Edit This Page

Reference

This section of the Kubernetes documentation contains references.

- API Reference
- API Client Libraries
- CLI Reference
- Config Reference
- Design Docs

API Reference

- Kubernetes API Overview - Overview of the API for Kubernetes.
- Kubernetes API Versions
    - 1.15
    - 1.14
    - 1.13
    - 1.12
    - 1.11

API 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

CLI Reference

- 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.
- kubefed - CLI tool to help you administrate your federated clusters.

Config Reference

- kubelet - The primary *node 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.
- federation-apiserver - API server for federated clusters.
- federation-controller-manager - Daemon that embeds the core control loops shipped with Kubernetes federation.

Design Docs

An archive of the design docs for Kubernetes functionality. Good starting points are Kubernetes Architecture and Kubernetes Design Overview.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 20, 2019 at 9:48 PM PST by generate and update api reference to 1.15 (#15042) (Page History)

Standardized Glossary

Analytics

Create an Issue
Page last modified on February 22, 2019 at 1:24 AM PST by Make k8s.io/docs/home to support i18n (#12570) (Page History)

Edit This Page

Kubernetes Issue Tracker

To report a security issue, please follow the Kubernetes security disclosure process.

Work on Kubernetes code and public issues are tracked using GitHub Issues.

- CVE-related issues

Security-related announcements are sent to the kubernetes-security-announce@googlegroups.com mailing list.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on March 28, 2019 at 2:45 PM PST by Update issues landing page (#13503) (Page History)

Edit This Page

Kubernetes Security and Disclosure Information

This page describes Kubernetes security and disclosure information.

- Security Announcements
- Report a Vulnerability
- Security Vulnerability Response
- Public Disclosure Timing

Security Announcements

Join the kubernetes-announce group for emails about security and major API announcements.

You can also subscribe to an RSS feed of the above using this link.

Report a Vulnerability

We're extremely grateful for security researchers and users that report vulnerabilities to the Kubernetes Open Source Community. All reports are thoroughly investigated by a set of community volunteers.

To make a report, please email the private security@kubernetes.io list with the security details and the details expected for all Kubernetes bug reports.

You may encrypt your email to this list using the GPG keys of the Product Security Committee members. Encryption using GPG is NOT required to make a disclosure.

When Should I Report a Vulnerability?

- You think you discovered a potential security vulnerability in Kubernetes
- You are unsure how a vulnerability affects Kubernetes
- You think you discovered a vulnerability in another project that Kubernetes depends on
  - For projects with their own vulnerability reporting and disclosure process, please report it directly there

When Should I NOT Report a Vulnerability?

- You need help tuning Kubernetes components for security
- You need help applying security related updates
- Your issue is not security related

Security Vulnerability Response

Each report is acknowledged and analyzed by Product Security Committee members within 3 working days. This will set off the Security Release Process.

Any vulnerability information shared with Product Security Committee stays within Kubernetes project and will not be disseminated to other projects unless it is necessary to get the issue fixed.

As the security issue moves from triage, to identified fix, to release planning we will keep the reporter updated.

Public Disclosure Timing

A public disclosure date is negotiated by the Kubernetes Product Security Committee and the bug submitter. We prefer to fully disclose the bug as soon as possible once a user mitigation is available. It is reasonable to delay disclosure when the bug or the fix is not yet fully understood, the solution is not well-tested, or for vendor coordination. The timeframe for disclosure is from immediate (especially if it's already publicly known) to a few weeks. For a vulnerability with a straightforward mitigation, we expect report date to disclosure date to be on the order of 7 days. The Kubernetes Product Security Committee holds the final say when setting a disclosure date.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Edit This Page

Kubernetes API Overview

This page provides an overview of the Kubernetes API.

- API versioning
- API groups
- Enabling API groups
- Enabling resources in the groups

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.

Most operations can be performed through the kubectl command-line interface or other command-line tools, such as kubeadm, which in turn use the API. However, you can also access the API directly using REST calls.

Consider using one of the client libraries if you are writing an application using the Kubernetes API.

API versioning

To eliminate fields or restructure resource representations, Kubernetes supports multiple API versions, each at a different API path. For example: `/api/v1` or `/apis/extensions/v1beta1`.

The version is set at the API level rather than at the resource or field level to:

- Ensure that the API presents a clear and consistent view of system resources and behavior.
- Enable control access to end-of-life and/or experimental APIs.

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

> **Note:** 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`).
  - The software may contain bugs. Enabling a feature may expose bugs. A feature may be disabled by default.
  - The support for a feature 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`).
  - The software is well tested. Enabling a feature is considered safe. Features are enabled by default.
  - 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 release. When this happens, migration instructions are provided. This may require deleting, editing, and re-creating API objects. The editing process may require some thought. This may require downtime for applications that rely on the feature.
  - The software is recommended for only non-business-critical uses because of potential for incompatible changes in subsequent releases. If you have multiple clusters which can be upgraded independently, you may be able to relax this restriction.

  Note: Try the 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.
  - The stable versions of features appear in released software for many subsequent versions.

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.

Currently, there are several API groups in use:

- The *core* (also called *legacy*) group, which is at REST path `/api/v1` and 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.

The two paths that support extending the API with custom resources are:

- CustomResourceDefinition for basic CRUD needs.
- aggregator for a full set of Kubernetes API semantics to implement their own apiserver.

Enabling API groups

Certain resources and API groups are enabled by default. You can enable or disable them by setting `--runtime-config` on the apiserver. `--runtime-config` accepts comma separated values. For example: - to disable batch/v1, set `--runtime-config=batch/v1=false` - to enable batch/v2alpha1, set `--runtime-config=batch/v2alpha1` The flag accepts comma separated set of key=value pairs describing runtime configuration of the apiserver.

> **Note:** When you enable or disable groups or resources, you need to restart the apiserver and controller-manager to pick up the `--runtime-config` changes.

Enabling resources in the groups

DaemonSets, Deployments, HorizontalPodAutoscalers, Ingress, Jobs and ReplicaSets are enabled by default. You can enable other extensions resources by setting `--runtime-config` on apiserver. `--runtime-config` accepts comma separated values. For example, to disable deployments and jobs, set `--runtime-config=extensions/v1beta1/deployments=false,extensions/v1beta1/jobs=false`

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on July 06, 2019 at 4:20 AM PST by Remove myself from review of all files except what-is-kubernetes and the deprecation policy. All of the lists of suggested reviewers need to be overhauled, but that's a separate task. (#15227) (Page History)

Edit This Page

Kubernetes API Concepts

This page describes common concepts in the Kubernetes API.

- Standard API terminology
- Efficient detection of changes
- Retrieving large results sets in chunks
- Receiving resources as Tables
- Alternate representations of resources
- Dry run
- Server Side Apply

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), includes additional subresources for many objects that allow fine grained authorization (such as binding a pod to a node), and can accept and serve those resources in different representations for convenience or efficiency. It also supports efficient change notifications on resources via "watches" and consistent lists to allow other components to effectively cache and synchