatedServiceAccount is a depreciated alias for ServiceAccountName. Deprecated: Use serviceAccountName instead.

## Container

A single application container that you want to run within a pod.

- **name** (string), required

Name of the container specified as a DNS\_LABEL. Each container in a pod must have a unique name (DNS\_LABEL). Cannot be updated.

## Image

- **image** (string)

Container image name. More info: <https://kubernetes.io/docs/concepts/containers/images> This field is optional to allow higher level config management to default or override container images in workload controllers like Deployments and StatefulSets.

- **imagePullPolicy** (string)

Image pull policy. One of Always, Never, IfNotPresent. Defaults to Always if :latest tag is specified, or IfNotPresent otherwise. Cannot be updated. More info: <https://kubernetes.io/docs/concepts/containers/images#updating-images>

## Entrypoint

- **command** ([]string)

Entrypoint array. Not executed within a shell. The container image's ENTRYPPOINT is used if this is not provided. Variable references \$(VAR\_NAME) are expanded using the container's environment. If a

variable cannot be resolved, the reference in the input string will be unchanged. Double \$\$ are reduced to a single \$, which allows for escaping the \$(VAR\_NAME) syntax: i.e. "\$\$(VAR\_NAME)" will produce the string literal "\$(VAR\_NAME)". Escaped references will never be expanded, regardless of whether the variable exists or not. Cannot be updated. More info: <https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/#running-a-command-in-a-shell>

- **args** ([]string)

Arguments to the entrypoint. The container image's CMD is used if this is not provided. Variable references \$(VAR\_NAME) are expanded using the container's environment. If a variable cannot be resolved, the reference in the input string will be unchanged. Double \$\$ are reduced to a single \$, which allows for escaping the \$(VAR\_NAME) syntax: i.e. "\$\$(VAR\_NAME)" will produce the string literal "\$(VAR\_NAME)". Escaped references will never be expanded, regardless of whether the variable exists or not. Cannot be updated. More info: <https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/#running-a-command-in-a-shell>

- **workingDir** (string)

Container's working directory. If not specified, the container runtime's default will be used, which might be configured in the container image. Cannot be updated.

## Ports

- **ports** ([]ContainerPort)

*Patch strategy: merge on key containerPort*

*Map: unique values on keys containerPort, protocol will be kept during a merge*

List of ports to expose from the container. Not specifying a port here DOES NOT prevent that port from being exposed. Any port which is listening on the default "0.0.0.0" address inside a container will be accessible from the network. Modifying this array with strategic merge patch may corrupt the data. For more information See <https://github.com/kubernetes/kubernetes/issues/108255>. Cannot be updated.

*ContainerPort represents a network port in a single container.*

- **ports.containerPort** (int32), required

Number of port to expose on the pod's IP address. This must be a valid port number, 0 < x < 65536.

- **ports.hostIP** (string)

What host IP to bind the external port to.

- **ports.hostPort** (int32)

Number of port to expose on the host. If specified, this must be a valid port number, 0 < x < 65536. If HostNetwork is specified, this must match ContainerPort. Most containers do not need this.

- **ports.name** (string)

If specified, this must be an IANA\_SVC\_NAME and unique

within the pod. Each named port in a pod must have a unique name. Name for the port that can be referred to by services.

- **ports.protocol** (string)  
Protocol for port. Must be UDP, TCP, or SCTP. Defaults to "TCP".

## Environment variables

- **env** ([]EnvVar)

*Patch strategy: merge on key name*

List of environment variables to set in the container. Cannot be updated.

*EnvVar represents an environment variable present in a Container.*

- **env.name** (string), required  
Name of the environment variable. Must be a C\_IDENTIFIER.  
○ **env.value** (string)  
Variable references \$(VAR\_NAME) are expanded using the previously defined environment variables in the container and any service environment variables. If a variable cannot be resolved, the reference in the input string will be unchanged. Double \$\$ are reduced to a single \$, which allows for escaping the \$(VAR\_NAME) syntax: i.e. "\$\$(VAR\_NAME)" will produce the string literal "\$(VAR\_NAME)". Escaped references will never be expanded, regardless of whether the variable exists or not. Defaults to "".

- **env.valueFrom** (EnvVarSource)  
Source for the environment variable's value. Cannot be used if value is not empty.

*EnvVarSource represents a source for the value of an EnvVar.*

- **env.valueFrom.configMapKeyRef**  
(ConfigMapKeySelector)  
Selects a key of a ConfigMap.  
*Selects a key from a ConfigMap.*
  - **env.valueFrom.configMapKeyRef.key** (string), required  
The key to select.
  - **env.valueFrom.configMapKeyRef.name** (string)  
Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>
  - **env.valueFrom.configMapKeyRef.optional**  
(boolean)  
Specify whether the ConfigMap or its key must be defined
  - **env.valueFrom.fieldRef** ([ObjectFieldSelector](#))  
Selects a field of the pod: supports metadata.name,

```
metadata.namespace, metadata.labels['<KEY>'] ,  
metadata.annotations['<KEY>'] , spec.nodeName,  
spec.serviceAccountName, status.hostIP, status.podIP,  
status.podIPs.
```

- **env.valueFrom.resourceFieldRef**

[\(ResourceFieldSelector\)](#)

Selects a resource of the container: only resources limits and requests (limits.cpu, limits.memory, limits.ephemeral-storage, requests.cpu, requests.memory and requests.ephemeral-storage) are currently supported.

- **env.valueFrom.secretKeyRef** (SecretKeySelector)

Selects a key of a secret in the pod's namespace

*SecretKeySelector selects a key of a Secret.*

- **env.valueFrom.secretKeyRef.key** (string),  
required

The key of the secret to select from. Must be a valid secret key.

- **env.valueFrom.secretKeyRef.name** (string)

Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **env.valueFrom.secretKeyRef.optional** (boolean)

Specify whether the Secret or its key must be defined

- **envFrom ([]EnvFromSource)**

List of sources to populate environment variables in the container. The keys defined within a source must be a C\_IDENTIFIER. All invalid keys will be reported as an event when the container is starting. When a key exists in multiple sources, the value associated with the last source will take precedence. Values defined by an Env with a duplicate key will take precedence. Cannot be updated.

*EnvFromSource represents the source of a set of ConfigMaps*

- **envFrom.configMapRef** (ConfigMapEnvSource)

The ConfigMap to select from

\*ConfigMapEnvSource selects a ConfigMap to populate the environment variables with.

The contents of the target ConfigMap's Data field will represent the key-value pairs as environment variables.\*

- **envFrom.configMapRef.name** (string)

Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **envFrom.configMapRef.optional** (boolean)

Specify whether the ConfigMap must be defined

- **envFrom.prefix** (string)

An optional identifier to prepend to each key in the ConfigMap. Must be a C\_IDENTIFIER.

- o **envFrom.secretRef** (SecretEnvSource)

The Secret to select from

\*SecretEnvSource selects a Secret to populate the environment variables with.

The contents of the target Secret's Data field will represent the key-value pairs as environment variables.\*

- **envFrom.secretRef.name** (string)

Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **envFrom.secretRef.optional** (boolean)

Specify whether the Secret must be defined

## Volumes

- **volumeMounts** ([]VolumeMount)

*Patch strategy: merge on key mountPath*

Pod volumes to mount into the container's filesystem. Cannot be updated.

*VolumeMount describes a mounting of a Volume within a container.*

- o **volumeMounts.mountPath** (string), required

Path within the container at which the volume should be mounted. Must not contain ':'.

- o **volumeMounts.name** (string), required

This must match the Name of a Volume.

- o **volumeMounts.mountPropagation** (string)

mountPropagation determines how mounts are propagated from the host to container and the other way around. When not set, MountPropagationNone is used. This field is beta in 1.10.

- o **volumeMounts.readOnly** (boolean)

Mounted read-only if true, read-write otherwise (false or unspecified). Defaults to false.

- o **volumeMounts.subPath** (string)

Path within the volume from which the container's volume should be mounted. Defaults to "" (volume's root).

- o **volumeMounts.subPathExpr** (string)

Expanded path within the volume from which the container's volume should be mounted. Behaves similarly to SubPath but environment variable references \${VAR\_NAME} are expanded using the container's environment. Defaults to "" (volume's root). SubPathExpr and SubPath are mutually exclusive.

- **volumeDevices** ([]VolumeDevice)

*Patch strategy: merge on key devicePath*

volumeDevices is the list of block devices to be used by the container.

*volumeDevice describes a mapping of a raw block device within a container.*

- **volumeDevices.devicePath** (string), required

devicePath is the path inside of the container that the device will be mapped to.

- **volumeDevices.name** (string), required

name must match the name of a persistentVolumeClaim in the pod

## Resources

- **resources** (ResourceRequirements)

Compute Resources required by this container. Cannot be updated.

More info: <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

*ResourceRequirements describes the compute resource requirements.*

- **resources.claims** ([]ResourceClaim)

*Map: unique values on key name will be kept during a merge*

Claims lists the names of resources, defined in spec.resourceClaims, that are used by this container.

This is an alpha field and requires enabling the DynamicResourceAllocation feature gate.

This field is immutable. It can only be set for containers.

*ResourceClaim references one entry in PodSpec.ResourceClaims.*

- **resources.claims.name** (string), required

Name must match the name of one entry in pod.spec.resourceClaims of the Pod where this field is used. It makes that resource available inside a container.

- **resources.limits** (map[string]Quantity)

Limits describes the maximum amount of compute resources allowed. More info: <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

- **resources.requests** (map[string]Quantity)

Requests describes the minimum amount of compute resources required. If Requests is omitted for a container, it defaults to Limits if that is explicitly specified, otherwise to an implementation-defined value. Requests cannot exceed Limits. More info: <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

- **resizePolicy** ([]ContainerResizePolicy)

*Atomic: will be replaced during a merge*

Resources resize policy for the container.

*ContainerResizePolicy represents resource resize policy for the container.*

- **resizePolicy.resourceName** (string), required  
Name of the resource to which this resource resize policy applies. Supported values: cpu, memory.
- **resizePolicy.restartPolicy** (string), required  
Restart policy to apply when specified resource is resized. If not specified, it defaults to NotRequired.

## Lifecycle

- **lifecycle** (Lifecycle)

Actions that the management system should take in response to container lifecycle events. Cannot be updated.

*Lifecycle describes actions that the management system should take in response to container lifecycle events. For the PostStart and PreStop lifecycle handlers, management of the container blocks until the action is complete, unless the container process fails, in which case the handler is aborted.*

- **lifecycle.postStart** ([LifecycleHandler](#))  
PostStart is called immediately after a container is created. If the handler fails, the container is terminated and restarted according to its restart policy. Other management of the container blocks until the hook completes. More info: <https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks/#container-hooks>

- **lifecycle.preStop** ([LifecycleHandler](#))  
PreStop is called immediately before a container is terminated due to an API request or management event such as liveness/startup probe failure, preemption, resource contention, etc. The handler is not called if the container crashes or exits. The Pod's termination grace period countdown begins before the PreStop hook is executed. Regardless of the outcome of the handler, the container will eventually terminate within the Pod's termination grace period (unless delayed by finalizers). Other management of the container blocks until the hook completes or until the termination grace period is reached. More info: <https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks/#container-hooks>

- **terminationMessagePath** (string)

Optional: Path at which the file to which the container's termination message will be written is mounted into the container's filesystem. Message written is intended to be brief final status, such as an assertion failure message. Will be truncated by the node if greater than 4096 bytes. The total message length across all containers will be limited to 12kb. Defaults to /dev/termination-log. Cannot be updated.

- **terminationMessagePolicy** (string)

Indicate how the termination message should be populated. File will use the contents of terminationMessagePath to populate the container status message on both success and failure. FallbackToLogsOnError will use the last chunk of container log

output if the termination message file is empty and the container exited with an error. The log output is limited to 2048 bytes or 80 lines, whichever is smaller. Defaults to File. Cannot be updated.

- **livenessProbe** ([Probe](#))

Periodic probe of container liveness. Container will be restarted if the probe fails. Cannot be updated. More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes>

- **readinessProbe** ([Probe](#))

Periodic probe of container service readiness. Container will be removed from service endpoints if the probe fails. Cannot be updated. More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes>

- **startupProbe** ([Probe](#))

StartupProbe indicates that the Pod has successfully initialized. If specified, no other probes are executed until this completes successfully. If this probe fails, the Pod will be restarted, just as if the livenessProbe failed. This can be used to provide different probe parameters at the beginning of a Pod's lifecycle, when it might take a long time to load data or warm a cache, than during steady-state operation. This cannot be updated. More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes>

- **restartPolicy** (string)

RestartPolicy defines the restart behavior of individual containers in a pod. This field may only be set for init containers, and the only allowed value is "Always". For non-init containers or when this field is not specified, the restart behavior is defined by the Pod's restart policy and the container type. Setting the RestartPolicy as "Always" for the init container will have the following effect: this init container will be continually restarted on exit until all regular containers have terminated. Once all regular containers have completed, all init containers with restartPolicy "Always" will be shut down. This lifecycle differs from normal init containers and is often referred to as a "sidecar" container. Although this init container still starts in the init container sequence, it does not wait for the container to complete before proceeding to the next init container. Instead, the next init container starts immediately after this init container is started, or after any startupProbe has successfully completed.

## Security Context

- **securityContext** ([SecurityContext](#))

SecurityContext defines the security options the container should be run with. If set, the fields of SecurityContext override the equivalent fields of PodSecurityContext. More info: <https://kubernetes.io/docs/tasks/configure-pod-container/security-context/>

*SecurityContext holds security configuration that will be applied to a container. Some fields are present in both SecurityContext and PodSecurityContext. When both are set, the values in SecurityContext take precedence.*

- **securityContext.runAsUser** (int64)

The UID to run the entrypoint of the container process.

Defaults to user specified in image metadata if unspecified. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.runAsNonRoot** (boolean)

Indicates that the container must run as a non-root user. If true, the Kubelet will validate the image at runtime to ensure that it does not run as UID 0 (root) and fail to start the container if it does. If unset or false, no such validation will be performed. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.

- **securityContext.runAsGroup** (int64)

The GID to run the entrypoint of the container process. Uses runtime default if unset. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.readOnlyRootFilesystem** (boolean)

Whether this container has a read-only root filesystem. Default is false. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.procMount** (string)

procMount denotes the type of proc mount to use for the containers. The default is DefaultProcMount which uses the container runtime defaults for readonly paths and masked paths. This requires the ProcMountType feature flag to be enabled. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.privileged** (boolean)

Run container in privileged mode. Processes in privileged containers are essentially equivalent to root on the host. Defaults to false. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.allowPrivilegeEscalation** (boolean)

AllowPrivilegeEscalation controls whether a process can gain more privileges than its parent process. This bool directly controls if the no\_new\_privs flag will be set on the container process. AllowPrivilegeEscalation is true always when the container is: 1) run as Privileged 2) has CAP\_SYS\_ADMIN Note that this field cannot be set when spec.os.name is windows.

- **securityContext.capabilities** (Capabilities)

The capabilities to add/drop when running containers. Defaults to the default set of capabilities granted by the container runtime. Note that this field cannot be set when spec.os.name is windows.

*Adds and removes POSIX capabilities from running containers.*

- **securityContext.capabilities.add** ([]string)

Added capabilities

- **securityContext.capabilities.drop** ([]string)

Removed capabilities

- **securityContext.seccompProfile** (SeccompProfile)

The seccomp options to use by this container. If seccomp options are provided at both the pod & container level, the container options override the pod options. Note that this field cannot be set when spec.os.name is windows.

*SeccompProfile defines a pod/container's seccomp profile settings. Only one profile source may be set.*

- **securityContext.seccompProfile.type** (string), required

type indicates which kind of seccomp profile will be applied. Valid options are:

Localhost - a profile defined in a file on the node should be used. RuntimeDefault - the container runtime default profile should be used. Unconfined - no profile should be applied.

- **securityContext.seccompProfile.localhostProfile** (string)

localhostProfile indicates a profile defined in a file on the node should be used. The profile must be preconfigured on the node to work. Must be a descending path, relative to the kubelet's configured seccomp profile location.

Must be set if type is "Localhost". Must NOT be set for any other type.

- **securityContext.seLinuxOptions** (SELinuxOptions)

The SELinux context to be applied to the container. If unspecified, the container runtime will allocate a random SELinux context for each container. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence. Note that this field cannot be set when spec.os.name is windows.

*SELinuxOptions are the labels to be applied to the container*

- **securityContext.seLinuxOptions.level** (string)

Level is SELinux level label that applies to the container.

- **securityContext.seLinuxOptions.role** (string)

Role is a SELinux role label that applies to the container.

- **securityContext.seLinuxOptions.type** (string)

Type is a SELinux type label that applies to the container.

- **securityContext.seLinuxOptions.user** (string)

User is a SELinux user label that applies to the container.

- **securityContext.windowsOptions** (WindowsSecurityContextOptions)

The Windows specific settings applied to all containers. If unspecified, the options from the PodSecurityContext will be

used. If set in both `SecurityContext` and `PodSecurityContext`, the value specified in `SecurityContext` takes precedence. Note that this field cannot be set when `spec.os.name` is `linux`.

*WindowsSecurityContextOptions* contain Windows-specific options and credentials.

- **securityContext.windowsOptions.gmsaCredentialSpec**  
(string)

`GMSACredentialSpec` is where the GMSA admission webhook (<https://github.com/kubernetes-sigs/windows-gmsa>) inlines the contents of the GMSA credential spec named by the `GMSACredentialSpecName` field.

- **securityContext.windowsOptions.gmsaCredentialSpecName**  
(string)

`GMSACredentialSpecName` is the name of the GMSA credential spec to use.

- **securityContext.windowsOptions.hostProcess**  
(boolean)

`HostProcess` determines if a container should be run as a 'Host Process' container. All of a Pod's containers must have the same effective `HostProcess` value (it is not allowed to have a mix of `HostProcess` containers and non-`HostProcess` containers). In addition, if `HostProcess` is true then `HostNetwork` must also be set to true.

- **securityContext.windowsOptions.runAsUserName**  
(string)

The `UserName` in Windows to run the entrypoint of the container process. Defaults to the user specified in image metadata if unspecified. May also be set in `PodSecurityContext`. If set in both `SecurityContext` and `PodSecurityContext`, the value specified in `SecurityContext` takes precedence.

## Debugging

- **stdin** (boolean)

Whether this container should allocate a buffer for `stdin` in the container runtime. If this is not set, reads from `stdin` in the container will always result in EOF. Default is false.

- **stdinOnce** (boolean)

Whether the container runtime should close the `stdin` channel after it has been opened by a single attach. When `stdin` is true the `stdin` stream will remain open across multiple attach sessions. If `stdinOnce` is set to true, `stdin` is opened on container start, is empty until the first client attaches to `stdin`, and then remains open and accepts data until the client disconnects, at which time `stdin` is closed and remains closed until the container is restarted. If this flag is false, a container processes that reads from `stdin` will never receive an EOF. Default is false

- **tty** (boolean)

Whether this container should allocate a TTY for itself, also requires '`stdin`' to be true. Default is false.

# EphemeralContainer

An EphemeralContainer is a temporary container that you may add to an existing Pod for user-initiated activities such as debugging. Ephemeral containers have no resource or scheduling guarantees, and they will not be restarted when they exit or when a Pod is removed or restarted. The kubelet may evict a Pod if an ephemeral container causes the Pod to exceed its resource allocation.

To add an ephemeral container, use the `ephemeralcontainers` subresource of an existing Pod. Ephemeral containers may not be removed or restarted.

- **name** (string), required

Name of the ephemeral container specified as a DNS\_LABEL. This name must be unique among all containers, init containers and ephemeral containers.

- **targetContainerName** (string)

If set, the name of the container from PodSpec that this ephemeral container targets. The ephemeral container will be run in the namespaces (IPC, PID, etc) of this container. If not set then the ephemeral container uses the namespaces configured in the Pod spec.

The container runtime must implement support for this feature. If the runtime does not support namespace targeting then the result of setting this field is undefined.

## Image

- **image** (string)

Container image name. More info: <https://kubernetes.io/docs/concepts/containers/images>

- **imagePullPolicy** (string)

Image pull policy. One of Always, Never, IfNotPresent. Defaults to Always if :latest tag is specified, or IfNotPresent otherwise. Cannot be updated. More info: <https://kubernetes.io/docs/concepts/containers/images#updating-images>

## Entrypoint

- **command** ([]string)

Entrypoint array. Not executed within a shell. The image's `ENTRYPOINT` is used if this is not provided. Variable references `$(VAR_NAME)` are expanded using the container's environment. If a variable cannot be resolved, the reference in the input string will be unchanged. Double `$$` are reduced to a single `$`, which allows for escaping the `$(VAR_NAME)` syntax: i.e. `"$$(VAR_NAME)"` will produce the string literal `"$(VAR_NAME)"`. Escaped references will never be expanded, regardless of whether the variable exists or not. Cannot be updated. More info: <https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/#running-a-command-in-a-shell>

- **args** ([]string)

Arguments to the entrypoint. The image's `CMD` is used if this is not

provided. Variable references `$(VAR_NAME)` are expanded using the container's environment. If a variable cannot be resolved, the reference in the input string will be unchanged. Double `$$` are reduced to a single `$`, which allows for escaping the `$(VAR_NAME)` syntax: i.e. `"$$(VAR_NAME)"` will produce the string literal `"$(VAR_NAME)"`. Escaped references will never be expanded, regardless of whether the variable exists or not. Cannot be updated. More info: <https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/#running-a-command-in-a-shell>

- **workingDir** (string)

Container's working directory. If not specified, the container runtime's default will be used, which might be configured in the container image. Cannot be updated.

## Environment variables

- **env** ([]EnvVar)

*Patch strategy: merge on key name*

List of environment variables to set in the container. Cannot be updated.

*EnvVar represents an environment variable present in a Container.*

- **env.name** (string), required

Name of the environment variable. Must be a C\_IDENTIFIER.

- **env.value** (string)

Variable references `$(VAR_NAME)` are expanded using the previously defined environment variables in the container and any service environment variables. If a variable cannot be resolved, the reference in the input string will be unchanged. Double `$$` are reduced to a single `$`, which allows for escaping the `$(VAR_NAME)` syntax: i.e. `"$$(VAR_NAME)"` will produce the string literal `"$(VAR_NAME)"`. Escaped references will never be expanded, regardless of whether the variable exists or not. Defaults to `""`.

- **env.valueFrom** (EnvVarSource)

Source for the environment variable's value. Cannot be used if value is not empty.

*EnvVarSource represents a source for the value of an EnvVar.*

- **env.valueFrom.configMapKeyRef**

(ConfigMapKeySelector)

Selects a key of a ConfigMap.

*Selects a key from a ConfigMap.*

- **env.valueFrom.configMapKeyRef.key** (string), required

The key to select.

- **env.valueFrom.configMapKeyRef.name** (string)

Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **env.valueFrom.configMapKeyRef.optional** (boolean)

Specify whether the ConfigMap or its key must be defined

- **env.valueFrom.fieldRef** ([ObjectFieldSelector](#))

Selects a field of the pod: supports metadata.name, metadata.namespace, metadata.labels['<KEY>'], metadata.annotations['<KEY>'], spec.nodeName, spec.serviceAccountName, status.hostIP, status.podIP, status.podIPs.

- **env.valueFrom.resourceFieldRef** ([ResourceFieldSelector](#))

Selects a resource of the container: only resources limits and requests (limits.cpu, limits.memory, limits.ephemeral-storage, requests.cpu, requests.memory and requests.ephemeral-storage) are currently supported.

- **env.valueFrom.secretKeyRef** ([SecretKeySelector](#))

Selects a key of a secret in the pod's namespace

*SecretKeySelector* selects a key of a Secret.

- **env.valueFrom.secretKeyRef.key** (string), required

The key of the secret to select from. Must be a valid secret key.

- **env.valueFrom.secretKeyRef.name** (string)

Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **env.valueFrom.secretKeyRef.optional** (boolean)

Specify whether the Secret or its key must be defined

- **envFrom** ([]EnvFromSource)

List of sources to populate environment variables in the container. The keys defined within a source must be a C\_IDENTIFIER. All invalid keys will be reported as an event when the container is starting. When a key exists in multiple sources, the value associated with the last source will take precedence. Values defined by an Env with a duplicate key will take precedence. Cannot be updated.

*EnvFromSource* represents the source of a set of ConfigMaps

- **envFrom.configMapRef** ([ConfigMapEnvSource](#))

The ConfigMap to select from

\*ConfigMapEnvSource selects a ConfigMap to populate the environment variables with.

The contents of the target ConfigMap's Data field will represent the key-value pairs as environment variables.\*

- **envFrom.configMapRef.name** (string)

Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **envFrom.configMapRef.optional** (boolean)

Specify whether the ConfigMap must be defined

- **envFrom.prefix** (string)

An optional identifier to prepend to each key in the ConfigMap. Must be a C\_IDENTIFIER.

- **envFrom.secretRef** (SecretEnvSource)

## The Secret to select from

\*SecretEnvSource selects a Secret to populate the environment variables with.

The contents of the target Secret's Data field will represent the key-value pairs as environment variables.\*

- **envFrom.secretRef.name** (string)

Name of the referent. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **envFrom.secretRef.optional** (boolean)

Specify whether the Secret must be defined

## Volumes

- **volumeMounts** ([]VolumeMount)

*Patch strategy: merge on key `mountPath`*

Pod volumes to mount into the container's filesystem. Subpath mounts are not allowed for ephemeral containers. Cannot be updated.

*VolumeMount describes a mounting of a Volume within a container.*

- **volumeMounts.mountPath** (string), required

- **volumeMounts.name** (string), required

This must match the Name of a Volume.

- **volumeMounts.mountPropagation** (string)

mountPropagation determines how mounts are propagated from the host to container and the other way around. When not set, MountPropagationNone is used. This field is beta in 1.10.

- **volumeMounts.readOnly** (boolean)

Mounted read-only if true, read-write otherwise (false or unspecified). Defaults to false.

- **volumeMounts.subPath** (string)

Path within the volume from which the container's volume should be mounted. Defaults to "" (volume's root).

- **volumeMounts.subPathExpr** (string)

Expanded path within the volume from which the container's volume should be mounted. Behaves similarly to SubPath but environment variable references `$(VAR_NAME)` are expanded using the container's environment. Defaults to `""` (volume's root). SubPathExpr and SubPath are mutually exclusive.

- **volumeDevices** ([]VolumeDevice)

*Patch strategy: merge on key `devicePath`*

volumeDevices is the list of block devices to be used by the container.

*volumeDevice describes a mapping of a raw block device within a container.*

- **volumeDevices.devicePath** (string), required

devicePath is the path inside of the container that the device will be mapped to.

- **volumeDevices.name** (string), required

name must match the name of a persistentVolumeClaim in the pod

## Resources

- **resizePolicy** ([]ContainerResizePolicy)

*Atomic: will be replaced during a merge*

Resources resize policy for the container.

*ContainerResizePolicy represents resource resize policy for the container.*

- **resizePolicy.resourceName** (string), required

Name of the resource to which this resource resize policy applies. Supported values: `cpu`, `memory`.

- **resizePolicy.restartPolicy** (string), required

Restart policy to apply when specified resource is resized. If not specified, it defaults to `NotRequired`.

## Lifecycle

- **terminationMessagePath** (string)

Optional: Path at which the file to which the container's termination message will be written is mounted into the container's filesystem. Message written is intended to be brief final status, such as an assertion failure message. Will be truncated by the node if greater than 4096 bytes. The total message length across all containers will be limited to 12kb. Defaults to `/dev/termination-log`. Cannot be updated.

- **terminationMessagePolicy** (string)

Indicate how the termination message should be populated. File will use the contents of `terminationMessagePath` to populate the container status message on both success and failure. `FallbackToLogsOnError` will use the last chunk of container log output if the termination message file is empty and the container

exited with an error. The log output is limited to 2048 bytes or 80 lines, whichever is smaller. Defaults to File. Cannot be updated.

- **restartPolicy** (string)

Restart policy for the container to manage the restart behavior of each container within a pod. This may only be set for init containers. You cannot set this field on ephemeral containers.

## Debugging

- **stdin** (boolean)

Whether this container should allocate a buffer for stdin in the container runtime. If this is not set, reads from stdin in the container will always result in EOF. Default is false.

- **stdinOnce** (boolean)

Whether the container runtime should close the stdin channel after it has been opened by a single attach. When stdin is true the stdin stream will remain open across multiple attach sessions. If stdinOnce is set to true, stdin is opened on container start, is empty until the first client attaches to stdin, and then remains open and accepts data until the client disconnects, at which time stdin is closed and remains closed until the container is restarted. If this flag is false, a container processes that reads from stdin will never receive an EOF. Default is false

- **tty** (boolean)

Whether this container should allocate a TTY for itself, also requires 'stdin' to be true. Default is false.

## Security context

- **securityContext** (SecurityContext)

Optional: SecurityContext defines the security options the ephemeral container should be run with. If set, the fields of SecurityContext override the equivalent fields of PodSecurityContext.

*SecurityContext holds security configuration that will be applied to a container. Some fields are present in both SecurityContext and PodSecurityContext. When both are set, the values in SecurityContext take precedence.*

- **securityContext.runAsUser** (int64)

The UID to run the entrypoint of the container process. Defaults to user specified in image metadata if unspecified. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.runAsNonRoot** (boolean)

Indicates that the container must run as a non-root user. If true, the Kubelet will validate the image at runtime to ensure that it does not run as UID 0 (root) and fail to start the container if it does. If unset or false, no such validation will be performed. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.

- **securityContext.runAsGroup** (int64)

The GID to run the entrypoint of the container process. Uses runtime default if unset. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.readOnlyRootFilesystem** (boolean)

Whether this container has a read-only root filesystem. Default is false. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.procMount** (string)

procMount denotes the type of proc mount to use for the containers. The default is DefaultProcMount which uses the container runtime defaults for readonly paths and masked paths. This requires the ProcMountType feature flag to be enabled. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.privileged** (boolean)

Run container in privileged mode. Processes in privileged containers are essentially equivalent to root on the host. Defaults to false. Note that this field cannot be set when spec.os.name is windows.

- **securityContext.allowPrivilegeEscalation** (boolean)

AllowPrivilegeEscalation controls whether a process can gain more privileges than its parent process. This bool directly controls if the no\_new\_privs flag will be set on the container process. AllowPrivilegeEscalation is true always when the container is: 1) run as Privileged 2) has CAP\_SYS\_ADMIN Note that this field cannot be set when spec.os.name is windows.

- **securityContext.capabilities** (Capabilities)

The capabilities to add/drop when running containers. Defaults to the default set of capabilities granted by the container runtime. Note that this field cannot be set when spec.os.name is windows.

*Adds and removes POSIX capabilities from running containers.*

- **securityContext.capabilities.add** ([]string)

Added capabilities

- **securityContext.capabilities.drop** ([]string)

Removed capabilities

- **securityContext.seccompProfile** (SeccompProfile)

The seccomp options to use by this container. If seccomp options are provided at both the pod & container level, the container options override the pod options. Note that this field cannot be set when spec.os.name is windows.

*SeccompProfile defines a pod/container's seccomp profile settings. Only one profile source may be set.*

- **securityContext.seccompProfile.type** (string), required

type indicates which kind of seccomp profile will be applied. Valid options are:

Localhost - a profile defined in a file on the node should be used. RuntimeDefault - the container runtime default profile should be used. Unconfined - no profile should be applied.

- **securityContext.seccompProfile.localhostProfile**

(string)

localhostProfile indicates a profile defined in a file on the node should be used. The profile must be preconfigured on the node to work. Must be a descending path, relative to the kubelet's configured seccomp profile location.

Must be set if type is "Localhost". Must NOT be set for any other type.

- **securityContext.seLinuxOptions** (SELinuxOptions)

The SELinux context to be applied to the container. If unspecified, the container runtime will allocate a random SELinux context for each container. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence. Note that this field cannot be set when spec.os.name is windows.

*SELinuxOptions are the labels to be applied to the container*

- **securityContext.seLinuxOptions.level** (string)

Level is SELinux level label that applies to the container.

- **securityContext.seLinuxOptions.role** (string)

Role is a SELinux role label that applies to the container.

- **securityContext.seLinuxOptions.type** (string)

Type is a SELinux type label that applies to the container.

- **securityContext.seLinuxOptions.user** (string)

User is a SELinux user label that applies to the container.

- **securityContext.windowsOptions**

(WindowsSecurityContextOptions)

The Windows specific settings applied to all containers. If unspecified, the options from the PodSecurityContext will be used. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence. Note that this field cannot be set when spec.os.name is linux.

*WindowsSecurityContextOptions contain Windows-specific options and credentials.*

- **securityContext.windowsOptions.gmsaCredentialSpec**  
(string)

GMSACredentialSpec is where the GMSA admission webhook (<https://github.com/kubernetes-sigs/windows-gmsa>) inlines the contents of the GMSA credential spec named by the GMSACredentialSpecName field.

- **securityContext.windowsOptions.gmsaCredentialSpecName**  
(string)

GMSACredentialSpecName is the name of the GMSA credential spec to use.

- **securityContext.windowsOptions.hostProcess**

(boolean)

HostProcess determines if a container should be run as a 'Host Process' container. All of a Pod's containers must have the same effective HostProcess value (it is not allowed to have a mix of HostProcess containers and non-HostProcess containers). In addition, if HostProcess is true then HostNetwork must also be set to true.

- **securityContext.windowsOptions.runAsUserName**

(string)

The UserName in Windows to run the entrypoint of the container process. Defaults to the user specified in image metadata if unspecified. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.

## Not allowed

- **ports ([]ContainerPort)**

*Patch strategy: merge on key containerPort*

*Map: unique values on keys containerPort, protocol will be kept during a merge*

Ports are not allowed for ephemeral containers.

*ContainerPort represents a network port in a single container.*

- **ports.containerPort** (int32), required

Number of port to expose on the pod's IP address. This must be a valid port number, 0 < x < 65536.

- **ports.hostIP** (string)

What host IP to bind the external port to.

- **ports.hostPort** (int32)

Number of port to expose on the host. If specified, this must be a valid port number, 0 < x < 65536. If HostNetwork is specified, this must match ContainerPort. Most containers do not need this.

- **ports.name** (string)

If specified, this must be an IANA\_SVC\_NAME and unique within the pod. Each named port in a pod must have a unique name. Name for the port that can be referred to by services.

- **ports.protocol** (string)

Protocol for port. Must be UDP, TCP, or SCTP. Defaults to "TCP".

- **resources (ResourceRequirements)**

Resources are not allowed for ephemeral containers. Ephemeral containers use spare resources already allocated to the pod.

*ResourceRequirements describes the compute resource requirements.*

- **resources.claims** ([]ResourceClaim)

*Map: unique values on key name will be kept during a merge*

Claims lists the names of resources, defined in spec.resourceClaims, that are used by this container.

This is an alpha field and requires enabling the DynamicResourceAllocation feature gate.

This field is immutable. It can only be set for containers.

*ResourceClaim references one entry in PodSpec.ResourceClaims.*

- **resources.claims.name** (string), required

Name must match the name of one entry in pod.spec.resourceClaims of the Pod where this field is used. It makes that resource available inside a container.

- **resources.limits** (map[string][Quantity](#))

Limits describes the maximum amount of compute resources allowed. More info: <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

- **resources.requests** (map[string][Quantity](#))

Requests describes the minimum amount of compute resources required. If Requests is omitted for a container, it defaults to Limits if that is explicitly specified, otherwise to an implementation-defined value. Requests cannot exceed Limits. More info: <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

- **lifecycle** (Lifecycle)

Lifecycle is not allowed for ephemeral containers.

*Lifecycle describes actions that the management system should take in response to container lifecycle events. For the PostStart and PreStop lifecycle handlers, management of the container blocks until the action is complete, unless the container process fails, in which case the handler is aborted.*

- **lifecycle.postStart** ([LifecycleHandler](#))

PostStart is called immediately after a container is created. If the handler fails, the container is terminated and restarted according to its restart policy. Other management of the container blocks until the hook completes. More info: <https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks/#container-hooks>

- **lifecycle.preStop** ([LifecycleHandler](#))

PreStop is called immediately before a container is terminated due to an API request or management event such as liveness/startup probe failure, preemption, resource contention, etc. The handler is not called if the container crashes or exits. The Pod's termination grace period countdown begins before the PreStop hook is executed. Regardless of the outcome of the handler, the container will eventually terminate within the Pod's termination grace period (unless delayed by finalizers). Other management of the container blocks until the hook completes or until the termination grace period is reached.

More info: <https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks/#container-hooks>

- **livenessProbe** ([Probe](#))

Probes are not allowed for ephemeral containers.

- **readinessProbe** ([Probe](#))

Probes are not allowed for ephemeral containers.

- **startupProbe** ([Probe](#))

Probes are not allowed for ephemeral containers.

## LifecycleHandler

LifecycleHandler defines a specific action that should be taken in a lifecycle hook. One and only one of the fields, except TCPSocket must be specified.

- **exec** ([ExecAction](#))

Exec specifies the action to take.

*ExecAction describes a "run in container" action.*

- **exec.command** ([\[\]string](#))

Command is the command line to execute inside the container, the working directory for the command is root ('/') in the container's filesystem. The command is simply exec'd, it is not run inside a shell, so traditional shell instructions ('|', etc) won't work. To use a shell, you need to explicitly call out to that shell. Exit status of 0 is treated as live/healthy and non-zero is unhealthy.

- **httpGet** ([HTTPGetAction](#))

HTTPGet specifies the http request to perform.

*HTTPGetAction describes an action based on HTTP Get requests.*

- **httpGet.port** ([IntOrString](#)), required

Name or number of the port to access on the container. Number must be in the range 1 to 65535. Name must be an IANA\_SVC\_NAME.

*IntOrString is a type that can hold an int32 or a string. When used in JSON or YAML marshalling and unmarshalling, it produces or consumes the inner type. This allows you to have, for example, a JSON field that can accept a name or number.*

- **httpGet.host** ([string](#))

Host name to connect to, defaults to the pod IP. You probably want to set "Host" in httpHeaders instead.

- **httpGet.httpHeaders** ([\[\]HTTPHeader](#))

Custom headers to set in the request. HTTP allows repeated headers.

*HTTPHeader describes a custom header to be used in HTTP probes*

- **httpGet.httpHeaders.name** (string), required

The header field name. This will be canonicalized upon output, so case-variant names will be understood as the same header.

- **httpGet.httpHeaders.value** (string), required

The header field value

- **httpGet.path** (string)

Path to access on the HTTP server.

- **httpGet.scheme** (string)

Scheme to use for connecting to the host. Defaults to HTTP.

- **tcpSocket** (TCPSocketAction)

Deprecated. TCPSocket is NOT supported as a LifecycleHandler and kept for the backward compatibility. There are no validation of this field and lifecycle hooks will fail in runtime when tcp handler is specified.

*TCPSocketAction describes an action based on opening a socket*

- **tcpSocket.port** (IntOrString), required

Number or name of the port to access on the container.

Number must be in the range 1 to 65535. Name must be an IANA\_SVC\_NAME.

*IntOrString is a type that can hold an int32 or a string. When used in JSON or YAML marshalling and unmarshalling, it produces or consumes the inner type. This allows you to have, for example, a JSON field that can accept a name or number.*

- **tcpSocket.host** (string)

Optional: Host name to connect to, defaults to the pod IP.

## NodeAffinity

Node affinity is a group of node affinity scheduling rules.

- **preferredDuringSchedulingIgnoredDuringExecution**

([]PreferredSchedulingTerm)

The scheduler will prefer to schedule pods to nodes that satisfy the affinity expressions specified by this field, but it may choose a node that violates one or more of the expressions. The node that is most preferred is the one with the greatest sum of weights, i.e. for each node that meets all of the scheduling requirements (resource request, requiredDuringScheduling affinity expressions, etc.), compute a sum by iterating through the elements of this field and adding "weight" to the sum if the node matches the corresponding matchExpressions; the node(s) with the highest sum are the most preferred.

*An empty preferred scheduling term matches all objects with implicit weight 0 (i.e. it's a no-op). A null preferred scheduling term matches no objects (i.e. is also a no-op).*

- **preferredDuringSchedulingIgnoredDuringExecution.preference**

(NodeSelectorTerm), required

A node selector term, associated with the corresponding weight.

*A null or empty node selector term matches no objects. The requirements of them are ANDed. The TopologySelectorTerm type implements a subset of the NodeSelectorTerm.*

- **preferredDuringSchedulingIgnoredDuringExecution.preference.matchExpressions**  
([][NodeSelectorRequirement](#))

A list of node selector requirements by node's labels.

- **preferredDuringSchedulingIgnoredDuringExecution.preference.matchFields**  
([][NodeSelectorRequirement](#))

A list of node selector requirements by node's fields.

- **preferredDuringSchedulingIgnoredDuringExecution.weight**  
(int32), required

Weight associated with matching the corresponding nodeSelectorTerm, in the range 1-100.

- **requiredDuringSchedulingIgnoredDuringExecution**  
(NodeSelector)

If the affinity requirements specified by this field are not met at scheduling time, the pod will not be scheduled onto the node. If the affinity requirements specified by this field cease to be met at some point during pod execution (e.g. due to an update), the system may or may not try to eventually evict the pod from its node.

*A node selector represents the union of the results of one or more label queries over a set of nodes; that is, it represents the OR of the selectors represented by the node selector terms.*

- **requiredDuringSchedulingIgnoredDuringExecution.nodeSelectorTerms**  
([][NodeSelectorTerm](#)), required

Required. A list of node selector terms. The terms are ORed.

*A null or empty node selector term matches no objects. The requirements of them are ANDed. The TopologySelectorTerm type implements a subset of the NodeSelectorTerm.*

- **requiredDuringSchedulingIgnoredDuringExecution.nodeSelectorTerms.matchExpressions**  
([][NodeSelectorRequirement](#))

A list of node selector requirements by node's labels.

- **requiredDuringSchedulingIgnoredDuringExecution.nodeSelectorTerms.matchFields**  
([][NodeSelectorRequirement](#))

A list of node selector requirements by node's fields.

## PodAffinity

Pod affinity is a group of inter pod affinity scheduling rules.

- **preferredDuringSchedulingIgnoredDuringExecution**  
([][WeightedPodAffinityTerm](#))

The scheduler will prefer to schedule pods to nodes that satisfy the affinity expressions specified by this field, but it may choose a node that violates one or more of the expressions. The node that is most preferred is the one with the greatest sum of weights, i.e. for each

node that meets all of the scheduling requirements (resource request, requiredDuringScheduling affinity expressions, etc.), compute a sum by iterating through the elements of this field and adding "weight" to the sum if the node has pods which matches the corresponding podAffinityTerm; the node(s) with the highest sum are the most preferred.

*The weights of all of the matched WeightedPodAffinityTerm fields are added per-node to find the most preferred node(s)*

- **preferredDuringSchedulingIgnoredDuringExecution.podAffinityTerm**

(PodAffinityTerm), required

Required. A pod affinity term, associated with the corresponding weight.

*Defines a set of pods (namely those matching the labelSelector relative to the given namespace(s)) that this pod should be co-located (affinity) or not co-located (anti-affinity) with, where co-located is defined as running on a node whose value of the label with key matches that of any node on which a pod of the set of pods is running*

- **preferredDuringSchedulingIgnoredDuringExecution.podAffinityTerm.topologyKey**

(string), required

This pod should be co-located (affinity) or not co-located (anti-affinity) with the pods matching the labelSelector in the specified namespaces, where co-located is defined as running on a node whose value of the label with key topologyKey matches that of any node on which any of the selected pods is running. Empty topologyKey is not allowed.

- **preferredDuringSchedulingIgnoredDuringExecution.podAffinityTerm.labelSelector**

([LabelSelector](#))

A label query over a set of resources, in this case pods.

- **preferredDuringSchedulingIgnoredDuringExecution.podAffinityTerm.namespaceSelector**

([LabelSelector](#))

A label query over the set of namespaces that the term applies to. The term is applied to the union of the namespaces selected by this field and the ones listed in the namespaces field. null selector and null or empty namespaces list means "this pod's namespace". An empty selector ({} ) matches all namespaces.

- **preferredDuringSchedulingIgnoredDuringExecution.podAffinityTerm.namespaces**

([]string)

namespaces specifies a static list of namespace names that the term applies to. The term is applied to the union of the namespaces listed in this field and the ones selected by namespaceSelector. null or empty namespaces list and null namespaceSelector means "this pod's namespace".

- **preferredDuringSchedulingIgnoredDuringExecution.weight**

(int32), required

weight associated with matching the corresponding podAffinityTerm, in the range 1-100.

- **requiredDuringSchedulingIgnoredDuringExecution**

([]PodAffinityTerm)

If the affinity requirements specified by this field are not met at scheduling time, the pod will not be scheduled onto the node. If the affinity requirements specified by this field cease to be met at some point during pod execution (e.g. due to a pod label update), the system may or may not try to eventually evict the pod from its node. When there are multiple elements, the lists of nodes corresponding to each podAffinityTerm are intersected, i.e. all terms must be satisfied.

*Defines a set of pods (namely those matching the labelSelector relative to the given namespace(s)) that this pod should be co-located (affinity) or not co-located (anti-affinity) with, where co-located is defined as running on a node whose value of the label with key matches that of any node on which a pod of the set of pods is running*

- **requiredDuringSchedulingIgnoredDuringExecution.topologyKey**  
(string), required

This pod should be co-located (affinity) or not co-located (anti-affinity) with the pods matching the labelSelector in the specified namespaces, where co-located is defined as running on a node whose value of the label with key topologyKey matches that of any node on which any of the selected pods is running. Empty topologyKey is not allowed.

- **requiredDuringSchedulingIgnoredDuringExecution.labelSelector**  
([LabelSelector](#))

A label query over a set of resources, in this case pods.

- **requiredDuringSchedulingIgnoredDuringExecution.namespaceSelector**  
([LabelSelector](#))

A label query over the set of namespaces that the term applies to. The term is applied to the union of the namespaces selected by this field and the ones listed in the namespaces field. null selector and null or empty namespaces list means "this pod's namespace". An empty selector ({}) matches all namespaces.

- **requiredDuringSchedulingIgnoredDuringExecution.namespaces**  
([]string)

namespaces specifies a static list of namespace names that the term applies to. The term is applied to the union of the namespaces listed in this field and the ones selected by namespaceSelector. null or empty namespaces list and null namespaceSelector means "this pod's namespace".

## PodAntiAffinity

Pod anti affinity is a group of inter pod anti affinity scheduling rules.

- **preferredDuringSchedulingIgnoredDuringExecution**  
([]WeightedPodAffinityTerm)

The scheduler will prefer to schedule pods to nodes that satisfy the anti-affinity expressions specified by this field, but it may choose a node that violates one or more of the expressions. The node that is most preferred is the one with the greatest sum of weights, i.e. for each node that meets all of the scheduling requirements (resource request, requiredDuringScheduling anti-affinity expressions, etc.), compute a sum by iterating through the elements of this field and

adding "weight" to the sum if the node has pods which matches the corresponding podAffinityTerm; the node(s) with the highest sum are the most preferred.

*The weights of all of the matched WeightedPodAffinityTerm fields are added per-node to find the most preferred node(s)*

- **preferredDuringSchedulingIgnoredDuringExecution.podAffinityTerm**  
(PodAffinityTerm), required

Required. A pod affinity term, associated with the corresponding weight.

*Defines a set of pods (namely those matching the labelSelector relative to the given namespace(s)) that this pod should be co-located (affinity) or not co-located (anti-affinity) with, where co-located is defined as running on a node whose value of the label with key matches that of any node on which a pod of the set of pods is running*

- **preferredDuringSchedulingIgnoredDuringExecution.podAffinityTerm.topologyKey**  
(string), required

This pod should be co-located (affinity) or not co-located (anti-affinity) with the pods matching the labelSelector in the specified namespaces, where co-located is defined as running on a node whose value of the label with key topologyKey matches that of any node on which any of the selected pods is running. Empty topologyKey is not allowed.

- **preferredDuringSchedulingIgnoredDuringExecution.podAffinityTerm.labelSelector**  
([LabelSelector](#))

A label query over a set of resources, in this case pods.

- **preferredDuringSchedulingIgnoredDuringExecution.podAffinityTerm.namespaceSelector**  
([LabelSelector](#))

A label query over the set of namespaces that the term applies to. The term is applied to the union of the namespaces selected by this field and the ones listed in the namespaces field. null selector and null or empty namespaces list means "this pod's namespace". An empty selector ({} ) matches all namespaces.

- **preferredDuringSchedulingIgnoredDuringExecution.podAffinityTerm.namespaces**  
([]string)

namespaces specifies a static list of namespace names that the term applies to. The term is applied to the union of the namespaces listed in this field and the ones selected by namespaceSelector. null or empty namespaces list and null namespaceSelector means "this pod's namespace".

- **preferredDuringSchedulingIgnoredDuringExecution.weight**  
(int32), required

weight associated with matching the corresponding podAffinityTerm, in the range 1-100.

- **requiredDuringSchedulingIgnoredDuringExecution**  
([]PodAffinityTerm)

If the anti-affinity requirements specified by this field are not met at scheduling time, the pod will not be scheduled onto the node. If the

anti-affinity requirements specified by this field cease to be met at some point during pod execution (e.g. due to a pod label update), the system may or may not try to eventually evict the pod from its node. When there are multiple elements, the lists of nodes corresponding to each podAffinityTerm are intersected, i.e. all terms must be satisfied.

*Defines a set of pods (namely those matching the labelSelector relative to the given namespace(s)) that this pod should be co-located (affinity) or not co-located (anti-affinity) with, where co-located is defined as running on a node whose value of the label with key matches that of any node on which a pod of the set of pods is running*

- **requiredDuringSchedulingIgnoredDuringExecution.topologyKey** (string), required

This pod should be co-located (affinity) or not co-located (anti-affinity) with the pods matching the labelSelector in the specified namespaces, where co-located is defined as running on a node whose value of the label with key topologyKey matches that of any node on which any of the selected pods is running. Empty topologyKey is not allowed.

- **requiredDuringSchedulingIgnoredDuringExecution.labelSelector** ([LabelSelector](#))

A label query over a set of resources, in this case pods.

- **requiredDuringSchedulingIgnoredDuringExecution.namespaceSelector** ([LabelSelector](#))

A label query over the set of namespaces that the term applies to. The term is applied to the union of the namespaces selected by this field and the ones listed in the namespaces field. null selector and null or empty namespaces list means "this pod's namespace". An empty selector ({} ) matches all namespaces.

- **requiredDuringSchedulingIgnoredDuringExecution.namespaces** ([]string)

namespaces specifies a static list of namespace names that the term applies to. The term is applied to the union of the namespaces listed in this field and the ones selected by namespaceSelector. null or empty namespaces list and null namespaceSelector means "this pod's namespace".

## Probe

Probe describes a health check to be performed against a container to determine whether it is alive or ready to receive traffic.

- **exec** (ExecAction)

Exec specifies the action to take.

*ExecAction describes a "run in container" action.*

- **exec.command** ([]string)

Command is the command line to execute inside the container, the working directory for the command is root ('/') in the container's filesystem. The command is simply exec'd, it is not run inside a shell, so traditional shell instructions ('|',

etc) won't work. To use a shell, you need to explicitly call out to that shell. Exit status of 0 is treated as live/healthy and non-zero is unhealthy.

- **httpGet** (HTTPGetAction)

HTTPGet specifies the http request to perform.

*HTTPGetAction describes an action based on HTTP Get requests.*

- **httpGet.port** (IntOrString), required

Name or number of the port to access on the container.

Number must be in the range 1 to 65535. Name must be an IANA\_SVC\_NAME.

*IntOrString is a type that can hold an int32 or a string. When used in JSON or YAML marshalling and unmarshalling, it produces or consumes the inner type. This allows you to have, for example, a JSON field that can accept a name or number.*

- **httpGet.host** (string)

Host name to connect to, defaults to the pod IP. You probably want to set "Host" in httpHeaders instead.

- **httpGet.httpHeaders** ([]HTTPHeader)

Custom headers to set in the request. HTTP allows repeated headers.

*HTTPHeader describes a custom header to be used in HTTP probes*

- **httpGet.httpHeaders.name** (string), required

The header field name. This will be canonicalized upon output, so case-variant names will be understood as the same header.

- **httpGet.httpHeaders.value** (string), required

The header field value

- **httpGet.path** (string)

Path to access on the HTTP server.

- **httpGet.scheme** (string)

Scheme to use for connecting to the host. Defaults to HTTP.

- **tcpSocket** (TCPSocketAction)

TCPSocket specifies an action involving a TCP port.

*TCPSocketAction describes an action based on opening a socket*

- **tcpSocket.port** (IntOrString), required

Name or number of the port to access on the container.

Number must be in the range 1 to 65535. Name must be an IANA\_SVC\_NAME.

*IntOrString is a type that can hold an int32 or a string. When used in JSON or YAML marshalling and unmarshalling, it produces or consumes the inner type. This allows you to have, for example, a JSON field that can accept a name or number.*

- **tcpSocket.host** (string)

Optional: Host name to connect to, defaults to the pod IP.

- **initialDelaySeconds** (int32)

Number of seconds after the container has started before liveness probes are initiated. More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes>

- **terminationGracePeriodSeconds** (int64)

Optional duration in seconds the pod needs to terminate gracefully upon probe failure. The grace period is the duration in seconds after the processes running in the pod are sent a termination signal and the time when the processes are forcibly halted with a kill signal. Set this value longer than the expected cleanup time for your process. If this value is nil, the pod's terminationGracePeriodSeconds will be used. Otherwise, this value overrides the value provided by the pod spec. Value must be non-negative integer. The value zero indicates stop immediately via the kill signal (no opportunity to shut down). This is a beta field and requires enabling ProbeTerminationGracePeriod feature gate. Minimum value is 1. spec.terminationGracePeriodSeconds is used if unset.

- **periodSeconds** (int32)

How often (in seconds) to perform the probe. Default to 10 seconds. Minimum value is 1.

- **timeoutSeconds** (int32)

Number of seconds after which the probe times out. Defaults to 1 second. Minimum value is 1. More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes>

- **failureThreshold** (int32)

Minimum consecutive failures for the probe to be considered failed after having succeeded. Defaults to 3. Minimum value is 1.

- **successThreshold** (int32)

Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1. Must be 1 for liveness and startup. Minimum value is 1.

- **grpc** (GRPCAction)

GRPC specifies an action involving a GRPC port.

\*\*

- **grpc.port** (int32), required

Port number of the gRPC service. Number must be in the range 1 to 65535.

- **grpc.service** (string)

Service is the name of the service to place in the gRPC HealthCheckRequest (see <https://github.com/grpc/grpc/blob/master/doc/health-checking.md>).

If this is not specified, the default behavior is defined by gRPC.

## PodStatus

PodStatus represents information about the status of a pod. Status may trail the actual state of a system, especially if the node that hosts the pod cannot contact the control plane.

- **nominatedNodeName** (string)

nominatedNodeName is set only when this pod preempts other pods on the node, but it cannot be scheduled right away as preemption victims receive their graceful termination periods. This field does not guarantee that the pod will be scheduled on this node. Scheduler may decide to place the pod elsewhere if other nodes become available sooner. Scheduler may also decide to give the resources on this node to a higher priority pod that is created after preemption. As a result, this field may be different than PodSpec.nodeName when the pod is scheduled.

- **hostIP** (string)

hostIP holds the IP address of the host to which the pod is assigned. Empty if the pod has not started yet. A pod can be assigned to a node that has a problem in kubelet which in turns mean that HostIP will not be updated even if there is a node assigned to pod

- **hostIPs** ([]HostIP)

*Patch strategy: merge on key ip*

*Atomic: will be replaced during a merge*

hostIPs holds the IP addresses allocated to the host. If this field is specified, the first entry must match the hostIP field. This list is empty if the pod has not started yet. A pod can be assigned to a node that has a problem in kubelet which in turns means that HostIPs will not be updated even if there is a node assigned to this pod.

*HostIP represents a single IP address allocated to the host.*

- **hostIPs.ip** (string)

IP is the IP address assigned to the host

- **startTime** (Time)

RFC 3339 date and time at which the object was acknowledged by the Kubelet. This is before the Kubelet pulled the container image(s) for the pod.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **phase** (string)

The phase of a Pod is a simple, high-level summary of where the Pod is in its lifecycle. The conditions array, the reason and message fields, and the individual container status arrays contain more detail about the pod's status. There are five possible phase values:

Pending: The pod has been accepted by the Kubernetes system, but one or more of the container images has not been created. This includes time before being scheduled as well as time spent downloading images over the network, which could take a while.

Running: The pod has been bound to a node, and all of the containers have been created. At least one container is still running, or is in the process of starting or restarting. Succeeded: All containers in the pod have terminated in success, and will not be

restarted. Failed: All containers in the pod have terminated, and at least one container has terminated in failure. The container either exited with non-zero status or was terminated by the system. Unknown: For some reason the state of the pod could not be obtained, typically due to an error in communicating with the host of the pod.

More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#pod-phase>

- **message** (string)

A human readable message indicating details about why the pod is in this condition.

- **reason** (string)

A brief CamelCase message indicating details about why the pod is in this state. e.g. 'Evicted'

- **podIP** (string)

podIP address allocated to the pod. Routable at least within the cluster. Empty if not yet allocated.

- **podIPs** ([]PodIP)

*Patch strategy: merge on key ip*

podIPs holds the IP addresses allocated to the pod. If this field is specified, the 0th entry must match the podIP field. Pods may be allocated at most 1 value for each of IPv4 and IPv6. This list is empty if no IPs have been allocated yet.

*PodIP represents a single IP address allocated to the pod.*

- **podIPs.ip** (string)

IP is the IP address assigned to the pod

- **conditions** ([]PodCondition)

*Patch strategy: merge on key type*

Current service state of pod. More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#pod-conditions>

*PodCondition contains details for the current condition of this pod.*

- **conditions.status** (string), required

Status is the status of the condition. Can be True, False, Unknown. More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#pod-conditions>

- **conditions.type** (string), required

Type is the type of the condition. More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#pod-conditions>

- **conditions.lastProbeTime** (Time)

Last time we probed the condition.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **conditions.lastTransitionTime** (Time)

Last time the condition transitioned from one status to another.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **conditions.message** (string)

Human-readable message indicating details about last transition.

- **conditions.reason** (string)

Unique, one-word, CamelCase reason for the condition's last transition.

- **qosClass** (string)

The Quality of Service (QOS) classification assigned to the pod based on resource requirements See PodQOSClass type for available QOS classes More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-qos/#quality-of-service-classes>

- **initContainerStatuses** ([]ContainerStatus)

The list has one entry per init container in the manifest. The most recent successful init container will have ready = true, the most recently started container will have startTime set. More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#pod-and-container-status>

*ContainerStatus contains details for the current status of this container.*

- **containerStatuses** ([]ContainerStatus)

The list has one entry per container in the manifest. More info: <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#pod-and-container-status>

*ContainerStatus contains details for the current status of this container.*

- **ephemeralContainerStatuses** ([]ContainerStatus)

Status for any ephemeral containers that have run in this pod.

*ContainerStatus contains details for the current status of this container.*

- **resourceClaimStatuses** ([]PodResourceClaimStatus)

*Patch strategies: retainKeys, merge on key name*

*Map: unique values on key name will be kept during a merge*

Status of resource claims.

*PodResourceClaimStatus is stored in the PodStatus for each PodResourceClaim which references a ResourceClaimTemplate. It stores the generated name for the corresponding ResourceClaim.*

- **resourceClaimStatuses.name** (string), required

Name uniquely identifies this resource claim inside the pod. This must match the name of an entry in pod.spec.resourceClaims, which implies that the string must be a DNS\_LABEL.

- **resourceClaimStatuses.resourceClaimName** (string)

ResourceClaimName is the name of the ResourceClaim that

was generated for the Pod in the namespace of the Pod. If this

is unset, then generating a ResourceClaim was not necessary.

The pod.spec.resourceClaims entry can be ignored in this case.

- **resize** (string)

Status of resources resize desired for pod's containers. It is empty if no resources resize is pending. Any changes to container resources will automatically set this to "Proposed"

## PodList

PodList is a list of Pods.

- **items** ([]Pod), required

List of pods. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md>

- **apiVersion** (string)

APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources>

- **kind** (string)

Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **metadata** (ListMeta)

Standard list metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

## Operations

### get read the specified Pod

#### HTTP Request

GET /api/v1/namespaces/{namespace}/pods/{name}

#### Parameters

- **name** (in path): string, required

name of the Pod

- **namespace** (in path): string, required

[namespace](#)

- **pretty** (in query): string

[pretty](#)

## Response

200 ([Pod](#)): OK

401: Unauthorized

**get** read ephemeralcontainers of the specified Pod

## HTTP Request

GET /api/v1/namespaces/{namespace}/pods/{name}/ephemeralcontainers

## Parameters

- **name** (*in path*): string, required  
name of the Pod
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([Pod](#)): OK

401: Unauthorized

**get** read log of the specified Pod

## HTTP Request

GET /api/v1/namespaces/{namespace}/pods/{name}/log

## Parameters

- **name** (*in path*): string, required  
name of the Pod
- **namespace** (*in path*): string, required  
[namespace](#)
- **container** (*in query*): string  
The container for which to stream logs. Defaults to only container if there is one container in the pod.
- **follow** (*in query*): boolean  
Follow the log stream of the pod. Defaults to false.
- **insecureSkipTLSVerifyBackend** (*in query*): boolean  
insecureSkipTLSVerifyBackend indicates that the apiserver should not confirm the validity of the serving certificate of the backend it is connecting to. This will make the HTTPS connection between the

apiserver and the backend insecure. This means the apiserver cannot verify the log data it is receiving came from the real kubelet. If the kubelet is configured to verify the apiserver's TLS credentials, it does not mean the connection to the real kubelet is vulnerable to a man in the middle attack (e.g. an attacker could not intercept the actual log data coming from the real kubelet).

- **limitBytes** (*in query*): integer

If set, the number of bytes to read from the server before terminating the log output. This may not display a complete final line of logging, and may return slightly more or slightly less than the specified limit.

- **pretty** (*in query*): string

[pretty](#)

- **previous** (*in query*): boolean

Return previous terminated container logs. Defaults to false.

- **sinceSeconds** (*in query*): integer

A relative time in seconds before the current time from which to show logs. If this value precedes the time a pod was started, only logs since the pod start will be returned. If this value is in the future, no logs will be returned. Only one of sinceSeconds or sinceTime may be specified.

- **tailLines** (*in query*): integer

If set, the number of lines from the end of the logs to show. If not specified, logs are shown from the creation of the container or sinceSeconds or sinceTime

- **timestamps** (*in query*): boolean

If true, add an RFC3339 or RFC3339Nano timestamp at the beginning of every line of log output. Defaults to false.

## Response

200 (string): OK

401: Unauthorized

## get read status of the specified Pod

### HTTP Request

GET /api/v1/namespaces/{namespace}/pods/{name}/status

### Parameters

- **name** (*in path*): string, required

name of the Pod

- **namespace** (*in path*): string, required

[namespace](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([Pod](#)): OK

401: Unauthorized

**list** list or watch objects of kind Pod

### HTTP Request

GET /api/v1/namespaces/{namespace}/pods

#### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([PodList](#)): OK

401: Unauthorized

## list list or watch objects of kind Pod

### HTTP Request

GET /api/v1/pods

### Parameters

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

### Response

200 ([PodList](#)): OK

401: Unauthorized

## create create a Pod

### HTTP Request

POST /api/v1/namespaces/{namespace}/pods

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [Pod](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([Pod](#)): OK

201 ([Pod](#)): Created

202 ([Pod](#)): Accepted

401: Unauthorized

**update** replace the specified Pod

## HTTP Request

PUT /api/v1/namespaces/{namespace}/pods/{name}

## Parameters

- **name** (*in path*): string, required  
name of the Pod
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Pod](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([Pod](#)): OK

201 ([Pod](#)): Created

401: Unauthorized

## update replace ephemeralcontainers of the specified Pod

### HTTP Request

PUT /api/v1/namespaces/{namespace}/pods/{name}/  
ephemeralcontainers

### Parameters

- **name** (*in path*): string, required  
name of the Pod
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Pod](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([Pod](#)): OK

201 ([Pod](#)): Created

401: Unauthorized

## update replace status of the specified Pod

### HTTP Request

PUT /api/v1/namespaces/{namespace}/pods/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the Pod
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Pod](#), required
- **dryRun** (*in query*): string  
[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([Pod](#)): OK

201 ([Pod](#)): Created

401: Unauthorized

**patch** partially update the specified Pod

## HTTP Request

PATCH /api/v1/namespaces/{namespace}/pods/{name}

## Parameters

- **name** (*in path*): string, required

name of the Pod

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [Patch](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([Pod](#)): OK

201 ([Pod](#)): Created

401: Unauthorized

**patch** partially update ephemeralcontainers of the specified

## Pod

### HTTP Request

PATCH /api/v1/namespaces/{namespace}/pods/{name}/  
ephemeralcontainers

### Parameters

- **name** (*in path*): string, required  
name of the Pod
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([Pod](#)): OK

201 ([Pod](#)): Created

401: Unauthorized

**patch** partially update status of the specified Pod

### HTTP Request

PATCH /api/v1/namespaces/{namespace}/pods/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the Pod
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([Pod](#)): OK

201 ([Pod](#)): Created

401: Unauthorized

## delete delete a Pod

### HTTP Request

DELETE /api/v1/namespaces/{namespace}/pods/{name}

### Parameters

- **name** (*in path*): string, required

name of the Pod

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

## Response

200 ([Pod](#)): OK

202 ([Pod](#)): Accepted

401: Unauthorized

## deletecollection delete collection of Pod

### HTTP Request

DELETE /api/v1/namespaces/{namespace}/pods

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **continue** (*in query*): string

[continue](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

### Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.2 - PodTemplate

PodTemplate describes a template for creating copies of a predefined pod.

```
apiVersion: v1

import "k8s.io/api/core/v1"
```

### PodTemplate

PodTemplate describes a template for creating copies of a predefined pod.

- **apiVersion**: v1
- **kind**: PodTemplate
- **metadata** ([ObjectMeta](#))

Standard object's metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **template** ([PodTemplateSpec](#))

Template defines the pods that will be created from this pod template. <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

### PodTemplateSpec

PodTemplateSpec describes the data a pod should have when created from a template

- **metadata** ([ObjectMeta](#))

Standard object's metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **spec** ([PodSpec](#))

Specification of the desired behavior of the pod. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

### PodTemplateList

PodTemplateList is a list of PodTemplates.

- **apiVersion**: v1
- **kind**: PodTemplateList
- **metadata** ([ListMeta](#))

Standard list metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **items** ([][PodTemplate](#)), required

List of pod templates

## Operations

---

### **get** read the specified PodTemplate

#### HTTP Request

GET /api/v1/namespaces/{namespace}/podtemplates/{name}

#### Parameters

- **name** (*in path*): string, required  
name of the PodTemplate
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

#### Response

200 ([PodTemplate](#)): OK

401: Unauthorized

### **list** list or watch objects of kind PodTemplate

#### HTTP Request

GET /api/v1/namespaces/{namespace}/podtemplates

#### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string  
[pretty](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)
- **sendInitialEvents** (*in query*): boolean  
[sendInitialEvents](#)
- **timeoutSeconds** (*in query*): integer  
[timeoutSeconds](#)
- **watch** (*in query*): boolean  
[watch](#)

## Response

200 ([PodTemplateList](#)): OK

401: Unauthorized

**list** list or watch objects of kind PodTemplate

## HTTP Request

GET /api/v1/podtemplates

## Parameters

- **allowWatchBookmarks** (*in query*): boolean  
[allowWatchBookmarks](#)
- **continue** (*in query*): string  
[continue](#)
- **fieldSelector** (*in query*): string  
[fieldSelector](#)
- **labelSelector** (*in query*): string  
[labelSelector](#)
- **limit** (*in query*): integer  
[limit](#)
- **pretty** (*in query*): string  
[pretty](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([PodTemplateList](#)): OK

401: Unauthorized

## create create a PodTemplate

### HTTP Request

POST /api/v1/namespaces/{namespace}/podtemplates

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [PodTemplate](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([PodTemplate](#)): OK

201 ([PodTemplate](#)): Created

202 ([PodTemplate](#)): Accepted

401: Unauthorized

## update replace the specified PodTemplate

### HTTP Request

PUT /api/v1/namespaces/{namespace}/podtemplates/{name}

### Parameters

- **name** (*in path*): string, required  
name of the PodTemplate
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [PodTemplate](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([PodTemplate](#)): OK

201 ([PodTemplate](#)): Created

401: Unauthorized

**patch** partially update the specified PodTemplate

## HTTP Request

PATCH /api/v1/namespaces/{namespace}/podtemplates/{name}

## Parameters

- **name** (*in path*): string, required  
name of the PodTemplate
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

[pretty](#)

## Response

200 ([PodTemplate](#)): OK

201 ([PodTemplate](#)): Created

401: Unauthorized

## delete delete a PodTemplate

### HTTP Request

DELETE /api/v1/namespaces/{namespace}/podtemplates/{name}

### Parameters

- **name** (*in path*): string, required  
[name](#)  
name of the PodTemplate
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [DeleteOptions](#)
- **dryRun** (*in query*): string  
[dryRun](#)
- **gracePeriodSeconds** (*in query*): integer  
[gracePeriodSeconds](#)
- **pretty** (*in query*): string  
[pretty](#)
- **propagationPolicy** (*in query*): string  
[propagationPolicy](#)

## Response

200 ([PodTemplate](#)): OK

202 ([PodTemplate](#)): Accepted

401: Unauthorized

## deletecollection delete collection of PodTemplate

### HTTP Request

DELETE /api/v1/namespaces/{namespace}/podtemplates

### Parameters

- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [DeleteOptions](#)

- **continue** (*in query*): string

[continue](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

## Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.3 - ReplicationController

ReplicationController represents the configuration of a replication controller.

```
apiVersion: v1

import "k8s.io/api/core/v1"
```

## ReplicationController

ReplicationController represents the configuration of a replication controller.

- **apiVersion**: v1
- **kind**: ReplicationController
- **metadata** ([ObjectMeta](#))

If the Labels of a ReplicationController are empty, they are defaulted to be the same as the Pod(s) that the replication controller manages. Standard object's metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **spec** ([ReplicationControllerSpec](#))

Spec defines the specification of the desired behavior of the replication controller. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

- **status** ([ReplicationControllerStatus](#))

Status is the most recently observed status of the replication controller. This data may be out of date by some window of time. Populated by the system. Read-only. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

## ReplicationControllerSpec

ReplicationControllerSpec is the specification of a replication controller.

- **selector** (map[string]string)

Selector is a label query over pods that should match the Replicas count. If Selector is empty, it is defaulted to the labels present on the Pod template. Label keys and values that must match in order to be controlled by this replication controller, if empty defaulted to labels on Pod template. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors>

- **template** ([PodTemplateSpec](#))

Template is the object that describes the pod that will be created if insufficient replicas are detected. This takes precedence over a TemplateRef. The only allowed template.spec.restartPolicy value is "Always". More info: <https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller#pod-template>

- **replicas** (int32)

Replicas is the number of desired replicas. This is a pointer to distinguish between explicit zero and unspecified. Defaults to 1. More info: <https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller#what-is-a-replicationcontroller>

- **minReadySeconds** (int32)

Minimum number of seconds for which a newly created pod should be ready without any of its container crashing, for it to be considered available. Defaults to 0 (pod will be considered available as soon as it is ready)

## ReplicationControllerStatus

ReplicationControllerStatus represents the current status of a replication controller.

- **replicas** (int32), required

Replicas is the most recently observed number of replicas. More info: <https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller#what-is-a-replicationcontroller>

- **availableReplicas** (int32)

The number of available replicas (ready for at least minReadySeconds) for this replication controller.

- **readyReplicas** (int32)

The number of ready replicas for this replication controller.

- **fullyLabeledReplicas** (int32)

The number of pods that have labels matching the labels of the pod template of the replication controller.

- **conditions** ([]ReplicationControllerCondition)

*Patch strategy: merge on key type*

Represents the latest available observations of a replication controller's current state.

*ReplicationControllerCondition describes the state of a replication controller at a certain point.*

- **conditions.status** (string), required

Status of the condition, one of True, False, Unknown.

- **conditions.type** (string), required

Type of replication controller condition.

- **conditions.lastTransitionTime** (Time)

The last time the condition transitioned from one status to another.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **conditions.message** (string)

A human readable message indicating details about the transition.

- **conditions.reason** (string)

The reason for the condition's last transition.

- **observedGeneration** (int64)

ObservedGeneration reflects the generation of the most recently observed replication controller.

## ReplicationControllerList

ReplicationControllerList is a collection of replication controllers.

- **apiVersion**: v1

- **kind**: ReplicationControllerList

- **metadata** ([ListMeta](#))

Standard list metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **items** ([][ReplicationController](#)), required

List of replication controllers. More info: <https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller>

## Operations

### **get** read the specified ReplicationController

#### HTTP Request

GET /api/v1/namespaces/{namespace}/replicationcontrollers/{name}

#### Parameters

- **name** (*in path*): string, required

name of the ReplicationController

- **namespace** (*in path*): string, required

[namespace](#)

- **pretty** (*in query*): string

[pretty](#)

#### Response

200 ([ReplicationController](#)): OK

401: Unauthorized

### **get** read status of the specified ReplicationController

## HTTP Request

GET /api/v1/namespaces/{namespace}/replicationcontrollers/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the ReplicationController
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([ReplicationController](#)): OK

401: Unauthorized

**list** list or watch objects of kind ReplicationController

## HTTP Request

GET /api/v1/namespaces/{namespace}/replicationcontrollers

### Parameters

- **namespace** (*in path*): string, required  
[namespace](#)
- **allowWatchBookmarks** (*in query*): boolean  
[allowWatchBookmarks](#)
- **continue** (*in query*): string  
[continue](#)
- **fieldSelector** (*in query*): string  
[fieldSelector](#)
- **labelSelector** (*in query*): string  
[labelSelector](#)
- **limit** (*in query*): integer  
[limit](#)
- **pretty** (*in query*): string  
[pretty](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([ReplicationControllerList](#)): OK

401: Unauthorized

**list** list or watch objects of kind ReplicationController

## HTTP Request

GET /api/v1/replicationcontrollers

## Parameters

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([ReplicationControllerList](#)): OK

401: Unauthorized

## create create a ReplicationController

### HTTP Request

POST /api/v1/namespaces/{namespace}/replicationcontrollers

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [ReplicationController](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([ReplicationController](#)): OK

201 ([ReplicationController](#)): Created

202 ([ReplicationController](#)): Accepted

401: Unauthorized

## update replace the specified ReplicationController

### HTTP Request

PUT /api/v1/namespaces/{namespace}/replicationcontrollers/{name}

### Parameters

- **name** (*in path*): string, required

name of the ReplicationController

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [ReplicationController](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([ReplicationController](#)): OK

201 ([ReplicationController](#)): Created

401: Unauthorized

**update** replace status of the specified ReplicationController

## HTTP Request

PUT /api/v1/namespaces/{namespace}/replicationcontrollers/{name}/status

## Parameters

- **name** (*in path*): string, required

name of the ReplicationController

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [ReplicationController](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([ReplicationController](#)): OK

201 ([ReplicationController](#)): Created

401: Unauthorized

**patch** partially update the specified ReplicationController

## HTTP Request

PATCH /api/v1/namespaces/{namespace}/replicationcontrollers/{name}

### Parameters

- **name** (*in path*): string, required  
name of the ReplicationController
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([ReplicationController](#)): OK

201 ([ReplicationController](#)): Created

401: Unauthorized

**patch** partially update status of the specified ReplicationController

## HTTP Request

PATCH /api/v1/namespaces/{namespace}/replicationcontrollers/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the ReplicationController
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([ReplicationController](#)): OK

201 ([ReplicationController](#)): Created

401: Unauthorized

## delete delete a ReplicationController

### HTTP Request

DELETE /api/v1/namespaces/{namespace}/replicationcontrollers/{name}

### Parameters

- **name** (*in path*): string, required

name of the ReplicationController

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

## Response

200 ([Status](#)): OK

202 ([Status](#)): Accepted

401: Unauthorized

## deletecollection delete collection of ReplicationController

## HTTP Request

DELETE /api/v1/namespaces/{namespace}/replicationcontrollers

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **continue** (*in query*): string

[continue](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

### Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.4 - ReplicaSet

ReplicaSet ensures that a specified number of pod replicas are running at any given time.

```
apiVersion: apps/v1

import "k8s.io/api/apps/v1"
```

## ReplicaSet

ReplicaSet ensures that a specified number of pod replicas are running at any given time.

- **apiVersion**: apps/v1

- **kind**: ReplicaSet

- **metadata** ([ObjectMeta](#))

If the Labels of a ReplicaSet are empty, they are defaulted to be the same as the Pod(s) that the ReplicaSet manages. Standard object's metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **spec** ([ReplicaSetSpec](#))

Spec defines the specification of the desired behavior of the ReplicaSet. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

- **status** ([ReplicaSetStatus](#))

Status is the most recently observed status of the ReplicaSet. This data may be out of date by some window of time. Populated by the system. Read-only. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

## ReplicaSetSpec

ReplicaSetSpec is the specification of a ReplicaSet.

- **selector** ([LabelSelector](#)), required

Selector is a label query over pods that should match the replica count. Label keys and values that must match in order to be controlled by this replica set. It must match the pod template's labels. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors>

- **template** ([PodTemplateSpec](#))

Template is the object that describes the pod that will be created if insufficient replicas are detected. More info: <https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller#pod-template>

- **replicas** (int32)

Replicas is the number of desired replicas. This is a pointer to distinguish between explicit zero and unspecified. Defaults to 1.

More info: <https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/#what-is-a-replicationcontroller>

- **minReadySeconds** (int32)

Minimum number of seconds for which a newly created pod should be ready without any of its container crashing, for it to be considered available. Defaults to 0 (pod will be considered available as soon as it is ready)

## ReplicaSetStatus

ReplicaSetStatus represents the current status of a ReplicaSet.

- **replicas** (int32), required

Replicas is the most recently observed number of replicas. More info: <https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/#what-is-a-replicationcontroller>

- **availableReplicas** (int32)

The number of available replicas (ready for at least minReadySeconds) for this replica set.

- **readyReplicas** (int32)

readyReplicas is the number of pods targeted by this ReplicaSet with a Ready Condition.

- **fullyLabeledReplicas** (int32)

The number of pods that have labels matching the labels of the pod template of the replicaset.

- **conditions** ([]ReplicaSetCondition)

*Patch strategy: merge on key type*

Represents the latest available observations of a replica set's current state.

*ReplicaSetCondition describes the state of a replica set at a certain point.*

- **conditions.status** (string), required

Status of the condition, one of True, False, Unknown.

- **conditions.type** (string), required

Type of replica set condition.

- **conditions.lastTransitionTime** (Time)

The last time the condition transitioned from one status to another.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **conditions.message** (string)

A human readable message indicating details about the transition.

- **conditions.reason** (string)

The reason for the condition's last transition.

- **observedGeneration** (int64)

ObservedGeneration reflects the generation of the most recently observed ReplicaSet.

## ReplicaSetList

ReplicaSetList is a collection of ReplicaSets.

- **apiVersion**: apps/v1
- **kind**: ReplicaSetList
- **metadata** ([ListMeta](#))

Standard list metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **items** ([][ReplicaSet](#)), required

List of ReplicaSets. More info: <https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller>

## Operations

### **get** read the specified ReplicaSet

#### HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/replicasets/{name}

#### Parameters

- **name** (*in path*): string, required  
name of the ReplicaSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

#### Response

200 ([ReplicaSet](#)): OK

401: Unauthorized

### **get** read status of the specified ReplicaSet

#### HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/replicasets/{name}/status

## Parameters

- **name** (*in path*): string, required  
name of the ReplicaSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([ReplicaSet](#)): OK

401: Unauthorized

**list** list or watch objects of kind ReplicaSet

## HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/replicasets

## Parameters

- **namespace** (*in path*): string, required  
[namespace](#)
- **allowWatchBookmarks** (*in query*): boolean  
[allowWatchBookmarks](#)
- **continue** (*in query*): string  
[continue](#)
- **fieldSelector** (*in query*): string  
[fieldSelector](#)
- **labelSelector** (*in query*): string  
[labelSelector](#)
- **limit** (*in query*): integer  
[limit](#)
- **pretty** (*in query*): string  
[pretty](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)
- **sendInitialEvents** (*in query*): boolean  
[sendInitialEvents](#)
- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([ReplicaSetList](#)): OK

401: Unauthorized

**list** list or watch objects of kind ReplicaSet

## HTTP Request

GET /apis/apps/v1/replicasets

## Parameters

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([ReplicaSetList](#)): OK

401: Unauthorized

## create create a ReplicaSet

### HTTP Request

POST /apis/apps/v1/namespaces/{namespace}/replicasets

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [ReplicaSet](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

### Response

200 ([ReplicaSet](#)): OK

201 ([ReplicaSet](#)): Created

202 ([ReplicaSet](#)): Accepted

401: Unauthorized

## update replace the specified ReplicaSet

### HTTP Request

PUT /apis/apps/v1/namespaces/{namespace}/replicasets/{name}

### Parameters

- **name** (*in path*): string, required

name of the ReplicaSet

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [ReplicaSet](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([ReplicaSet](#)): OK

201 ([ReplicaSet](#)): Created

401: Unauthorized

## update replace status of the specified ReplicaSet

### HTTP Request

PUT /apis/apps/v1/namespaces/{namespace}/replicasets/{name}/status

### Parameters

- **name** (*in path*): string, required

name of the ReplicaSet

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [ReplicaSet](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([ReplicaSet](#)): OK

201 ([ReplicaSet](#)): Created

401: Unauthorized

## patch partially update the specified ReplicaSet

### HTTP Request

PATCH /apis/apps/v1/namespaces/{namespace}/replicasets/{name}

### Parameters

- **name** (*in path*): string, required  
name of the ReplicaSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([ReplicaSet](#)): OK

201 ([ReplicaSet](#)): Created

401: Unauthorized

**patch** partially update status of the specified ReplicaSet

## HTTP Request

PATCH /apis/apps/v1/namespaces/{namespace}/replicasets/{name}/status

## Parameters

- **name** (*in path*): string, required  
name of the ReplicaSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([ReplicaSet](#)): OK201 ([ReplicaSet](#)): Created

401: Unauthorized

## delete delete a ReplicaSet

### HTTP Request

DELETE /apis/apps/v1/namespaces/{namespace}/replicasets/{name}

### Parameters

- **name** (*in path*): string, required

name of the ReplicaSet

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

## Response

200 ([Status](#)): OK202 ([Status](#)): Accepted

401: Unauthorized

## deletecollection delete collection of ReplicaSet

### HTTP Request

DELETE /apis/apps/v1/namespaces/{namespace}/replicasets

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)
- **continue** (*in query*): string  
[continue](#)
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldSelector** (*in query*): string  
[fieldSelector](#)
- **gracePeriodSeconds** (*in query*): integer  
[gracePeriodSeconds](#)
- **labelSelector** (*in query*): string  
[labelSelector](#)
- **limit** (*in query*): integer  
[limit](#)
- **pretty** (*in query*): string  
[pretty](#)
- **propagationPolicy** (*in query*): string  
[propagationPolicy](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)
- **sendInitialEvents** (*in query*): boolean  
[sendInitialEvents](#)
- **timeoutSeconds** (*in query*): integer  
[timeoutSeconds](#)

## Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.5 - Deployment

Deployment enables declarative updates for Pods and ReplicaSets.

```
apiVersion: apps/v1

import "k8s.io/api/apps/v1"
```

## Deployment

Deployment enables declarative updates for Pods and ReplicaSets.

- **apiVersion**: apps/v1

- **kind**: Deployment

- **metadata** ([ObjectMeta](#))

Standard object's metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **spec** ([DeploymentSpec](#))

Specification of the desired behavior of the Deployment.

- **status** ([DeploymentStatus](#))

Most recently observed status of the Deployment.

## DeploymentSpec

DeploymentSpec is the specification of the desired behavior of the Deployment.

- **selector** ([LabelSelector](#)), required

Label selector for pods. Existing ReplicaSets whose pods are selected by this will be the ones affected by this deployment. It must match the pod template's labels.

- **template** ([PodTemplateSpec](#)), required

Template describes the pods that will be created. The only allowed template.spec.restartPolicy value is "Always".

- **replicas** (int32)

Number of desired pods. This is a pointer to distinguish between explicit zero and not specified. Defaults to 1.

- **minReadySeconds** (int32)

Minimum number of seconds for which a newly created pod should be ready without any of its container crashing, for it to be considered available. Defaults to 0 (pod will be considered available as soon as it is ready)

- **strategy** ([DeploymentStrategy](#))

*Patch strategy: retainKeys*

The deployment strategy to use to replace existing pods with new ones.

*DeploymentStrategy describes how to replace existing pods with new ones.*

- **strategy.type** (string)

Type of deployment. Can be "Recreate" or "RollingUpdate". Default is RollingUpdate.

- **strategy.rollingUpdate** (RollingUpdateDeployment)

Rolling update config params. Present only if DeploymentStrategyType = RollingUpdate.

*Spec to control the desired behavior of rolling update.*

- **strategy.rollingUpdate.maxSurge** (IntOrString)

The maximum number of pods that can be scheduled above the desired number of pods. Value can be an absolute number (ex: 5) or a percentage of desired pods (ex: 10%). This can not be 0 if MaxUnavailable is 0. Absolute number is calculated from percentage by rounding up. Defaults to 25%. Example: when this is set to 30%, the new ReplicaSet can be scaled up immediately when the rolling update starts, such that the total number of old and new pods do not exceed 130% of desired pods. Once old pods have been killed, new ReplicaSet can be scaled up further, ensuring that total number of pods running at any time during the update is at most 130% of desired pods.

*IntOrString is a type that can hold an int32 or a string. When used in JSON or YAML marshalling and unmarshalling, it produces or consumes the inner type. This allows you to have, for example, a JSON field that can accept a name or number.*

- **strategy.rollingUpdate.maxUnavailable** (IntOrString)

The maximum number of pods that can be unavailable during the update. Value can be an absolute number (ex: 5) or a percentage of desired pods (ex: 10%). Absolute number is calculated from percentage by rounding down. This can not be 0 if MaxSurge is 0. Defaults to 25%. Example: when this is set to 30%, the old ReplicaSet can be scaled down to 70% of desired pods immediately when the rolling update starts. Once new pods are ready, old ReplicaSet can be scaled down further, followed by scaling up the new ReplicaSet, ensuring that the total number of pods available at all times during the update is at least 70% of desired pods.

*IntOrString is a type that can hold an int32 or a string. When used in JSON or YAML marshalling and unmarshalling, it produces or consumes the inner type. This allows you to have, for example, a JSON field that can accept a name or number.*

- **revisionHistoryLimit** (int32)

The number of old ReplicaSets to retain to allow rollback. This is a pointer to distinguish between explicit zero and not specified. Defaults to 10.

- **progressDeadlineSeconds** (int32)

The maximum time in seconds for a deployment to make progress before it is considered to be failed. The deployment controller will continue to process failed deployments and a condition with a ProgressDeadlineExceeded reason will be surfaced in the deployment status. Note that progress will not be estimated during the time a deployment is paused. Defaults to 600s.

- **paused** (boolean)

Indicates that the deployment is paused.

## DeploymentStatus

DeploymentStatus is the most recently observed status of the Deployment.

- **replicas** (int32)

Total number of non-terminated pods targeted by this deployment (their labels match the selector).

- **availableReplicas** (int32)

Total number of available pods (ready for at least minReadySeconds) targeted by this deployment.

- **readyReplicas** (int32)

readyReplicas is the number of pods targeted by this Deployment with a Ready Condition.

- **unavailableReplicas** (int32)

Total number of unavailable pods targeted by this deployment. This is the total number of pods that are still required for the deployment to have 100% available capacity. They may either be pods that are running but not yet available or pods that still have not been created.

- **updatedReplicas** (int32)

Total number of non-terminated pods targeted by this deployment that have the desired template spec.

- **collisionCount** (int32)

Count of hash collisions for the Deployment. The Deployment controller uses this field as a collision avoidance mechanism when it needs to create the name for the newest ReplicaSet.

- **conditions** ([]DeploymentCondition)

*Patch strategy: merge on key type*

Represents the latest available observations of a deployment's current state.

*DeploymentCondition describes the state of a deployment at a certain point.*

- **conditions.status** (string), required

Status of the condition, one of True, False, Unknown.

- **conditions.type** (string), required

Type of deployment condition.

- **conditions.lastTransitionTime** (Time)

Last time the condition transitioned from one status to another.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **conditions.lastUpdateTime** (Time)

The last time this condition was updated.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **conditions.message** (string)

A human readable message indicating details about the transition.

- **conditions.reason** (string)

The reason for the condition's last transition.

- **observedGeneration** (int64)

The generation observed by the deployment controller.

## DeploymentList

DeploymentList is a list of Deployments.

- **apiVersion**: apps/v1

- **kind**: DeploymentList

- **metadata** ([ListMeta](#))

Standard list metadata.

- **items** ([\[\]Deployment](#)), required

Items is the list of Deployments.

## Operations

### **get** read the specified Deployment

#### HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/deployments/{name}

#### Parameters

- **name** (*in path*): string, required

name of the Deployment

- **namespace** (*in path*): string, required

[namespace](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([Deployment](#)): OK

401: Unauthorized

## get read status of the specified Deployment

### HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/deployments/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the Deployment
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([Deployment](#)): OK

401: Unauthorized

## list list or watch objects of kind Deployment

### HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/deployments

### Parameters

- **namespace** (*in path*): string, required  
[namespace](#)

- **allowWatchBookmarks** (*in query*): boolean  
[allowWatchBookmarks](#)

- **continue** (*in query*): string  
[continue](#)

- **fieldSelector** (*in query*): string  
[fieldSelector](#)

- **labelSelector** (*in query*): string  
[labelSelector](#)

- **limit** (*in query*): integer  
[limit](#)
- **pretty** (*in query*): string  
[pretty](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)
- **sendInitialEvents** (*in query*): boolean  
[sendInitialEvents](#)
- **timeoutSeconds** (*in query*): integer  
[timeoutSeconds](#)
- **watch** (*in query*): boolean  
[watch](#)

## Response

200 ([DeploymentList](#)): OK

401: Unauthorized

**list** list or watch objects of kind Deployment

## HTTP Request

GET /apis/apps/v1/deployments

## Parameters

- **allowWatchBookmarks** (*in query*): boolean  
[allowWatchBookmarks](#)
- **continue** (*in query*): string  
[continue](#)
- **fieldSelector** (*in query*): string  
[fieldSelector](#)
- **labelSelector** (*in query*): string  
[labelSelector](#)
- **limit** (*in query*): integer  
[limit](#)
- **pretty** (*in query*): string  
[pretty](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([DeploymentList](#)): OK

401: Unauthorized

## create create a Deployment

### HTTP Request

POST /apis/apps/v1/namespaces/{namespace}/deployments

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [Deployment](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([Deployment](#)): OK

201 ([Deployment](#)): Created

202 ([Deployment](#)): Accepted

401: Unauthorized

## update replace the specified Deployment

### HTTP Request

PUT /apis/apps/v1/namespaces/{namespace}/deployments/{name}

## Parameters

- **name** (*in path*): string, required  
name of the Deployment
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Deployment](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([Deployment](#)): OK

201 ([Deployment](#)): Created

401: Unauthorized

**update** replace status of the specified Deployment

## HTTP Request

PUT /apis/apps/v1/namespaces/{namespace}/deployments/{name}/status

## Parameters

- **name** (*in path*): string, required  
name of the Deployment
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Deployment](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

[pretty](#)

## Response

200 ([Deployment](#)): OK

201 ([Deployment](#)): Created

401: Unauthorized

**patch** partially update the specified Deployment

## HTTP Request

PATCH /apis/apps/v1/namespaces/{namespace}/deployments/{name}

## Parameters

- **name** (*in path*): string, required  
name of the Deployment
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([Deployment](#)): OK

201 ([Deployment](#)): Created

401: Unauthorized

**patch** partially update status of the specified Deployment

## HTTP Request

PATCH /apis/apps/v1/namespaces/{namespace}/deployments/{name}/status

## Parameters

- **name** (*in path*): string, required

name of the Deployment

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [Patch](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([Deployment](#)): OK

201 ([Deployment](#)): Created

401: Unauthorized

## delete delete a Deployment

### HTTP Request

DELETE /apis/apps/v1/namespaces/{namespace}/deployments/{name}

### Parameters

- **name** (*in path*): string, required

name of the Deployment

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

## Response

200 ([Status](#)): OK

202 ([Status](#)): Accepted

401: Unauthorized

## deletecollection delete collection of Deployment

### HTTP Request

DELETE /apis/apps/v1/namespaces/{namespace}/deployments

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **continue** (*in query*): string

[continue](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

## Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.6 - StatefulSet

StatefulSet represents a set of pods with consistent identities.

```
apiVersion: apps/v1

import "k8s.io/api/apps/v1"
```

## StatefulSet

StatefulSet represents a set of pods with consistent identities. Identities are defined as:

- Network: A single stable DNS and hostname.
- Storage: As many VolumeClaims as requested.

The StatefulSet guarantees that a given network identity will always map to the same storage identity.

- **apiVersion**: apps/v1

- **kind**: StatefulSet

- **metadata** ([ObjectMeta](#))

Standard object's metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **spec** ([StatefulSetSpec](#))

Spec defines the desired identities of pods in this set.

- **status** ([StatefulSetStatus](#))

Status is the current status of Pods in this StatefulSet. This data may be out of date by some window of time.

## StatefulSetSpec

A StatefulSetSpec is the specification of a StatefulSet.

- **serviceName** (string), required

serviceName is the name of the service that governs this StatefulSet. This service must exist before the StatefulSet, and is responsible for the network identity of the set. Pods get DNS/ hostnames that follow the pattern: pod-specific-string.serviceName.default.svc.cluster.local where "pod-specific-string" is managed by the StatefulSet controller.

- **selector** ([LabelSelector](#)), required

selector is a label query over pods that should match the replica count. It must match the pod template's labels. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors>

- **template** ([PodTemplateSpec](#)), required

template is the object that describes the pod that will be created if

insufficient replicas are detected. Each pod stamped out by the StatefulSet will fulfill this Template, but have a unique identity from the rest of the StatefulSet. Each pod will be named with the format <statefulsetname>-<podindex>. For example, a pod in a StatefulSet named "web" with index number "3" would be named "web-3". The only allowed template.spec.restartPolicy value is "Always".

- **replicas** (int32)

replicas is the desired number of replicas of the given Template. These are replicas in the sense that they are instantiations of the same Template, but individual replicas also have a consistent identity. If unspecified, defaults to 1.

- **updateStrategy** (StatefulSetUpdateStrategy)

updateStrategy indicates the StatefulSetUpdateStrategy that will be employed to update Pods in the StatefulSet when a revision is made to Template.

*StatefulSetUpdateStrategy indicates the strategy that the StatefulSet controller will use to perform updates. It includes any additional parameters necessary to perform the update for the indicated strategy.*

- **updateStrategy.type** (string)

Type indicates the type of the StatefulSetUpdateStrategy.  
Default is RollingUpdate.

- **updateStrategy.rollingUpdate**

(RollingUpdateStatefulSetStrategy)

RollingUpdate is used to communicate parameters when Type is RollingUpdateStatefulSetStrategyType.

*RollingUpdateStatefulSetStrategy is used to communicate parameter for RollingUpdateStatefulSetStrategyType.*

- **updateStrategy.rollingUpdate.maxUnavailable**

(IntOrString)

The maximum number of pods that can be unavailable during the update. Value can be an absolute number (ex: 5) or a percentage of desired pods (ex: 10%). Absolute number is calculated from percentage by rounding up. This can not be 0. Defaults to 1. This field is alpha-level and is only honored by servers that enable the MaxUnavailableStatefulSet feature. The field applies to all pods in the range 0 to Replicas-1. That means if there is any unavailable pod in the range 0 to Replicas-1, it will be counted towards MaxUnavailable.

*IntOrString is a type that can hold an int32 or a string. When used in JSON or YAML marshalling and unmarshalling, it produces or consumes the inner type. This allows you to have, for example, a JSON field that can accept a name or number.*

- **updateStrategy.rollingUpdate.partition** (int32)

Partition indicates the ordinal at which the StatefulSet should be partitioned for updates. During a rolling update, all pods from ordinal Replicas-1 to Partition are updated. All pods from ordinal Partition-1 to 0 remain untouched. This is helpful in being able to do a canary based deployment. The default value is 0.

- **podManagementPolicy** (string)

podManagementPolicy controls how pods are created during initial scale up, when replacing pods on nodes, or when scaling down. The default policy is `OrderedReady`, where pods are created in increasing order (pod-0, then pod-1, etc) and the controller will wait until each pod is ready before continuing. When scaling down, the pods are removed in the opposite order. The alternative policy is `Parallel` which will create pods in parallel to match the desired scale without waiting, and on scale down will delete all pods at once.

- **revisionHistoryLimit** (int32)

revisionHistoryLimit is the maximum number of revisions that will be maintained in the StatefulSet's revision history. The revision history consists of all revisions not represented by a currently applied StatefulSetSpec version. The default value is 10.

- **volumeClaimTemplates** ([]PersistentVolumeClaim)

volumeClaimTemplates is a list of claims that pods are allowed to reference. The StatefulSet controller is responsible for mapping network identities to claims in a way that maintains the identity of a pod. Every claim in this list must have at least one matching (by name) volumeMount in one container in the template. A claim in this list takes precedence over any volumes in the template, with the same name.

- **minReadySeconds** (int32)

Minimum number of seconds for which a newly created pod should be ready without any of its container crashing for it to be considered available. Defaults to 0 (pod will be considered available as soon as it is ready)

- **persistentVolumeClaimRetentionPolicy**

(StatefulSetPersistentVolumeClaimRetentionPolicy)

persistentVolumeClaimRetentionPolicy describes the lifecycle of persistent volume claims created from volumeClaimTemplates. By default, all persistent volume claims are created as needed and retained until manually deleted. This policy allows the lifecycle to be altered, for example by deleting persistent volume claims when their stateful set is deleted, or when their pod is scaled down. This requires the StatefulSetAutoDeletePVC feature gate to be enabled, which is alpha. +optional

*StatefulSetPersistentVolumeClaimRetentionPolicy describes the policy used for PVCs created from the StatefulSet VolumeClaimTemplates.*

- **persistentVolumeClaimRetentionPolicy.whenDeleted**  
(string)

WhenDeleted specifies what happens to PVCs created from StatefulSet VolumeClaimTemplates when the StatefulSet is deleted. The default policy of `Retain` causes PVCs to not be affected by StatefulSet deletion. The `Delete` policy causes those PVCs to be deleted.

- **persistentVolumeClaimRetentionPolicy.whenScaled**  
(string)

WhenScaled specifies what happens to PVCs created from StatefulSet VolumeClaimTemplates when the StatefulSet is scaled down. The default policy of `Retain` causes PVCs to not be affected by a scaledown. The `Delete` policy causes the

associated PVCs for any excess pods above the replica count to be deleted.

- **ordinals** (StatefulSetOrdinals)

ordinals controls the numbering of replica indices in a StatefulSet. The default ordinals behavior assigns a "0" index to the first replica and increments the index by one for each additional replica requested. Using the ordinals field requires the StatefulSetStartOrdinal feature gate to be enabled, which is beta.

*StatefulSetOrdinals describes the policy used for replica ordinal assignment in this StatefulSet.*

- **ordinals.start** (int32)

start is the number representing the first replica's index. It may be used to number replicas from an alternate index (eg: 1-indexed) over the default 0-indexed names, or to orchestrate progressive movement of replicas from one StatefulSet to another. If set, replica indices will be in the range: [.spec.ordinals.start, .spec.ordinals.start + .spec.replicas]. If unset, defaults to 0. Replica indices will be in the range: [0, .spec.replicas).

## StatefulSetStatus

StatefulSetStatus represents the current state of a StatefulSet.

- **replicas** (int32), required

replicas is the number of Pods created by the StatefulSet controller.

- **readyReplicas** (int32)

readyReplicas is the number of pods created for this StatefulSet with a Ready Condition.

- **currentReplicas** (int32)

currentReplicas is the number of Pods created by the StatefulSet controller from the StatefulSet version indicated by currentRevision.

- **updatedReplicas** (int32)

updatedReplicas is the number of Pods created by the StatefulSet controller from the StatefulSet version indicated by updateRevision.

- **availableReplicas** (int32)

Total number of available pods (ready for at least minReadySeconds) targeted by this statefulset.

- **collisionCount** (int32)

collisionCount is the count of hash collisions for the StatefulSet. The StatefulSet controller uses this field as a collision avoidance mechanism when it needs to create the name for the newest ControllerRevision.

- **conditions** ([]StatefulSetCondition)

*Patch strategy: merge on key type*

Represents the latest available observations of a statefulset's current state.

*StatefulSetCondition describes the state of a statefulset at a certain point.*

- **conditions.status** (string), required

Status of the condition, one of True, False, Unknown.

- **conditions.type** (string), required

Type of statefulset condition.

- **conditions.lastTransitionTime** (Time)

Last time the condition transitioned from one status to another.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **conditions.message** (string)

A human readable message indicating details about the transition.

- **conditions.reason** (string)

The reason for the condition's last transition.

- **currentRevision** (string)

currentRevision, if not empty, indicates the version of the StatefulSet used to generate Pods in the sequence [0,currentReplicas).

- **updateRevision** (string)

updateRevision, if not empty, indicates the version of the StatefulSet used to generate Pods in the sequence [replicas-updatedReplicas,replicas)

- **observedGeneration** (int64)

observedGeneration is the most recent generation observed for this StatefulSet. It corresponds to the StatefulSet's generation, which is updated on mutation by the API Server.

## StatefulSetList

StatefulSetList is a collection of StatefulSets.

- **apiVersion**: apps/v1

- **kind**: StatefulSetList

- **metadata** ([ListMeta](#))

Standard list's metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **items** ([][StatefulSet](#)), required

Items is the list of stateful sets.

## Operations

## get read the specified StatefulSet

### HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/statefulsets/{name}

### Parameters

- **name** (*in path*): string, required  
name of the StatefulSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([StatefulSet](#)): OK

401: Unauthorized

## get read status of the specified StatefulSet

### HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/statefulsets/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the StatefulSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([StatefulSet](#)): OK

401: Unauthorized

## list list or watch objects of kind StatefulSet

### HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/statefulsets

### Parameters

- **namespace** (*in path*): string, required  
[namespace](#)

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([StatefulSetList](#)): OK

401: Unauthorized

**list** list or watch objects of kind StatefulSet

## HTTP Request

GET /apis/apps/v1/statefulsets

## Parameters

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([StatefulSetList](#)): OK

401: Unauthorized

## create create a StatefulSet

### HTTP Request

POST /apis/apps/v1/namespaces/{namespace}/statefulsets

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [StatefulSet](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([StatefulSet](#)): OK

201 ([StatefulSet](#)): Created

202 ([StatefulSet](#)): Accepted

401: Unauthorized

## update replace the specified StatefulSet

### HTTP Request

PUT /apis/apps/v1/namespaces/{namespace}/statefulsets/{name}

### Parameters

- **name** (*in path*): string, required  
name of the StatefulSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [StatefulSet](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([StatefulSet](#)): OK

201 ([StatefulSet](#)): Created

401: Unauthorized

## update replace status of the specified StatefulSet

### HTTP Request

PUT /apis/apps/v1/namespaces/{namespace}/statefulsets/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the StatefulSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [StatefulSet](#), required

- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([StatefulSet](#)): OK

201 ([StatefulSet](#)): Created

401: Unauthorized

**patch** partially update the specified StatefulSet

## HTTP Request

PATCH /apis/apps/v1/namespaces/{namespace}/statefulsets/{name}

## Parameters

- **name** (*in path*): string, required  
name of the StatefulSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([StatefulSet](#)): OK

201 ([StatefulSet](#)): Created

401: Unauthorized

## patch partially update status of the specified StatefulSet

### HTTP Request

PATCH /apis/apps/v1/namespaces/{namespace}/statefulsets/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the StatefulSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([StatefulSet](#)): OK

201 ([StatefulSet](#)): Created

401: Unauthorized

## delete delete a StatefulSet

### HTTP Request

DELETE /apis/apps/v1/namespaces/{namespace}/statefulsets/{name}

### Parameters

- **name** (*in path*): string, required  
name of the StatefulSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [DeleteOptions](#)
- **dryRun** (*in query*): string

[dryRun](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

## Response

200 ([Status](#)): OK202 ([Status](#)): Accepted

401: Unauthorized

## **deletecollection** delete collection of StatefulSet

### HTTP Request

DELETE /apis/apps/v1/namespaces/{namespace}/statefulsets

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **continue** (*in query*): string

[continue](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

## Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.7 - ControllerRevision

ControllerRevision implements an immutable snapshot of state data.

```
apiVersion: apps/v1

import "k8s.io/api/apps/v1"
```

## ControllerRevision

ControllerRevision implements an immutable snapshot of state data. Clients are responsible for serializing and deserializing the objects that contain their internal state. Once a ControllerRevision has been successfully created, it can not be updated. The API Server will fail validation of all requests that attempt to mutate the Data field.

ControllerRevisions may, however, be deleted. Note that, due to its use by both the DaemonSet and StatefulSet controllers for update and rollback, this object is beta. However, it may be subject to name and representation changes in future releases, and clients should not depend on its stability. It is primarily for internal use by controllers.

- **apiVersion**: apps/v1

- **kind**: ControllerRevision

- **metadata** ([ObjectMeta](#))

Standard object's metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **revision** (int64), required

Revision indicates the revision of the state represented by Data.

- **data** (RawExtension)

Data is the serialized representation of the state.

\*RawExtension is used to hold extensions in external versions.

To use this, make a field which has RawExtension as its type in your external, versioned struct, and Object in your internal struct. You also need to register your various plugin types.

// Internal package:

```
type MyAPIObject struct { runtime.TypeMeta json:",inline"
    MyPlugin runtime.Object json:"myPlugin" }
```

```
type PluginA struct { AOption string json:"aOption" }
```

// External package:

```
type MyAPIObject struct { runtime.TypeMeta json:",inline"
    MyPlugin runtime.RawExtension json:"myPlugin" }
```

```
type PluginA struct { AOption string json:"aOption" }
```

// On the wire, the JSON will look something like this:

```
{ "kind":"MyAPIObject", "apiVersion":"v1", "myPlugin": {
    "kind":"PluginA", "aOption":"foo", }, }
```

So what happens? `Decode` first uses `json` or `yaml` to unmarshal the serialized data into your external `MyAPIObject`. That causes the raw JSON to be stored, but not unpacked. The next step is to copy (using `pkg/conversion`) into the internal struct. The runtime package's `DefaultScheme` has conversion functions installed which will unpack the JSON stored in `RawExtension`, turning it into the correct object type, and storing it in the `Object`. (TODO: In the case where the object is of an unknown type, a `runtime.Unknown` object will be created and stored.)\*

## ControllerRevisionList

`ControllerRevisionList` is a resource containing a list of `ControllerRevision` objects.

- **apiVersion**: `apps/v1`
- **kind**: `ControllerRevisionList`
- **metadata** ([ListMeta](#))  
More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>
- **items** ([\[\]ControllerRevision](#)), required

`Items` is the list of `ControllerRevisions`

## Operations

### **get** read the specified `ControllerRevision`

#### HTTP Request

GET `/apis/apps/v1/namespaces/{namespace}/controllerrevisions/{name}`

#### Parameters

- **name** (*in path*): string, required  
name of the `ControllerRevision`
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

#### Response

200 ([ControllerRevision](#)): OK

401: Unauthorized

### **list** list or watch objects of kind `ControllerRevision`

#### HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/controllerrevisions

## Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([ControllerRevisionList](#)): OK

401: Unauthorized

**list** list or **watch** objects of kind ControllerRevision

## HTTP Request

GET /apis/apps/v1/controllerrevisions

## Parameters

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string  
[continue](#)
- **fieldSelector** (*in query*): string  
[fieldSelector](#)
- **labelSelector** (*in query*): string  
[labelSelector](#)
- **limit** (*in query*): integer  
[limit](#)
- **pretty** (*in query*): string  
[pretty](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)
- **sendInitialEvents** (*in query*): boolean  
[sendInitialEvents](#)
- **timeoutSeconds** (*in query*): integer  
[timeoutSeconds](#)
- **watch** (*in query*): boolean  
[watch](#)

## Response

200 ([ControllerRevisionList](#)): OK

401: Unauthorized

## create create a ControllerRevision

### HTTP Request

POST /apis/apps/v1/namespaces/{namespace}/controllerrevisions

### Parameters

- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [ControllerRevision](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([ControllerRevision](#)): OK

201 ([ControllerRevision](#)): Created

202 ([ControllerRevision](#)): Accepted

401: Unauthorized

**update** replace the specified ControllerRevision

## HTTP Request

PUT /apis/apps/v1/namespaces/{namespace}/controllerrevisions/{name}

## Parameters

- **name** (*in path*): string, required

name of the ControllerRevision

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [ControllerRevision](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([ControllerRevision](#)): OK

201 ([ControllerRevision](#)): Created

401: Unauthorized

**patch** partially update the specified ControllerRevision

## HTTP Request

PATCH /apis/apps/v1/namespaces/{namespace}/controllerrevisions/{name}

## Parameters

- **name** (*in path*): string, required  
name of the ControllerRevision
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([ControllerRevision](#)): OK

201 ([ControllerRevision](#)): Created

401: Unauthorized

## delete delete a ControllerRevision

### HTTP Request

DELETE /apis/apps/v1/namespaces/{namespace}/controllerrevisions/{name}

### Parameters

- **name** (*in path*): string, required  
name of the ControllerRevision
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [DeleteOptions](#)
- **dryRun** (*in query*): string  
[dryRun](#)
- **gracePeriodSeconds** (*in query*): integer  
[gracePeriodSeconds](#)
- **pretty** (*in query*): string  
[pretty](#)
- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

## Response

200 ([Status](#)): OK

202 ([Status](#)): Accepted

401: Unauthorized

## **deletecollection** delete collection of ControllerRevision

### HTTP Request

DELETE /apis/apps/v1/namespaces/{namespace}/controllerrevisions

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **continue** (*in query*): string

[continue](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

## Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.8 - DaemonSet

DaemonSet represents the configuration of a daemon set.

```
apiVersion: apps/v1

import "k8s.io/api/apps/v1"
```

## DaemonSet

DaemonSet represents the configuration of a daemon set.

- **apiVersion**: apps/v1

- **kind**: DaemonSet

- **metadata** ([ObjectMeta](#))

Standard object's metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **spec** ([DaemonSetSpec](#))

The desired behavior of this daemon set. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

- **status** ([DaemonSetStatus](#))

The current status of this daemon set. This data may be out of date by some window of time. Populated by the system. Read-only. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

## DaemonSetSpec

DaemonSetSpec is the specification of a daemon set.

- **selector** ([LabelSelector](#)), required

A label query over pods that are managed by the daemon set. Must match in order to be controlled. It must match the pod template's labels. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors>

- **template** ([PodTemplateSpec](#)), required

An object that describes the pod that will be created. The DaemonSet will create exactly one copy of this pod on every node that matches the template's node selector (or on every node if no node selector is specified). The only allowed template.spec.restartPolicy value is "Always". More info: <https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller#pod-template>

- **minReadySeconds** (int32)

The minimum number of seconds for which a newly created DaemonSet pod should be ready without any of its container crashing, for it to be considered available. Defaults to 0 (pod will be considered available as soon as it is ready).

- **updateStrategy** (DaemonSetUpdateStrategy)

An update strategy to replace existing DaemonSet pods with new pods.

*DaemonSetUpdateStrategy is a struct used to control the update strategy for a DaemonSet.*

- **updateStrategy.type** (string)

Type of daemon set update. Can be "RollingUpdate" or "onDelete". Default is RollingUpdate.

- **updateStrategy.rollingUpdate** (RollingUpdateDaemonSet)

Rolling update config params. Present only if type = "RollingUpdate".

*Spec to control the desired behavior of daemon set rolling update.*

- **updateStrategy.rollingUpdate.maxSurge** (IntOrString)

The maximum number of nodes with an existing available DaemonSet pod that can have an updated DaemonSet pod during an update. Value can be an absolute number (ex: 5) or a percentage of desired pods (ex: 10%). This can not be 0 if MaxUnavailable is 0. Absolute number is calculated from percentage by rounding up to a minimum of 1. Default value is 0. Example: when this is set to 30%, at most 30% of the total number of nodes that should be running the daemon pod (i.e. status.desiredNumberScheduled) can have their a new pod created before the old pod is marked as deleted. The update starts by launching new pods on 30% of nodes. Once an updated pod is available (Ready for at least minReadySeconds) the old DaemonSet pod on that node is marked deleted. If the old pod becomes unavailable for any reason (Ready transitions to false, is evicted, or is drained) an updated pod is immediately created on that node without considering surge limits. Allowing surge implies the possibility that the resources consumed by the daemonset on any given node can double if the readiness check fails, and so resource intensive daemonsets should take into account that they may cause evictions during disruption.

*IntOrString is a type that can hold an int32 or a string. When used in JSON or YAML marshalling and unmarshalling, it produces or consumes the inner type. This allows you to have, for example, a JSON field that can accept a name or number.*

- **updateStrategy.rollingUpdate.maxUnavailable** (IntOrString)

The maximum number of DaemonSet pods that can be unavailable during the update. Value can be an absolute number (ex: 5) or a percentage of total number of DaemonSet pods at the start of the update (ex: 10%). Absolute number is calculated from percentage by rounding up. This cannot be 0 if MaxSurge is 0. Default value is 1. Example: when this is set to 30%, at most 30% of the total number of nodes that should be running the daemon pod (i.e. status.desiredNumberScheduled) can have their pods stopped for an update at any given time. The update starts by stopping at most 30% of those

DaemonSet pods and then brings up new DaemonSet pods in their place. Once the new pods are available, it then proceeds onto other DaemonSet pods, thus ensuring that at least 70% of original number of DaemonSet pods are available at all times during the update.

*IntOrString is a type that can hold an int32 or a string. When used in JSON or YAML marshalling and unmarshalling, it produces or consumes the inner type. This allows you to have, for example, a JSON field that can accept a name or number.*

- **revisionHistoryLimit** (int32)

The number of old history to retain to allow rollback. This is a pointer to distinguish between explicit zero and not specified. Defaults to 10.

## DaemonsetStatus

DaemonsetStatus represents the current status of a daemon set.

- **numberReady** (int32), required

numberReady is the number of nodes that should be running the daemon pod and have one or more of the daemon pod running with a Ready Condition.

- **numberAvailable** (int32)

The number of nodes that should be running the daemon pod and have one or more of the daemon pod running and available (ready for at least spec.minReadySeconds)

- **numberUnavailable** (int32)

The number of nodes that should be running the daemon pod and have none of the daemon pod running and available (ready for at least spec.minReadySeconds)

- **numberMisscheduled** (int32), required

The number of nodes that are running the daemon pod, but are not supposed to run the daemon pod. More info: <https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/>

- **desiredNumberScheduled** (int32), required

The total number of nodes that should be running the daemon pod (including nodes correctly running the daemon pod). More info: <https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/>

- **currentNumberScheduled** (int32), required

The number of nodes that are running at least 1 daemon pod and are supposed to run the daemon pod. More info: <https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/>

- **updatedNumberScheduled** (int32)

The total number of nodes that are running updated daemon pod

- **collisionCount** (int32)

Count of hash collisions for the DaemonSet. The DaemonSet controller uses this field as a collision avoidance mechanism when it needs to create the name for the newest ControllerRevision.

- **conditions** ([]DaemonSetCondition)

*Patch strategy: merge on key type*

Represents the latest available observations of a DaemonSet's current state.

*DaemonSetCondition describes the state of a DaemonSet at a certain point.*

- **conditions.status** (string), required

Status of the condition, one of True, False, Unknown.

- **conditions.type** (string), required

Type of DaemonSet condition.

- **conditions.lastTransitionTime** (Time)

Last time the condition transitioned from one status to another.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **conditions.message** (string)

A human readable message indicating details about the transition.

- **conditions.reason** (string)

The reason for the condition's last transition.

- **observedGeneration** (int64)

The most recent generation observed by the daemon set controller.

## DaemonSetList

DaemonSetList is a collection of daemon sets.

- **apiVersion**: apps/v1

- **kind**: DaemonSetList

- **metadata** ([ListMeta](#))

Standard list metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **items** ([][DaemonSet](#)), required

A list of daemon sets.

## Operations

### **get** read the specified DaemonSet

## HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/daemonsets/{name}

### Parameters

- **name** (*in path*): string, required  
name of the DaemonSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([DaemonSet](#)): OK

401: Unauthorized

**get** read status of the specified DaemonSet

## HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/daemonsets/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the DaemonSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([DaemonSet](#)): OK

401: Unauthorized

**list** list or watch objects of kind DaemonSet

## HTTP Request

GET /apis/apps/v1/namespaces/{namespace}/daemonsets

### Parameters

- **namespace** (*in path*): string, required  
[namespace](#)
- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([DaemonSetList](#)): OK

401: Unauthorized

**list** list or watch objects of kind DaemonSet

## HTTP Request

GET /apis/apps/v1/daemonsets

## Parameters

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([DaemonSetList](#)): OK

401: Unauthorized

**create** create a DaemonSet

## HTTP Request

POST /apis/apps/v1/namespaces/{namespace}/daemonsets

## Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DaemonSet](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([DaemonSet](#)): OK

201 ([DaemonSet](#)): Created

202 ([DaemonSet](#)): Accepted

401: Unauthorized

## update replace the specified DaemonSet

### HTTP Request

PUT /apis/apps/v1/namespaces/{namespace}/daemonsets/{name}

### Parameters

- **name** (*in path*): string, required

name of the DaemonSet

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DaemonSet](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

### Response

200 ([DaemonSet](#)): OK

201 ([DaemonSet](#)): Created

401: Unauthorized

## update replace status of the specified DaemonSet

### HTTP Request

PUT /apis/apps/v1/namespaces/{namespace}/daemonsets/{name}/status

### Parameters

- **name** (*in path*): string, required

name of the DaemonSet

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DaemonSet](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([DaemonSet](#)): OK

201 ([DaemonSet](#)): Created

401: Unauthorized

**patch** partially update the specified DaemonSet

## HTTP Request

PATCH /apis/apps/v1/namespaces/{namespace}/daemonsets/{name}

## Parameters

- **name** (*in path*): string, required

name of the DaemonSet

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [Patch](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([DaemonSet](#)): OK

201 ([DaemonSet](#)): Created

401: Unauthorized

## patch partially update status of the specified DaemonSet

### HTTP Request

PATCH /apis/apps/v1/namespaces/{namespace}/daemonsets/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the DaemonSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([DaemonSet](#)): OK

201 ([DaemonSet](#)): Created

401: Unauthorized

## delete delete a DaemonSet

### HTTP Request

DELETE /apis/apps/v1/namespaces/{namespace}/daemonsets/{name}

### Parameters

- **name** (*in path*): string, required  
name of the DaemonSet
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [DeleteOptions](#)
- **dryRun** (*in query*): string

[dryRun](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

## Response

200 ([Status](#)): OK

202 ([Status](#)): Accepted

401: Unauthorized

## **deletecollection** delete collection of DaemonSet

### HTTP Request

DELETE /apis/apps/v1/namespaces/{namespace}/daemonsets

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **continue** (*in query*): string

[continue](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

## Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.9 - Job

Job represents the configuration of a single job.

```
apiVersion: batch/v1

import "k8s.io/api/batch/v1"
```

## Job

Job represents the configuration of a single job.

- **apiVersion**: batch/v1

- **kind**: Job

- **metadata** ([ObjectMeta](#))

Standard object's metadata. More info: <https://git.k8s.io/community/contributors-devel/sig-architecture/api-conventions.md#metadata>

- **spec** ([JobSpec](#))

Specification of the desired behavior of a job. More info: <https://git.k8s.io/community/contributors-devel/sig-architecture/api-conventions.md#spec-and-status>

- **status** ([JobStatus](#))

Current status of a job. More info: <https://git.k8s.io/community/contributors-devel/sig-architecture/api-conventions.md#spec-and-status>

## JobSpec

JobSpec describes how the job execution will look like.

## Replicas

- **template** ([PodTemplateSpec](#)), required

Describes the pod that will be created when executing a job. The only allowed template.spec.restartPolicy values are "Never" or "OnFailure". More info: <https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/>

- **parallelism** (int32)

Specifies the maximum desired number of pods the job should run at any given time. The actual number of pods running in steady state will be less than this number when ((.spec.completions - .status.successful) < .spec.parallelism), i.e. when the work left to do is less than max parallelism. More info: <https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/>

## Lifecycle

- **completions** (int32)

Specifies the desired number of successfully finished pods the job

should be run with. Setting to null means that the success of any pod signals the success of all pods, and allows parallelism to have any positive value. Setting to 1 means that parallelism is limited to 1 and the success of that pod signals the success of the job. More info: <https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/>

- **completionMode** (string)

completionMode specifies how Pod completions are tracked. It can be `NonIndexed` (default) or `Indexed`.

`NonIndexed` means that the Job is considered complete when there have been `.spec.completions` successfully completed Pods. Each Pod completion is homologous to each other.

`Indexed` means that the Pods of a Job get an associated completion index from 0 to `(.spec.completions - 1)`, available in the annotation `batch.kubernetes.io/job-completion-index`. The Job is considered complete when there is one successfully completed Pod for each index. When value is `Indexed`, `.spec.completions` must be specified and `.spec.parallelism` must be less than or equal to  $10^5$ . In addition, The Pod name takes the form `$(job-name)-$(index)-$(random-string)`, the Pod hostname takes the form `$(job-name)-$(index)`.

More completion modes can be added in the future. If the Job controller observes a mode that it doesn't recognize, which is possible during upgrades due to version skew, the controller skips updates for the Job.

- **backoffLimit** (int32)

Specifies the number of retries before marking this job failed. Defaults to 6

- **activeDeadlineSeconds** (int64)

Specifies the duration in seconds relative to the `startTime` that the job may be continuously active before the system tries to terminate it; value must be positive integer. If a Job is suspended (at creation or through an update), this timer will effectively be stopped and reset when the Job is resumed again.

- **ttlSecondsAfterFinished** (int32)

`ttlSecondsAfterFinished` limits the lifetime of a Job that has finished execution (either Complete or Failed). If this field is set, `ttlSecondsAfterFinished` after the Job finishes, it is eligible to be automatically deleted. When the Job is being deleted, its lifecycle guarantees (e.g. finalizers) will be honored. If this field is unset, the Job won't be automatically deleted. If this field is set to zero, the Job becomes eligible to be deleted immediately after it finishes.

- **suspend** (boolean)

`suspend` specifies whether the Job controller should create Pods or not. If a Job is created with `suspend` set to true, no Pods are created by the Job controller. If a Job is suspended after creation (i.e. the flag goes from false to true), the Job controller will delete all active Pods associated with this Job. Users must design their workload to gracefully handle this. Suspending a Job will reset the `StartTime` field of the Job, effectively resetting the `ActiveDeadlineSeconds` timer too. Defaults to false.

## Selector

- **selector** ([LabelSelector](#))

A label query over pods that should match the pod count. Normally, the system sets this field for you. More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors>

- **manualSelector** (boolean)

manualSelector controls generation of pod labels and pod selectors. Leave `manualSelector` unset unless you are certain what you are doing. When false or unset, the system picks labels unique to this job and appends those labels to the pod template. When true, the user is responsible for picking unique labels and specifying the selector. Failure to pick a unique label may cause this and other jobs to not function correctly. However, You may see `manualSelector=true` in jobs that were created with the old `extensions/v1beta1` API. More info: <https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/#specifying-your-own-pod-selector>

## Beta level

- **podFailurePolicy** (PodFailurePolicy)

Specifies the policy of handling failed pods. In particular, it allows to specify the set of actions and conditions which need to be satisfied to take the associated action. If empty, the default behaviour applies - the counter of failed pods, represented by the job's `.status.failed` field, is incremented and it is checked against the `backoffLimit`. This field cannot be used in combination with `restartPolicy=OnFailure`.

This field is beta-level. It can be used when the `JobPodFailurePolicy` feature gate is enabled (enabled by default).

*PodFailurePolicy describes how failed pods influence the backoffLimit.*

- **podFailurePolicy.rules** (`[]PodFailurePolicyRule`), required

*Atomic: will be replaced during a merge*

A list of pod failure policy rules. The rules are evaluated in order. Once a rule matches a Pod failure, the remaining of the rules are ignored. When no rule matches the Pod failure, the default handling applies - the counter of pod failures is incremented and it is checked against the `backoffLimit`. At most 20 elements are allowed.

*PodFailurePolicyRule describes how a pod failure is handled when the requirements are met. One of `onExitCodes` and `onPodConditions`, but not both, can be used in each rule.*

- **podFailurePolicy.rules.action** (string), required

Specifies the action taken on a pod failure when the requirements are satisfied. Possible values are:

- FailJob: indicates that the pod's job is marked as Failed and all running pods are terminated.
- FailIndex: indicates that the pod's index is marked as Failed and will not be restarted. This value is alpha-level. It can be used when the `JobBackoffLimitPerIndex` feature gate is enabled

(disabled by default).

- Ignore: indicates that the counter towards the `.backoffLimit` is not incremented and a replacement pod is created.
- Count: indicates that the pod is handled in the default way - the counter towards the `.backoffLimit` is incremented. Additional values are considered to be added in the future. Clients should react to an unknown action by skipping the rule.

- **podFailurePolicy.rules.onPodConditions**

(`[]PodFailurePolicyOnPodConditionsPattern`), required

*Atomic: will be replaced during a merge*

Represents the requirement on the pod conditions. The requirement is represented as a list of pod condition patterns. The requirement is satisfied if at least one pattern matches an actual pod condition. At most 20 elements are allowed.

*PodFailurePolicyOnPodConditionsPattern describes a pattern for matching an actual pod condition type.*

- **podFailurePolicy.rules.onPodConditions.status**  
(string), required

Specifies the required Pod condition status. To match a pod condition it is required that the specified status equals the pod condition status. Defaults to True.

- **podFailurePolicy.rules.onPodConditions.type**  
(string), required

Specifies the required Pod condition type. To match a pod condition it is required that specified type equals the pod condition type.

- **podFailurePolicy.rules.onExitCodes**

(`PodFailurePolicyOnExitCodesRequirement`)

Represents the requirement on the container exit codes.

*PodFailurePolicyOnExitCodesRequirement describes the requirement for handling a failed pod based on its container exit codes. In particular, it lookups the `.state.terminated.exitCode` for each app container and init container status, represented by the `.status.containerStatuses` and `.status.initContainerStatuses` fields in the Pod status, respectively. Containers completed with success (exit code 0) are excluded from the requirement check.*

- **podFailurePolicy.rules.onExitCodes.operator**  
(string), required

Represents the relationship between the container exit code(s) and the specified values. Containers completed with success (exit code 0) are excluded from the requirement check. Possible values are:

- In: the requirement is satisfied if at least one container exit code (might be multiple if there are multiple containers not restricted by the 'containerName' field) is in the set of specified

values.

- NotIn: the requirement is satisfied if at least one container exit code (might be multiple if there are multiple containers not restricted by the 'containerName' field) is not in the set of specified values. Additional values are considered to be added in the future. Clients should react to an unknown operator by assuming the requirement is not satisfied.

- **podFailurePolicy.rules.onExitCodes.values**

([]int32), required

*Set: unique values will be kept during a merge*

Specifies the set of values. Each returned container exit code (might be multiple in case of multiple containers) is checked against this set of values with respect to the operator. The list of values must be ordered and must not contain duplicates. Value '0' cannot be used for the In operator. At least one element is required. At most 255 elements are allowed.

- **podFailurePolicy.rules.onExitCodes.containerName**

(string)

Restricts the check for exit codes to the container with the specified name. When null, the rule applies to all containers. When specified, it should match one the container or initContainer names in the pod template.

## Alpha level

- **backoffLimitPerIndex** (int32)

Specifies the limit for the number of retries within an index before marking this index as failed. When enabled the number of failures per index is kept in the pod's batch.kubernetes.io/job-index-failure-count annotation. It can only be set when Job's completionMode=Indexed, and the Pod's restart policy is Never. The field is immutable. This field is alpha-level. It can be used when the `JobBackoffLimitPerIndex` feature gate is enabled (disabled by default).

- **maxFailedIndexes** (int32)

Specifies the maximal number of failed indexes before marking the Job as failed, when backoffLimitPerIndex is set. Once the number of failed indexes exceeds this number the entire Job is marked as Failed and its execution is terminated. When left as null the job continues execution of all of its indexes and is marked with the `Complete` Job condition. It can only be specified when backoffLimitPerIndex is set. It can be null or up to completions. It is required and must be less than or equal to  $10^4$  when completions greater than  $10^5$ . This field is alpha-level. It can be used when the `JobBackoffLimitPerIndex` feature gate is enabled (disabled by default).

- **podReplacementPolicy** (string)

podReplacementPolicy specifies when to create replacement Pods. Possible values are: - TerminatingOrFailed means that we recreate pods when they are terminating (has a

metadata.deletionTimestamp) or failed.

- Failed means to wait until a previously created Pod is fully terminated (has phase Failed or Succeeded) before creating a replacement Pod.

When using podFailurePolicy, Failed is the the only allowed value. TerminatingOrFailed and Failed are allowed values when podFailurePolicy is not in use. This is an alpha field. Enable JobPodReplacementPolicy to be able to use this field.

## JobStatus

JobStatus represents the current state of a Job.

- **startTime** (Time)

Represents time when the job controller started processing a job. When a Job is created in the suspended state, this field is not set until the first time it is resumed. This field is reset every time a Job is resumed from suspension. It is represented in RFC3339 form and is in UTC.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **completionTime** (Time)

Represents time when the job was completed. It is not guaranteed to be set in happens-before order across separate operations. It is represented in RFC3339 form and is in UTC. The completion time is only set when the job finishes successfully.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **active** (int32)

The number of pending and running pods.

- **failed** (int32)

The number of pods which reached phase Failed.

- **succeeded** (int32)

The number of pods which reached phase Succeeded.

- **completedIndexes** (string)

completedIndexes holds the completed indexes when .spec.completionMode = "Indexed" in a text format. The indexes are represented as decimal integers separated by commas. The numbers are listed in increasing order. Three or more consecutive numbers are compressed and represented by the first and last element of the series, separated by a hyphen. For example, if the completed indexes are 1, 3, 4, 5 and 7, they are represented as "1,3-5,7".

- **conditions** ([]JobCondition)

*Patch strategy: merge on key type*

*Atomic: will be replaced during a merge*

The latest available observations of an object's current state. When a Job fails, one of the conditions will have type "Failed" and status true. When a Job is suspended, one of the conditions will have type "Suspended" and status true; when the Job is resumed, the status of this condition will become false. When a Job is completed, one of the conditions will have type "Complete" and status true. More info: <https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/>

*JobCondition describes current state of a job.*

- **conditions.status** (string), required

Status of the condition, one of True, False, Unknown.

- **conditions.type** (string), required

Type of job condition, Complete or Failed.

- **conditions.lastProbeTime** (Time)

Last time the condition was checked.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **conditions.lastTransitionTime** (Time)

Last time the condition transit from one status to another.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **conditions.message** (string)

Human readable message indicating details about last transition.

- **conditions.reason** (string)

(brief) reason for the condition's last transition.

- **uncountedTerminatedPods** (UncountedTerminatedPods)

uncountedTerminatedPods holds the UIDs of Pods that have terminated but the job controller hasn't yet accounted for in the status counters.

The job controller creates pods with a finalizer. When a pod terminates (succeeded or failed), the controller does three steps to account for it in the job status:

1. Add the pod UID to the arrays in this field.
2. Remove the pod finalizer.
3. Remove the pod UID from the arrays while increasing the corresponding counter.

Old jobs might not be tracked using this field, in which case the field remains null.

*UncountedTerminatedPods holds UIDs of Pods that have terminated but haven't been accounted in Job status counters.*

- **uncountedTerminatedPods.failed** ([]string)

*Set: unique values will be kept during a merge*

failed holds UIDs of failed Pods.

- **uncountedTerminatedPods.succeeded** ([]string)

*Set: unique values will be kept during a merge*

succeeded holds UIDs of succeeded Pods.

## Beta level

- **ready** (int32)

The number of pods which have a Ready condition.

This field is beta-level. The job controller populates the field when the feature gate JobReadyPods is enabled (enabled by default).

## Alpha level

- **failedIndexes** (string)

FailedIndexes holds the failed indexes when backoffLimitPerIndex=true. The indexes are represented in the text format analogous as for the completedIndexes field, ie. they are kept as decimal integers separated by commas. The numbers are listed in increasing order. Three or more consecutive numbers are compressed and represented by the first and last element of the series, separated by a hyphen. For example, if the failed indexes are 1, 3, 4, 5 and 7, they are represented as "1,3-5,7". This field is alpha-level. It can be used when the JobBackoffLimitPerIndex feature gate is enabled (disabled by default).

- **terminating** (int32)

The number of pods which are terminating (in phase Pending or Running and have a deletionTimestamp).

This field is alpha-level. The job controller populates the field when the feature gate JobPodReplacementPolicy is enabled (disabled by default).

## JobList

JobList is a collection of jobs.

- **apiVersion**: batch/v1

- **kind**: JobList

- **metadata** ([ListMeta](#))

Standard list metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **items** ([]Job), required

items is the list of Jobs.

## Operations

### get read the specified Job

#### HTTP Request

GET /apis/batch/v1/namespaces/{namespace}/jobs/{name}

## Parameters

- **name** (*in path*): string, required  
name of the Job
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([Job](#)): OK

401: Unauthorized

**get** read status of the specified Job

## HTTP Request

GET /apis/batch/v1/namespaces/{namespace}/jobs/{name}/status

## Parameters

- **name** (*in path*): string, required  
name of the Job
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([Job](#)): OK

401: Unauthorized

**list** list or watch objects of kind Job

## HTTP Request

GET /apis/batch/v1/namespaces/{namespace}/jobs

## Parameters

- **namespace** (*in path*): string, required  
[namespace](#)
- **allowWatchBookmarks** (*in query*): boolean  
[allowWatchBookmarks](#)
- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([JobList](#)): OK

401: Unauthorized

**list** list or watch objects of kind Job

## HTTP Request

GET /apis/batch/v1/jobs

## Parameters

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

## [limit](#)

- **pretty** (*in query*): string  
[pretty](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)
- **sendInitialEvents** (*in query*): boolean  
[sendInitialEvents](#)
- **timeoutSeconds** (*in query*): integer  
[timeoutSeconds](#)
- **watch** (*in query*): boolean  
[watch](#)

## Response

200 ([JobList](#)): OK

401: Unauthorized

## create create a Job

### HTTP Request

POST /apis/batch/v1/namespaces/{namespace}/jobs

## Parameters

- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Job](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([Job](#)): OK

201 ([Job](#)): Created

202 ([Job](#)): Accepted

401: Unauthorized

## update replace the specified Job

### HTTP Request

PUT /apis/batch/v1/namespaces/{namespace}/jobs/{name}

### Parameters

- **name** (*in path*): string, required  
name of the Job
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Job](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([Job](#)): OK

201 ([Job](#)): Created

401: Unauthorized

## update replace status of the specified Job

### HTTP Request

PUT /apis/batch/v1/namespaces/{namespace}/jobs/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the Job
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Job](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([Job](#)): OK201 ([Job](#)): Created

401: Unauthorized

**patch** partially update the specified Job

## HTTP Request

PATCH /apis/batch/v1/namespaces/{namespace}/jobs/{name}

## Parameters

- **name** (*in path*): string, required

name of the Job

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [Patch](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([Job](#)): OK201 ([Job](#)): Created

401: Unauthorized

**patch** partially update status of the specified Job

## HTTP Request

PATCH /apis/batch/v1/namespaces/{namespace}/jobs/{name}/status

## Parameters

- **name** (*in path*): string, required  
name of the Job
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([Job](#)): OK

201 ([Job](#)): Created

401: Unauthorized

## delete delete a Job

### HTTP Request

DELETE /apis/batch/v1/namespaces/{namespace}/jobs/{name}

## Parameters

- **name** (*in path*): string, required  
name of the Job
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [DeleteOptions](#)
- **dryRun** (*in query*): string  
[dryRun](#)
- **gracePeriodSeconds** (*in query*): integer  
[gracePeriodSeconds](#)
- **pretty** (*in query*): string  
[pretty](#)

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

## Response

200 ([Status](#)): OK

202 ([Status](#)): Accepted

401: Unauthorized

## **deletecollection** delete collection of Job

### HTTP Request

DELETE /apis/batch/v1/namespaces/{namespace}/jobs

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **continue** (*in query*): string

[continue](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

## Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.10 - CronJob

CronJob represents the configuration of a single cron job.

```
apiVersion: batch/v1

import "k8s.io/api/batch/v1"
```

## CronJob

CronJob represents the configuration of a single cron job.

- **apiVersion**: batch/v1

- **kind**: CronJob

- **metadata** ([ObjectMeta](#))

Standard object's metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **spec** ([CronJobSpec](#))

Specification of the desired behavior of a cron job, including the schedule. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

- **status** ([CronJobStatus](#))

Current status of a cron job. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

## CronJobSpec

CronJobSpec describes how the job execution will look like and when it will actually run.

- **jobTemplate** ([JobTemplateSpec](#)), required

Specifies the job that will be created when executing a CronJob.

*JobTemplateSpec describes the data a Job should have when created from a template*

- **jobTemplate.metadata** ([ObjectMeta](#))

Standard object's metadata of the jobs created from this template. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **jobTemplate.spec** ([JobSpec](#))

Specification of the desired behavior of the job. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>

- **schedule** (string), required

The schedule in Cron format, see <https://en.wikipedia.org/wiki/Cron>.

- **timeZone** (string)

The time zone name for the given schedule, see [https://en.wikipedia.org/wiki/List\\_of\\_tz\\_database\\_time\\_zones](https://en.wikipedia.org/wiki/List_of_tz_database_time_zones). If not specified, this will default to the time zone of the kube-controller-manager process. The set of valid time zone names and the time zone offset is loaded from the system-wide time zone database by the API server during CronJob validation and the controller manager during execution. If no system-wide time zone database can be found a bundled version of the database is used instead. If the time zone name becomes invalid during the lifetime of a CronJob or due to a change in host configuration, the controller will stop creating new new Jobs and will create a system event with the reason UnknownTimeZone. More information can be found in <https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/#time-zones>

- **concurrencyPolicy** (string)

Specifies how to treat concurrent executions of a Job. Valid values are:

- "Allow" (default): allows CronJobs to run concurrently; - "Forbid": forbids concurrent runs, skipping next run if previous run hasn't finished yet; - "Replace": cancels currently running job and replaces it with a new one

- **startingDeadlineSeconds** (int64)

Optional deadline in seconds for starting the job if it misses scheduled time for any reason. Missed jobs executions will be counted as failed ones.

- **suspend** (boolean)

This flag tells the controller to suspend subsequent executions, it does not apply to already started executions. Defaults to false.

- **successfulJobsHistoryLimit** (int32)

The number of successful finished jobs to retain. Value must be non-negative integer. Defaults to 3.

- **failedJobsHistoryLimit** (int32)

The number of failed finished jobs to retain. Value must be non-negative integer. Defaults to 1.

## CronJobStatus

CronJobStatus represents the current state of a cron job.

- **active** ([][ObjectReference](#))

*Atomic: will be replaced during a merge*

A list of pointers to currently running jobs.

- **lastScheduleTime** (Time)

Information when was the last time the job was successfully scheduled.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **lastSuccessfulTime** (Time)

Information when was the last time the job successfully completed.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

## CronJobList

CronJobList is a collection of cron jobs.

- **apiVersion**: batch/v1

- **kind**: CronJobList

- **metadata** ([ListMeta](#))

Standard list metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **items** ([]CronJob), required

items is the list of CronJobs.

## Operations

### get read the specified CronJob

#### HTTP Request

GET /apis/batch/v1/namespaces/{namespace}/cronjobs/{name}

#### Parameters

- **name** (*in path*): string, required

name of the CronJob

- **namespace** (*in path*): string, required

[namespace](#)

- **pretty** (*in query*): string

[pretty](#)

#### Response

200 ([CronJob](#)): OK

401: Unauthorized

### get read status of the specified CronJob

#### HTTP Request

GET /apis/batch/v1/namespaces/{namespace}/cronjobs/{name}/status

#### Parameters

- **name** (*in path*): string, required  
name of the CronJob
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([CronJob](#)): OK

401: Unauthorized

**list** list or watch objects of kind CronJob

## HTTP Request

GET /apis/batch/v1/namespaces/{namespace}/cronjobs

## Parameters

- **namespace** (*in path*): string, required  
[namespace](#)
- **allowWatchBookmarks** (*in query*): boolean  
[allowWatchBookmarks](#)
- **continue** (*in query*): string  
[continue](#)
- **fieldSelector** (*in query*): string  
[fieldSelector](#)
- **labelSelector** (*in query*): string  
[labelSelector](#)
- **limit** (*in query*): integer  
[limit](#)
- **pretty** (*in query*): string  
[pretty](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)
- **sendInitialEvents** (*in query*): boolean  
[sendInitialEvents](#)
- **timeoutSeconds** (*in query*): integer  
[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([CronJobList](#)): OK

401: Unauthorized

**list** list or watch objects of kind CronJob

## HTTP Request

GET /apis/batch/v1/cronjobs

## Parameters

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([CronJobList](#)): OK

401: Unauthorized

## create create a CronJob

### HTTP Request

POST /apis/batch/v1/namespaces/{namespace}/cronjobs

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [CronJob](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

### Response

200 ([CronJob](#)): OK

201 ([CronJob](#)): Created

202 ([CronJob](#)): Accepted

401: Unauthorized

## update replace the specified CronJob

### HTTP Request

PUT /apis/batch/v1/namespaces/{namespace}/cronjobs/{name}

### Parameters

- **name** (*in path*): string, required

name of the CronJob

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [CronJob](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([CronJob](#)): OK

201 ([CronJob](#)): Created

401: Unauthorized

## update replace status of the specified CronJob

### HTTP Request

PUT /apis/batch/v1/namespaces/{namespace}/cronjobs/{name}/status

### Parameters

- **name** (*in path*): string, required

name of the CronJob

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [CronJob](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([CronJob](#)): OK

201 ([CronJob](#)): Created

401: Unauthorized

## patch partially update the specified CronJob

### HTTP Request

PATCH /apis/batch/v1/namespaces/{namespace}/cronjobs/{name}

### Parameters

- **name** (*in path*): string, required

name of the CronJob

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [Patch](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([CronJob](#)): OK

201 ([CronJob](#)): Created

401: Unauthorized

**patch** partially update status of the specified CronJob

## HTTP Request

PATCH /apis/batch/v1/namespaces/{namespace}/cronjobs/{name}/status

## Parameters

- **name** (*in path*): string, required

name of the CronJob

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [Patch](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([CronJob](#)): OK

201 ([CronJob](#)): Created

401: Unauthorized

## delete delete a CronJob

### HTTP Request

DELETE /apis/batch/v1/namespaces/{namespace}/cronjobs/{name}

### Parameters

- **name** (*in path*): string, required

name of the CronJob

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

## Response

200 ([Status](#)): OK

202 ([Status](#)): Accepted

401: Unauthorized

## deletecollection delete collection of CronJob

### HTTP Request

DELETE /apis/batch/v1/namespaces/{namespace}/cronjobs

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)
- **continue** (*in query*): string  
[continue](#)
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldSelector** (*in query*): string  
[fieldSelector](#)
- **gracePeriodSeconds** (*in query*): integer  
[gracePeriodSeconds](#)
- **labelSelector** (*in query*): string  
[labelSelector](#)
- **limit** (*in query*): integer  
[limit](#)
- **pretty** (*in query*): string  
[pretty](#)
- **propagationPolicy** (*in query*): string  
[propagationPolicy](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)
- **sendInitialEvents** (*in query*): boolean  
[sendInitialEvents](#)
- **timeoutSeconds** (*in query*): integer  
[timeoutSeconds](#)

## Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.11 - HorizontalPodAutoscaler

configuration of a horizontal pod autoscaler.

```
apiVersion: autoscaling/v1

import "k8s.io/api/autoscaling/v1"
```

### HorizontalPodAutoscaler

configuration of a horizontal pod autoscaler.

- **apiVersion**: autoscaling/v1
- **kind**: HorizontalPodAutoscaler
- **metadata** ([ObjectMeta](#))

Standard object metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **spec** ([HorizontalPodAutoscalerSpec](#))

spec defines the behaviour of autoscaler. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>.

- **status** ([HorizontalPodAutoscalerStatus](#))

status is the current information about the autoscaler.

### HorizontalPodAutoscalerSpec

specification of a horizontal pod autoscaler.

- **maxReplicas** (int32), required

maxReplicas is the upper limit for the number of pods that can be set by the autoscaler; cannot be smaller than MinReplicas.

- **scaleTargetRef** (CrossVersionObjectReference), required

reference to scaled resource; horizontal pod autoscaler will learn the current resource consumption and will set the desired number of pods by using its Scale subresource.

*CrossVersionObjectReference contains enough information to let you identify the referred resource.*

- **scaleTargetRef.kind** (string), required

kind is the kind of the referent; More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **scaleTargetRef.name** (string), required

name is the name of the referent; More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **scaleTargetRef.apiVersion** (string)

apiVersion is the API version of the referent

- **minReplicas** (int32)

minReplicas is the lower limit for the number of replicas to which the autoscaler can scale down. It defaults to 1 pod. minReplicas is allowed to be 0 if the alpha feature gate HPAScaleToZero is enabled and at least one Object or External metric is configured. Scaling is active as long as at least one metric value is available.

- **targetCPUUtilizationPercentage** (int32)

targetCPUUtilizationPercentage is the target average CPU utilization (represented as a percentage of requested CPU) over all the pods; if not specified the default autoscaling policy will be used.

## HorizontalPodAutoscalerStatus

current status of a horizontal pod autoscaler

- **currentReplicas** (int32), required

currentReplicas is the current number of replicas of pods managed by this autoscaler.

- **desiredReplicas** (int32), required

desiredReplicas is the desired number of replicas of pods managed by this autoscaler.

- **currentCPUUtilizationPercentage** (int32)

currentCPUUtilizationPercentage is the current average CPU utilization over all pods, represented as a percentage of requested CPU, e.g. 70 means that an average pod is using now 70% of its requested CPU.

- **lastScaleTime** (Time)

lastScaleTime is the last time the HorizontalPodAutoscaler scaled the number of pods; used by the autoscaler to control how often the number of pods is changed.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **observedGeneration** (int64)

observedGeneration is the most recent generation observed by this autoscaler.

## HorizontalPodAutoscalerList

list of horizontal pod autoscaler objects.

- **apiVersion**: autoscaling/v1

- **kind**: HorizontalPodAutoscalerList

- **metadata** ([ListMeta](#))

Standard list metadata.

- **items** ([][HorizontalPodAutoscaler](#)), required

items is the list of horizontal pod autoscaler objects.

## Operations

---

### **get** read the specified HorizontalPodAutoscaler

#### HTTP Request

GET /apis/autoscaling/v1/namespaces/{namespace}/horizontalpodautoscalers/{name}

#### Parameters

- **name** (*in path*): string, required  
name of the HorizontalPodAutoscaler
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

#### Response

200 ([HorizontalPodAutoscaler](#)): OK

401: Unauthorized

### **get** read status of the specified HorizontalPodAutoscaler

#### HTTP Request

GET /apis/autoscaling/v1/namespaces/{namespace}/horizontalpodautoscalers/{name}/status

#### Parameters

- **name** (*in path*): string, required  
name of the HorizontalPodAutoscaler
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

#### Response

200 ([HorizontalPodAutoscaler](#)): OK

401: Unauthorized

### **list** list or watch objects of kind HorizontalPodAutoscaler

## HTTP Request

GET /apis/autoscaling/v1/namespaces/{namespace}/horizontalpodautoscalers

### Parameters

- **namespace** (*in path*): string, required  
[namespace](#)
- **allowWatchBookmarks** (*in query*): boolean  
[allowWatchBookmarks](#)
- **continue** (*in query*): string  
[continue](#)
- **fieldSelector** (*in query*): string  
[fieldSelector](#)
- **labelSelector** (*in query*): string  
[labelSelector](#)
- **limit** (*in query*): integer  
[limit](#)
- **pretty** (*in query*): string  
[pretty](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)
- **sendInitialEvents** (*in query*): boolean  
[sendInitialEvents](#)
- **timeoutSeconds** (*in query*): integer  
[timeoutSeconds](#)
- **watch** (*in query*): boolean  
[watch](#)

### Response

200 ([HorizontalPodAutoscalerList](#)): OK

401: Unauthorized

**list** list or watch objects of kind HorizontalPodAutoscaler

## HTTP Request

GET /apis/autoscaling/v1/horizontalpodautoscalers

### Parameters

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([HorizontalPodAutoscalerList](#)): OK

401: Unauthorized

**create** create a HorizontalPodAutoscaler

## HTTP Request

POST /apis/autoscaling/v1/namespaces/{namespace}/horizontalpodautoscalers

## Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [HorizontalPodAutoscaler](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([HorizontalPodAutoscaler](#)): OK

201 ([HorizontalPodAutoscaler](#)): Created

202 ([HorizontalPodAutoscaler](#)): Accepted

401: Unauthorized

**update** replace the specified HorizontalPodAutoscaler

## HTTP Request

PUT /apis/autoscaling/v1/namespaces/{namespace}/horizontalpodautoscalers/{name}

## Parameters

- **name** (*in path*): string, required

name of the HorizontalPodAutoscaler

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [HorizontalPodAutoscaler](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([HorizontalPodAutoscaler](#)): OK

201 ([HorizontalPodAutoscaler](#)): Created

401: Unauthorized

**update** replace status of the specified HorizontalPodAutoscaler

## HTTP Request

PUT /apis/autoscaling/v1/namespaces/{namespace}/horizontalpodautoscalers/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the HorizontalPodAutoscaler
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [HorizontalPodAutoscaler](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([HorizontalPodAutoscaler](#)): OK

201 ([HorizontalPodAutoscaler](#)): Created

401: Unauthorized

**patch** partially update the specified HorizontalPodAutoscaler

## HTTP Request

PATCH /apis/autoscaling/v1/namespaces/{namespace}/horizontalpodautoscalers/{name}

### Parameters

- **name** (*in path*): string, required  
name of the HorizontalPodAutoscaler
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([HorizontalPodAutoscaler](#)): OK

201 ([HorizontalPodAutoscaler](#)): Created

401: Unauthorized

**patch** partially update status of the specified HorizontalPodAutoscaler

## HTTP Request

PATCH /apis/autoscaling/v1/namespaces/{namespace}/horizontalpodautoscalers/{name}/status

## Parameters

- **name** (*in path*): string, required

name of the HorizontalPodAutoscaler

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [Patch](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([HorizontalPodAutoscaler](#)): OK

201 ([HorizontalPodAutoscaler](#)): Created

401: Unauthorized

## delete delete a HorizontalPodAutoscaler

### HTTP Request

DELETE /apis/autoscaling/v1/namespaces/{namespace}/horizontalpodautoscalers/{name}

### Parameters

- **name** (*in path*): string, required  
name of the HorizontalPodAutoscaler
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [DeleteOptions](#)
- **dryRun** (*in query*): string  
[dryRun](#)
- **gracePeriodSeconds** (*in query*): integer  
[gracePeriodSeconds](#)
- **pretty** (*in query*): string  
[pretty](#)
- **propagationPolicy** (*in query*): string  
[propagationPolicy](#)

### Response

200 ([Status](#)): OK

202 ([Status](#)): Accepted

401: Unauthorized

## deletecollection delete collection of HorizontalPodAutoscaler

### HTTP Request

DELETE /apis/autoscaling/v1/namespaces/{namespace}/horizontalpodautoscalers

### Parameters

- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [DeleteOptions](#)
- **continue** (*in query*): string  
[continue](#)
- **dryRun** (*in query*): string  
[dryRun](#)

- **fieldSelector** (*in query*): string  
[fieldSelector](#)
- **gracePeriodSeconds** (*in query*): integer  
[gracePeriodSeconds](#)
- **labelSelector** (*in query*): string  
[labelSelector](#)
- **limit** (*in query*): integer  
[limit](#)
- **pretty** (*in query*): string  
[pretty](#)
- **propagationPolicy** (*in query*): string  
[propagationPolicy](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)
- **sendInitialEvents** (*in query*): boolean  
[sendInitialEvents](#)
- **timeoutSeconds** (*in query*): integer  
[timeoutSeconds](#)

## Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.12 - HorizontalPodAutoscaler

HorizontalPodAutoscaler is the configuration for a horizontal pod autoscaler, which automatically manages the replica count of any resource implementing the scale subresource based on the metrics specified.

```
apiVersion: autoscaling/v2

import "k8s.io/api/autoscaling/v2"
```

## HorizontalPodAutoscaler

HorizontalPodAutoscaler is the configuration for a horizontal pod autoscaler, which automatically manages the replica count of any resource implementing the scale subresource based on the metrics specified.

- **apiVersion**: autoscaling/v2
- **kind**: HorizontalPodAutoscaler
- **metadata** ([ObjectMeta](#))

metadata is the standard object metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **spec** ([HorizontalPodAutoscalerSpec](#))

spec is the specification for the behaviour of the autoscaler. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>.

- **status** ([HorizontalPodAutoscalerStatus](#))

status is the current information about the autoscaler.

## HorizontalPodAutoscalerSpec

HorizontalPodAutoscalerSpec describes the desired functionality of the HorizontalPodAutoscaler.

- **maxReplicas** (int32), required

maxReplicas is the upper limit for the number of replicas to which the autoscaler can scale up. It cannot be less than minReplicas.

- **scaleTargetRef** (CrossVersionObjectReference), required

scaleTargetRef points to the target resource to scale, and is used to the pods for which metrics should be collected, as well as to actually change the replica count.

*CrossVersionObjectReference contains enough information to let you identify the referred resource.*

- **scaleTargetRef.kind** (string), required

kind is the kind of the referent; More info: <https://git.k8s.io/>

[community/contributors/devel/sig-architecture/api-conventions.md#types-kinds](https://kubernetes.io/docs/concepts/overview/working-with-objects/api-conventions.md#types-kinds)

- **scaleTargetRef.name** (string), required

name is the name of the referent; More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **scaleTargetRef.apiVersion** (string)

apiVersion is the API version of the referent

- **minReplicas** (int32)

minReplicas is the lower limit for the number of replicas to which the autoscaler can scale down. It defaults to 1 pod. minReplicas is allowed to be 0 if the alpha feature gate HPAScaleToZero is enabled and at least one Object or External metric is configured. Scaling is active as long as at least one metric value is available.

- **behavior** (HorizontalPodAutoscalerBehavior)

behavior configures the scaling behavior of the target in both Up and Down directions (scaleUp and scaleDown fields respectively). If not set, the default HPAScalingRules for scale up and scale down are used.

*HorizontalPodAutoscalerBehavior configures the scaling behavior of the target in both Up and Down directions (scaleUp and scaleDown fields respectively).*

- **behavior.scaleDown** (HPAScalingRules)

scaleDown is scaling policy for scaling Down. If not set, the default value is to allow to scale down to minReplicas pods, with a 300 second stabilization window (i.e., the highest recommendation for the last 300sec is used).

*HPAScalingRules configures the scaling behavior for one direction. These Rules are applied after calculating DesiredReplicas from metrics for the HPA. They can limit the scaling velocity by specifying scaling policies. They can prevent flapping by specifying the stabilization window, so that the number of replicas is not set instantly, instead, the safest value from the stabilization window is chosen.*

- **behavior.scaleDown.policies** ([]HPAScalingPolicy)

*Atomic: will be replaced during a merge*

policies is a list of potential scaling polices which can be used during scaling. At least one policy must be specified, otherwise the HPAScalingRules will be discarded as invalid

*HPAScalingPolicy is a single policy which must hold true for a specified past interval.*

- **behavior.scaleDown.policies.type** (string), required

type is used to specify the scaling policy.

- **behavior.scaleDown.policies.value** (int32), required

value contains the amount of change which is

permitted by the policy. It must be greater than zero

- **behavior.scaleDown.policies.periodSeconds**

(int32), required

periodSeconds specifies the window of time for which the policy should hold true. PeriodSeconds must be greater than zero and less than or equal to 1800 (30 min).

- **behavior.scaleDown.selectPolicy** (string)

selectPolicy is used to specify which policy should be used. If not set, the default value Max is used.

- **behavior.scaleDown.stabilizationWindowSeconds**

(int32)

stabilizationWindowSeconds is the number of seconds for which past recommendations should be considered while scaling up or scaling down.

StabilizationWindowSeconds must be greater than or equal to zero and less than or equal to 3600 (one hour). If not set, use the default values: - For scale up: 0 (i.e. no stabilization is done). - For scale down: 300 (i.e. the stabilization window is 300 seconds long).

- **behavior.scaleUp** (HPAScalingRules)

scaleUp is scaling policy for scaling Up. If not set, the default value is the higher of:

- increase no more than 4 pods per 60 seconds
- double the number of pods per 60 seconds No stabilization is used.

*HPAScalingRules configures the scaling behavior for one direction. These Rules are applied after calculating DesiredReplicas from metrics for the HPA. They can limit the scaling velocity by specifying scaling policies. They can prevent flapping by specifying the stabilization window, so that the number of replicas is not set instantly, instead, the safest value from the stabilization window is chosen.*

- **behavior.scaleUp.policies** ([]HPAScalingPolicy)

*Atomic: will be replaced during a merge*

policies is a list of potential scaling polices which can be used during scaling. At least one policy must be specified, otherwise the HPAScalingRules will be discarded as invalid

*HPAScalingPolicy is a single policy which must hold true for a specified past interval.*

- **behavior.scaleUp.policies.type** (string), required

type is used to specify the scaling policy.

- **behavior.scaleUp.policies.value** (int32), required

value contains the amount of change which is permitted by the policy. It must be greater than zero

- **behavior.scaleUp.policies.periodSeconds** (int32),

required

periodSeconds specifies the window of time for which the policy should hold true. PeriodSeconds must be greater than zero and less than or equal to 1800 (30 min).

- **behavior.scaleUp.selectPolicy** (string)

selectPolicy is used to specify which policy should be used. If not set, the default value Max is used.

- **behavior.scaleUp.stabilizationWindowSeconds** (int32)

stabilizationWindowSeconds is the number of seconds for which past recommendations should be considered while scaling up or scaling down.

StabilizationWindowSeconds must be greater than or equal to zero and less than or equal to 3600 (one hour). If not set, use the default values: - For scale up: 0 (i.e. no stabilization is done). - For scale down: 300 (i.e. the stabilization window is 300 seconds long).

- **metrics** ([]MetricSpec)

*Atomic: will be replaced during a merge*

metrics contains the specifications for which to use to calculate the desired replica count (the maximum replica count across all metrics will be used). The desired replica count is calculated multiplying the ratio between the target value and the current value by the current number of pods. Ergo, metrics used must decrease as the pod count is increased, and vice-versa. See the individual metric source types for more information about how each type of metric must respond. If not set, the default metric will be set to 80% average CPU utilization.

*MetricSpec specifies how to scale based on a single metric (only type and one other matching field should be set at once).*

- **metrics.type** (string), required

type is the type of metric source. It should be one of "ContainerResource", "External", "Object", "Pods" or "Resource", each mapping to a matching field in the object.

Note: "ContainerResource" type is available on when the feature-gate HPAContainerMetrics is enabled

- **metrics.containerResource**

(ContainerResourceMetricSource)

containerResource refers to a resource metric (such as those specified in requests and limits) known to Kubernetes describing a single container in each pod of the current scale target (e.g. CPU or memory). Such metrics are built in to Kubernetes, and have special scaling options on top of those available to normal per-pod metrics using the "pods" source. This is an alpha feature and can be enabled by the HPAContainerMetrics feature flag.

*ContainerResourceMetricSource indicates how to scale on a resource metric known to Kubernetes, as specified in requests and limits, describing each pod in the current scale target (e.g. CPU or memory). The values will be averaged together before being compared to the target. Such metrics are built in to Kubernetes, and have special scaling options on top of those available to*

normal per-pod metrics using the "pods" source. Only one "target" type should be set.

- **metrics.containerResource.container** (string), required

container is the name of the container in the pods of the scaling target

- **metrics.containerResource.name** (string), required

name is the name of the resource in question.

- **metrics.containerResource.target** (MetricTarget), required

target specifies the target value for the given metric

*MetricTarget defines the target value, average value, or average utilization of a specific metric*

- **metrics.containerResource.target.type** (string), required

type represents whether the metric type is Utilization, Value, or AverageValue

- **metrics.containerResource.target.averageUtilization** (int32)

averageUtilization is the target value of the average of the resource metric across all relevant pods, represented as a percentage of the requested value of the resource for the pods. Currently only valid for Resource metric source type

- **metrics.containerResource.target.averageValue** ([Quantity](#))

averageValue is the target value of the average of the metric across all relevant pods (as a quantity)

- **metrics.containerResource.target.value** ([Quantity](#))

value is the target value of the metric (as a quantity).

- **metrics.external** (ExternalMetricSource)

external refers to a global metric that is not associated with any Kubernetes object. It allows autoscaling based on information coming from components running outside of cluster (for example length of queue in cloud messaging service, or QPS from loadbalancer running outside of cluster).

*ExternalMetricSource indicates how to scale on a metric not associated with any Kubernetes object (for example length of queue in cloud messaging service, or QPS from loadbalancer running outside of cluster).*

- **metrics.external.metric** (MetricIdentifier), required

metric identifies the target metric by name and selector

*MetricIdentifier defines the name and optionally selector for a metric*

- **metrics.external.metric.name** (string), required

name is the name of the given metric

- **metrics.external.metric.selector** ([LabelSelector](#))

selector is the string-encoded form of a standard kubernetes label selector for the given metric. When set, it is passed as an additional parameter to the metrics server for more specific metrics scoping. When unset, just the metricName will be used to gather metrics.

- **metrics.external.target** ([MetricTarget](#)), required

target specifies the target value for the given metric

*MetricTarget defines the target value, average value, or average utilization of a specific metric*

- **metrics.external.target.type** (string), required

type represents whether the metric type is Utilization, Value, or AverageValue

- **metrics.external.target.averageUtilization**

(int32)

averageUtilization is the target value of the average of the resource metric across all relevant pods, represented as a percentage of the requested value of the resource for the pods. Currently only valid for Resource metric source type

- **metrics.external.target.averageValue** ([Quantity](#))

averageValue is the target value of the average of the metric across all relevant pods (as a quantity)

- **metrics.external.target.value** ([Quantity](#))

value is the target value of the metric (as a quantity).

- **metrics.object** ([ObjectMetricSource](#))

object refers to a metric describing a single kubernetes object (for example, hits-per-second on an Ingress object).

*ObjectMetricSource indicates how to scale on a metric describing a kubernetes object (for example, hits-per-second on an Ingress object).*

- **metrics.object.describedObject**

([CrossVersionObjectReference](#)), required

describedObject specifies the descriptions of a object, such as kind, name, apiVersion

*CrossVersionObjectReference contains enough information to let you identify the referred resource.*

- **metrics.object.describedObject.kind** (string),

required

kind is the kind of the referent; More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **metrics.object.describedObject.name** (string),

required

name is the name of the referent; More info:  
<https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **metrics.object.describedObject.apiVersion** (string)

apiVersion is the API version of the referent

- **metrics.object.metric** (MetricIdentifier), required

metric identifies the target metric by name and selector  
*MetricIdentifier defines the name and optionally selector for a metric*

- **metrics.object.metric.name** (string), required

name is the name of the given metric

- **metrics.object.metric.selector** ([LabelSelector](#))

selector is the string-encoded form of a standard kubernetes label selector for the given metric. When set, it is passed as an additional parameter to the metrics server for more specific metrics scoping. When unset, just the metricName will be used to gather metrics.

- **metrics.object.target** (MetricTarget), required

target specifies the target value for the given metric

*MetricTarget defines the target value, average value, or average utilization of a specific metric*

- **metrics.object.target.type** (string), required

type represents whether the metric type is Utilization, Value, or AverageValue

- **metrics.object.target.averageUtilization** (int32)

averageUtilization is the target value of the average of the resource metric across all relevant pods, represented as a percentage of the requested value of the resource for the pods. Currently only valid for Resource metric source type

- **metrics.object.target.averageValue** ([Quantity](#))

averageValue is the target value of the average of the metric across all relevant pods (as a quantity)

- **metrics.object.target.value** ([Quantity](#))

value is the target value of the metric (as a quantity).

- **metrics.pods** (PodsMetricSource)

pods refers to a metric describing each pod in the current scale target (for example, transactions-processed-per-second). The values will be averaged together before being compared to the target value.

*PodsMetricSource indicates how to scale on a metric describing each pod in the current scale target (for example, transactions-processed-per-second). The values will be averaged together*

before being compared to the target value.

- **metrics.pods.metric** (MetricIdentifier), required

metric identifies the target metric by name and selector

*MetricIdentifier defines the name and optionally selector for a metric*

- **metrics.pods.metric.name** (string), required

name is the name of the given metric

- **metrics.pods.metric.selector** ([LabelSelector](#))

selector is the string-encoded form of a standard kubernetes label selector for the given metric. When set, it is passed as an additional parameter to the metrics server for more specific metrics scoping. When unset, just the metricName will be used to gather metrics.

- **metrics.pods.target** (MetricTarget), required

target specifies the target value for the given metric

*MetricTarget defines the target value, average value, or average utilization of a specific metric*

- **metrics.pods.target.type** (string), required

type represents whether the metric type is Utilization, Value, or AverageValue

- **metrics.pods.target.averageUtilization** (int32)

averageUtilization is the target value of the average of the resource metric across all relevant pods, represented as a percentage of the requested value of the resource for the pods. Currently only valid for Resource metric source type

- **metrics.pods.target.averageValue** ([Quantity](#))

averageValue is the target value of the average of the metric across all relevant pods (as a quantity)

- **metrics.pods.target.value** ([Quantity](#))

value is the target value of the metric (as a quantity).

- **metrics.resource** (ResourceMetricSource)

resource refers to a resource metric (such as those specified in requests and limits) known to Kubernetes describing each pod in the current scale target (e.g. CPU or memory). Such metrics are built in to Kubernetes, and have special scaling options on top of those available to normal per-pod metrics using the "pods" source.

*ResourceMetricSource indicates how to scale on a resource metric known to Kubernetes, as specified in requests and limits, describing each pod in the current scale target (e.g. CPU or memory). The values will be averaged together before being compared to the target. Such metrics are built in to Kubernetes, and have special scaling options on top of those available to normal per-pod metrics using the "pods" source. Only one "target" type should be set.*

- **metrics.resource.name** (string), required
  - name is the name of the resource in question.
- **metrics.resource.target** (MetricTarget), required
  - target specifies the target value for the given metric

*MetricTarget defines the target value, average value, or average utilization of a specific metric*

  - **metrics.resource.target.type** (string), required
    - type represents whether the metric type is Utilization, Value, or AverageValue
  - **metrics.resource.target.averageUtilization** (int32)
    - averageUtilization is the target value of the average of the resource metric across all relevant pods, represented as a percentage of the requested value of the resource for the pods. Currently only valid for Resource metric source type
  - **metrics.resource.target.averageValue** ([Quantity](#))
    - averageValue is the target value of the average of the metric across all relevant pods (as a quantity)
  - **metrics.resource.target.value** ([Quantity](#))
    - value is the target value of the metric (as a quantity).

## HorizontalPodAutoscalerStatus

HorizontalPodAutoscalerStatus describes the current status of a horizontal pod autoscaler.

- 
- **desiredReplicas** (int32), required
    - desiredReplicas is the desired number of replicas of pods managed by this autoscaler, as last calculated by the autoscaler.
  - **conditions** ([]HorizontalPodAutoscalerCondition)
    - Patch strategy: merge on key type*
    - Map: unique values on key type will be kept during a merge*
    - conditions is the set of conditions required for this autoscaler to scale its target, and indicates whether or not those conditions are met.
    - HorizontalPodAutoscalerCondition describes the state of a HorizontalPodAutoscaler at a certain point.*
    - **conditions.status** (string), required
      - status is the status of the condition (True, False, Unknown)
    - **conditions.type** (string), required
      - type describes the current condition
    - **conditions.lastTransitionTime** (Time)

lastTransitionTime is the last time the condition transitioned from one status to another

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **conditions.message** (string)

message is a human-readable explanation containing details about the transition

- **conditions.reason** (string)

reason is the reason for the condition's last transition.

- **currentMetrics** ([]MetricStatus)

*Atomic: will be replaced during a merge*

currentMetrics is the last read state of the metrics used by this autoscaler.

*MetricStatus describes the last-read state of a single metric.*

- **currentMetrics.type** (string), required

type is the type of metric source. It will be one of "ContainerResource", "External", "Object", "Pods" or "Resource", each corresponds to a matching field in the object.

Note: "ContainerResource" type is available on when the feature-gate HPAContainerMetrics is enabled

- **currentMetrics.containerResource**

(ContainerResourceMetricStatus)

container resource refers to a resource metric (such as those specified in requests and limits) known to Kubernetes describing a single container in each pod in the current scale target (e.g. CPU or memory). Such metrics are built in to Kubernetes, and have special scaling options on top of those available to normal per-pod metrics using the "pods" source.

*ContainerResourceMetricStatus indicates the current value of a resource metric known to Kubernetes, as specified in requests and limits, describing a single container in each pod in the current scale target (e.g. CPU or memory). Such metrics are built in to Kubernetes, and have special scaling options on top of those available to normal per-pod metrics using the "pods" source.*

- **currentMetrics.containerResource.container** (string), required

container is the name of the container in the pods of the scaling target

- **currentMetrics.containerResource.current**

(MetricValueStatus), required

current contains the current value for the given metric

*MetricValueStatus holds the current value for a metric*

- **currentMetrics.containerResource.current.averageUtilization** (int32)

currentAverageUtilization is the current value of the average of the resource metric across all relevant pods, represented as a percentage of the requested

value of the resource for the pods.

- **currentMetrics.containerResource.current.averageValue** ([Quantity](#))

averageValue is the current value of the average of the metric across all relevant pods (as a quantity)

- **currentMetrics.containerResource.current.value** ([Quantity](#))

value is the current value of the metric (as a quantity).

- **currentMetrics.containerResource.name** (string), required

name is the name of the resource in question.

- **currentMetrics.external** (ExternalMetricStatus)

external refers to a global metric that is not associated with any Kubernetes object. It allows autoscaling based on information coming from components running outside of cluster (for example length of queue in cloud messaging service, or QPS from loadbalancer running outside of cluster).

*ExternalMetricStatus indicates the current value of a global metric not associated with any Kubernetes object.*

- **currentMetrics.external.current** (MetricValueStatus), required

current contains the current value for the given metric

*MetricValueStatus holds the current value for a metric*

- **currentMetrics.external.current.averageUtilization** (int32)

currentAverageUtilization is the current value of the average of the resource metric across all relevant pods, represented as a percentage of the requested value of the resource for the pods.

- **currentMetrics.external.current.averageValue** ([Quantity](#))

averageValue is the current value of the average of the metric across all relevant pods (as a quantity)

- **currentMetrics.external.current.value** ([Quantity](#))

value is the current value of the metric (as a quantity).

- **currentMetrics.external.metric** (MetricIdentifier), required

metric identifies the target metric by name and selector

*MetricIdentifier defines the name and optionally selector for a metric*

- **currentMetrics.external.metric.name** (string), required

name is the name of the given metric

- **currentMetrics.external.metric.selector**

[\(LabelSelector\)](#)

selector is the string-encoded form of a standard kubernetes label selector for the given metric. When set, it is passed as an additional parameter to the metrics server for more specific metrics scoping. When unset, just the metricName will be used to gather metrics.

- **currentMetrics.object** (ObjectMetricStatus)

object refers to a metric describing a single kubernetes object (for example, hits-per-second on an Ingress object).

*ObjectMetricStatus indicates the current value of a metric describing a kubernetes object (for example, hits-per-second on an Ingress object).*

- **currentMetrics.object.current** (MetricValueStatus), required

current contains the current value for the given metric

*MetricValueStatus holds the current value for a metric*

- **currentMetrics.object.current.averageUtilization** (int32)

currentAverageUtilization is the current value of the average of the resource metric across all relevant pods, represented as a percentage of the requested value of the resource for the pods.

- **currentMetrics.object.current.averageValue** ([Quantity](#))

averageValue is the current value of the average of the metric across all relevant pods (as a quantity)

- **currentMetrics.object.current.value** ([Quantity](#))

value is the current value of the metric (as a quantity).

- **currentMetrics.object.describedObject**

(CrossVersionObjectReference), required

DescribedObject specifies the descriptions of a object, such as kind, name, apiVersion

*CrossVersionObjectReference contains enough information to let you identify the referred resource.*

- **currentMetrics.object.describedObject.kind** (string), required

kind is the kind of the referent; More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds>

- **currentMetrics.object.describedObject.name** (string), required

name is the name of the referent; More info: <https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names>

- **currentMetrics.object.describedObject.apiVersion** (string)

apiVersion is the API version of the referent

- **currentMetrics.object.metric** (MetricIdentifier), required

metric identifies the target metric by name and selector

*MetricIdentifier defines the name and optionally selector for a metric*

- **currentMetrics.object.metric.name** (string), required

name is the name of the given metric

- **currentMetrics.object.metric.selector** ([LabelSelector](#))

selector is the string-encoded form of a standard kubernetes label selector for the given metric. When set, it is passed as an additional parameter to the metrics server for more specific metrics scoping. When unset, just the metricName will be used to gather metrics.

- **currentMetrics.pods** (PodsMetricStatus)

pods refers to a metric describing each pod in the current scale target (for example, transactions-processed-per-second). The values will be averaged together before being compared to the target value.

*PodsMetricStatus indicates the current value of a metric describing each pod in the current scale target (for example, transactions-processed-per-second).*

- **currentMetrics.pods.current** (MetricValueStatus), required

current contains the current value for the given metric

*MetricValueStatus holds the current value for a metric*

- **currentMetrics.pods.current.averageUtilization** (int32)

currentAverageUtilization is the current value of the average of the resource metric across all relevant pods, represented as a percentage of the requested value of the resource for the pods.

- **currentMetrics.pods.current.averageValue** ([Quantity](#))

averageValue is the current value of the average of the metric across all relevant pods (as a quantity)

- **currentMetrics.pods.current.value** ([Quantity](#))

value is the current value of the metric (as a quantity).

- **currentMetrics.pods.metric** (MetricIdentifier), required

metric identifies the target metric by name and selector

*MetricIdentifier defines the name and optionally selector for a metric*

- **currentMetrics.pods.metric.name** (string), required  
name is the name of the given metric
- **currentMetrics.pods.metric.selector** ([LabelSelector](#))  
selector is the string-encoded form of a standard kubernetes label selector for the given metric. When set, it is passed as an additional parameter to the metrics server for more specific metrics scoping. When unset, just the metricName will be used to gather metrics.

- **currentMetrics.resource** ([ResourceMetricStatus](#))  
resource refers to a resource metric (such as those specified in requests and limits) known to Kubernetes describing each pod in the current scale target (e.g. CPU or memory). Such metrics are built in to Kubernetes, and have special scaling options on top of those available to normal per-pod metrics using the "pods" source.

*ResourceMetricStatus indicates the current value of a resource metric known to Kubernetes, as specified in requests and limits, describing each pod in the current scale target (e.g. CPU or memory). Such metrics are built in to Kubernetes, and have special scaling options on top of those available to normal per-pod metrics using the "pods" source.*

- **currentMetrics.resource.current** ([MetricValueStatus](#)), required

current contains the current value for the given metric

*MetricValueStatus holds the current value for a metric*

- **currentMetrics.resource.current.averageUtilization** (int32)

currentAverageUtilization is the current value of the average of the resource metric across all relevant pods, represented as a percentage of the requested value of the resource for the pods.

- **currentMetrics.resource.current.averageValue** ([Quantity](#))

averageValue is the current value of the average of the metric across all relevant pods (as a quantity)

- **currentMetrics.resource.current.value** ([Quantity](#))

value is the current value of the metric (as a quantity).

- **currentMetrics.resource.name** (string), required

name is the name of the resource in question.

- **currentReplicas** (int32)

currentReplicas is current number of replicas of pods managed by this autoscaler, as last seen by the autoscaler.

- **lastScaleTime** (Time)

lastScaleTime is the last time the HorizontalPodAutoscaler scaled

the number of pods, used by the autoscaler to control how often the number of pods is changed.

*Time is a wrapper around time.Time which supports correct marshaling to YAML and JSON. Wrappers are provided for many of the factory methods that the time package offers.*

- **observedGeneration** (int64)

observedGeneration is the most recent generation observed by this autoscaler.

## HorizontalPodAutoscalerList

HorizontalPodAutoscalerList is a list of horizontal pod autoscaler objects.

- **apiVersion**: autoscaling/v2
- **kind**: HorizontalPodAutoscalerList
- **metadata** ([ListMeta](#))  
metadata is the standard list metadata.
- **items** ([\[\]HorizontalPodAutoscaler](#)), required  
items is the list of horizontal pod autoscaler objects.

## Operations

### **get** read the specified HorizontalPodAutoscaler

#### HTTP Request

```
GET /apis/autoscaling/v2/namespaces/{namespace}/  
horizontalpodautoscalers/{name}
```

#### Parameters

- **name** (*in path*): string, required  
name of the HorizontalPodAutoscaler
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

#### Response

200 ([HorizontalPodAutoscaler](#)): OK

401: Unauthorized

### **get** read status of the specified HorizontalPodAutoscaler

#### HTTP Request

GET /apis/autoscaling/v2/namespaces/{namespace}/horizontalpodautoscalers/{name}/status

## Parameters

- **name** (*in path*): string, required  
name of the HorizontalPodAutoscaler
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([HorizontalPodAutoscaler](#)): OK

401: Unauthorized

**list** list or watch objects of kind HorizontalPodAutoscaler

## HTTP Request

GET /apis/autoscaling/v2/namespaces/{namespace}/horizontalpodautoscalers

## Parameters

- **namespace** (*in path*): string, required  
[namespace](#)
- **allowWatchBookmarks** (*in query*): boolean  
[allowWatchBookmarks](#)
- **continue** (*in query*): string  
[continue](#)
- **fieldSelector** (*in query*): string  
[fieldSelector](#)
- **labelSelector** (*in query*): string  
[labelSelector](#)
- **limit** (*in query*): integer  
[limit](#)
- **pretty** (*in query*): string  
[pretty](#)
- **resourceVersion** (*in query*): string  
[resourceVersion](#)
- **resourceVersionMatch** (*in query*): string  
[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([HorizontalPodAutoscalerList](#)): OK

401: Unauthorized

**list** list or watch objects of kind HorizontalPodAutoscaler

## HTTP Request

GET /apis/autoscaling/v2/horizontalpodautoscalers

## Parameters

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([HorizontalPodAutoscalerList](#)): OK

401: Unauthorized

## create create a HorizontalPodAutoscaler

### HTTP Request

POST /apis/autoscaling/v2/namespaces/{namespace}/horizontalpodautoscalers

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [HorizontalPodAutoscaler](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([HorizontalPodAutoscaler](#)): OK

201 ([HorizontalPodAutoscaler](#)): Created

202 ([HorizontalPodAutoscaler](#)): Accepted

401: Unauthorized

## update replace the specified HorizontalPodAutoscaler

### HTTP Request

PUT /apis/autoscaling/v2/namespaces/{namespace}/horizontalpodautoscalers/{name}

### Parameters

- **name** (*in path*): string, required

name of the HorizontalPodAutoscaler

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [HorizontalPodAutoscaler](#), required

- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([HorizontalPodAutoscaler](#)): OK

201 ([HorizontalPodAutoscaler](#)): Created

401: Unauthorized

**update** replace status of the specified HorizontalPodAutoscaler

## HTTP Request

PUT /apis/autoscaling/v2/namespaces/{namespace}/horizontalpodautoscalers/{name}/status

## Parameters

- **name** (*in path*): string, required  
name of the HorizontalPodAutoscaler
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [HorizontalPodAutoscaler](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([HorizontalPodAutoscaler](#)): OK

201 ([HorizontalPodAutoscaler](#)): Created

401: Unauthorized

## patch partially update the specified HorizontalPodAutoscaler

### HTTP Request

PATCH /apis/autoscaling/v2/namespaces/{namespace}/horizontalpodautoscalers/{name}

### Parameters

- **name** (*in path*): string, required  
name of the HorizontalPodAutoscaler
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([HorizontalPodAutoscaler](#)): OK

201 ([HorizontalPodAutoscaler](#)): Created

401: Unauthorized

## patch partially update status of the specified HorizontalPodAutoscaler

### HTTP Request

PATCH /apis/autoscaling/v2/namespaces/{namespace}/horizontalpodautoscalers/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the HorizontalPodAutoscaler
- **namespace** (*in path*): string, required  
[namespace](#)

- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([HorizontalPodAutoscaler](#)): OK

201 ([HorizontalPodAutoscaler](#)): Created

401: Unauthorized

## delete delete a HorizontalPodAutoscaler

### HTTP Request

DELETE /apis/autoscaling/v2/namespaces/{namespace}/horizontalpodautoscalers/{name}

### Parameters

- **name** (*in path*): string, required  
name of the HorizontalPodAutoscaler
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [DeleteOptions](#)
- **dryRun** (*in query*): string  
[dryRun](#)
- **gracePeriodSeconds** (*in query*): integer  
[gracePeriodSeconds](#)
- **pretty** (*in query*): string  
[pretty](#)
- **propagationPolicy** (*in query*): string  
[propagationPolicy](#)

## Response

200 ([Status](#)): OK

202 ([Status](#)): Accepted

401: Unauthorized

## **deletecollection** delete collection of HorizontalPodAutoscaler

### HTTP Request

DELETE /apis/autoscaling/v2/namespaces/{namespace}/horizontalpodautoscalers

### Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **continue** (*in query*): string

[continue](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

### Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.13 - PriorityClass

PriorityClass defines mapping from a priority class name to the priority integer value.

```
apiVersion: scheduling.k8s.io/v1

import "k8s.io/api/scheduling/v1"
```

### PriorityClass

PriorityClass defines mapping from a priority class name to the priority integer value. The value can be any valid integer.

- **apiVersion**: scheduling.k8s.io/v1

- **kind**: PriorityClass

- **metadata** ([ObjectMeta](#))

Standard object's metadata. More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **value** (int32), required

value represents the integer value of this priority class. This is the actual priority that pods receive when they have the name of this class in their pod spec.

- **description** (string)

description is an arbitrary string that usually provides guidelines on when this priority class should be used.

- **globalDefault** (boolean)

globalDefault specifies whether this PriorityClass should be considered as the default priority for pods that do not have any priority class. Only one PriorityClass can be marked as `globalDefault`. However, if more than one PriorityClasses exists with their `globalDefault` field set to true, the smallest value of such global default PriorityClasses will be used as the default priority.

- **preemptionPolicy** (string)

preemptionPolicy is the Policy for preempting pods with lower priority. One of Never, PreemptLowerPriority. Defaults to PreemptLowerPriority if unset.

### PriorityClassList

PriorityClassList is a collection of priority classes.

- **apiVersion**: scheduling.k8s.io/v1

- **kind**: PriorityClassList

- **metadata** ([ListMeta](#))

Standard list metadata More info: <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

- **items** ([][PriorityClass](#)), required

items is the list of PriorityClasses

## Operations

---

### **get** read the specified PriorityClass

#### HTTP Request

GET /apis/scheduling.k8s.io/v1/priorityclasses/{name}

#### Parameters

- **name** (*in path*): string, required

name of the PriorityClass

- **pretty** (*in query*): string

[pretty](#)

#### Response

200 ([PriorityClass](#)): OK

401: Unauthorized

### **list** list or watch objects of kind PriorityClass

#### HTTP Request

GET /apis/scheduling.k8s.io/v1/priorityclasses

#### Parameters

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([PriorityClassList](#)): OK

401: Unauthorized

## create create a PriorityClass

### HTTP Request

POST /apis/scheduling.k8s.io/v1/priorityclasses

### Parameters

- **body**: [PriorityClass](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([PriorityClass](#)): OK

201 ([PriorityClass](#)): Created

202 ([PriorityClass](#)): Accepted

401: Unauthorized

## update replace the specified PriorityClass

### HTTP Request

PUT /apis/scheduling.k8s.io/v1/priorityclasses/{name}

### Parameters

- **name** (*in path*): string, required

name of the PriorityClass

- **body**: [PriorityClass](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([PriorityClass](#)): OK

201 ([PriorityClass](#)): Created

401: Unauthorized

**patch** partially update the specified PriorityClass

## HTTP Request

PATCH /apis/scheduling.k8s.io/v1/priorityclasses/{name}

## Parameters

- **name** (*in path*): string, required

name of the PriorityClass

- **body**: [Patch](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([PriorityClass](#)): OK

201 ([PriorityClass](#)): Created

401: Unauthorized

## delete delete a PriorityClass

### HTTP Request

DELETE /apis/scheduling.k8s.io/v1/priorityclasses/{name}

### Parameters

- **name** (*in path*): string, required  
name of the PriorityClass
- **body**: [DeleteOptions](#)
- **dryRun** (*in query*): string  
[dryRun](#)
- **gracePeriodSeconds** (*in query*): integer  
[gracePeriodSeconds](#)
- **pretty** (*in query*): string  
[pretty](#)
- **propagationPolicy** (*in query*): string  
[propagationPolicy](#)

### Response

200 ([Status](#)): OK

202 ([Status](#)): Accepted

401: Unauthorized

## deletecollection delete collection of PriorityClass

### HTTP Request

DELETE /apis/scheduling.k8s.io/v1/priorityclasses

### Parameters

- **body**: [DeleteOptions](#)
- **continue** (*in query*): string  
[continue](#)
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldSelector** (*in query*): string  
[fieldSelector](#)
- **gracePeriodSeconds** (*in query*): integer  
[gracePeriodSeconds](#)
- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

## Response

200 ([Status](#)): OK

401: Unauthorized

## 5.1.14 - PodSchedulingContext v1alpha2

PodSchedulingContext objects hold information that is needed to schedule a Pod with ResourceClaims that use "WaitForFirstConsumer" allocation mode.

```
apiVersion: resource.k8s.io/v1alpha2

import "k8s.io/api/resource/v1alpha2"
```

### PodSchedulingContext

PodSchedulingContext objects hold information that is needed to schedule a Pod with ResourceClaims that use "WaitForFirstConsumer" allocation mode.

This is an alpha type and requires enabling the DynamicResourceAllocation feature gate.

- **apiVersion**: resource.k8s.io/v1alpha2
- **kind**: PodSchedulingContext
- **metadata** ([ObjectMeta](#))

Standard object metadata

- **spec** ([PodSchedulingContextSpec](#)), required
- Spec describes where resources for the Pod are needed.
- **status** ([PodSchedulingContextStatus](#))

Status describes where resources for the Pod can be allocated.

### PodSchedulingContextSpec

PodSchedulingContextSpec describes where resources for the Pod are needed.

- **potentialNodes** ([]string)

*Set: unique values will be kept during a merge*

PotentialNodes lists nodes where the Pod might be able to run.

The size of this field is limited to 128. This is large enough for many clusters. Larger clusters may need more attempts to find a node that suits all pending resources. This may get increased in the future, but not reduced.

- **selectedNode** (string)

SelectedNode is the node for which allocation of ResourceClaims that are referenced by the Pod and that use "WaitForFirstConsumer" allocation is to be attempted.

### PodSchedulingContextStatus

PodSchedulingContextStatus describes where resources for the Pod can be allocated.

---

- **resourceClaims** ([]ResourceClaimSchedulingStatus)

*Map: unique values on key name will be kept during a merge*

ResourceClaims describes resource availability for each pod.spec.resourceClaim entry where the corresponding ResourceClaim uses "WaitForFirstConsumer" allocation mode.

*ResourceClaimSchedulingStatus contains information about one particular ResourceClaim with "WaitForFirstConsumer" allocation mode.*

- **resourceClaims.name** (string)

Name matches the pod.spec.resourceClaims[\*].Name field.

- **resourceClaims.unsuitableNodes** ([]string)

*Set: unique values will be kept during a merge*

UnsuitableNodes lists nodes that the ResourceClaim cannot be allocated for.

The size of this field is limited to 128, the same as for PodSchedulingSpec.PotentialNodes. This may get increased in the future, but not reduced.

## PodSchedulingContextList

PodSchedulingContextList is a collection of Pod scheduling objects.

---

- **apiVersion**: resource.k8s.io/v1alpha2

- **kind**: PodSchedulingContextList

- **metadata** ([ListMeta](#))

Standard list metadata

- **items** ([]PodSchedulingContext), required

Items is the list of PodSchedulingContext objects.

## Operations

---

### **get** read the specified PodSchedulingContext

#### HTTP Request

GET /apis/resource.k8s.io/v1alpha2/namespaces/{namespace}/podschedulingcontexts/{name}

#### Parameters

- **name** (*in path*): string, required

name of the PodSchedulingContext

- **namespace** (*in path*): string, required

[namespace](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([PodSchedulingContext](#)): OK

401: Unauthorized

**get** read status of the specified PodSchedulingContext

### HTTP Request

GET /apis/resource.k8s.io/v1alpha2/namespaces/{namespace}/podschedulingcontexts/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the PodSchedulingContext
- **namespace** (*in path*): string, required  
[namespace](#)
- **pretty** (*in query*): string  
[pretty](#)

## Response

200 ([PodSchedulingContext](#)): OK

401: Unauthorized

**list** list or watch objects of kind PodSchedulingContext

### HTTP Request

GET /apis/resource.k8s.io/v1alpha2/namespaces/{namespace}/podschedulingcontexts

### Parameters

- **namespace** (*in path*): string, required  
[namespace](#)

- **allowWatchBookmarks** (*in query*): boolean  
[allowWatchBookmarks](#)

- **continue** (*in query*): string  
[continue](#)

- **fieldSelector** (*in query*): string  
[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([PodSchedulingContextList](#)): OK

401: Unauthorized

**list** list or watch objects of kind PodSchedulingContext

## HTTP Request

GET /apis/resource.k8s.io/v1alpha2/podschedulingcontexts

## Parameters

- **allowWatchBookmarks** (*in query*): boolean

[allowWatchBookmarks](#)

- **continue** (*in query*): string

[continue](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

- **watch** (*in query*): boolean

[watch](#)

## Response

200 ([PodSchedulingContextList](#)): OK

401: Unauthorized

**create** create a PodSchedulingContext

## HTTP Request

POST /apis/resource.k8s.io/v1alpha2/namespaces/{namespace}/podschedulingcontexts

## Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [PodSchedulingContext](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([PodSchedulingContext](#)): OK201 ([PodSchedulingContext](#)): Created202 ([PodSchedulingContext](#)): Accepted

401: Unauthorized

**update** replace the specified PodSchedulingContext

## HTTP Request

PUT /apis/resource.k8s.io/v1alpha2/namespaces/{namespace}/podschedulingcontexts/{name}

### Parameters

- **name** (*in path*): string, required  
name of the PodSchedulingContext
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [PodSchedulingContext](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([PodSchedulingContext](#)): OK

201 ([PodSchedulingContext](#)): Created

401: Unauthorized

**update** replace status of the specified PodSchedulingContext

## HTTP Request

PUT /apis/resource.k8s.io/v1alpha2/namespaces/{namespace}/podschedulingcontexts/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the PodSchedulingContext
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [PodSchedulingContext](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([PodSchedulingContext](#)): OK

201 ([PodSchedulingContext](#)): Created

401: Unauthorized

**patch** partially update the specified PodSchedulingContext

## HTTP Request

PATCH /apis/resource.k8s.io/v1alpha2/namespaces/{namespace}/podschedulingcontexts/{name}

## Parameters

- **name** (*in path*): string, required

name of the PodSchedulingContext

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [Patch](#), required

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldManager** (*in query*): string

[fieldManager](#)

- **fieldValidation** (*in query*): string

[fieldValidation](#)

- **force** (*in query*): boolean

[force](#)

- **pretty** (*in query*): string

[pretty](#)

## Response

200 ([PodSchedulingContext](#)): OK

201 ([PodSchedulingContext](#)): Created

401: Unauthorized

**patch** partially update status of the specified PodSchedulingContext

## HTTP Request

PATCH /apis/resource.k8s.io/v1alpha2/namespaces/{namespace}/podschedulingcontexts/{name}/status

### Parameters

- **name** (*in path*): string, required  
name of the PodSchedulingContext
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [Patch](#), required
- **dryRun** (*in query*): string  
[dryRun](#)
- **fieldManager** (*in query*): string  
[fieldManager](#)
- **fieldValidation** (*in query*): string  
[fieldValidation](#)
- **force** (*in query*): boolean  
[force](#)
- **pretty** (*in query*): string  
[pretty](#)

### Response

200 ([PodSchedulingContext](#)): OK

201 ([PodSchedulingContext](#)): Created

401: Unauthorized

## delete delete a PodSchedulingContext

### HTTP Request

DELETE /apis/resource.k8s.io/v1alpha2/namespaces/{namespace}/podschedulingcontexts/{name}

### Parameters

- **name** (*in path*): string, required  
name of the PodSchedulingContext
- **namespace** (*in path*): string, required  
[namespace](#)
- **body**: [DeleteOptions](#)
- **dryRun** (*in query*): string  
[dryRun](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

## Response

200 ([PodSchedulingContext](#)): OK

202 ([PodSchedulingContext](#)): Accepted

401: Unauthorized

**deletecollection** delete collection of PodSchedulingContext

## HTTP Request

DELETE /apis/resource.k8s.io/v1alpha2/namespaces/{namespace}/podschedulingcontexts

## Parameters

- **namespace** (*in path*): string, required

[namespace](#)

- **body**: [DeleteOptions](#)

- **continue** (*in query*): string

[continue](#)

- **dryRun** (*in query*): string

[dryRun](#)

- **fieldSelector** (*in query*): string

[fieldSelector](#)

- **gracePeriodSeconds** (*in query*): integer

[gracePeriodSeconds](#)

- **labelSelector** (*in query*): string

[labelSelector](#)

- **limit** (*in query*): integer

[limit](#)

- **pretty** (*in query*): string

[pretty](#)

- **propagationPolicy** (*in query*): string

[propagationPolicy](#)

- **resourceVersion** (*in query*): string

[resourceVersion](#)

- **resourceVersionMatch** (*in query*): string

[resourceVersionMatch](#)

- **sendInitialEvents** (*in query*): boolean

[sendInitialEvents](#)

- **timeoutSeconds** (*in query*): integer

[timeoutSeconds](#)

## Response

200 ([Status](#)): OK

401: Unauthorized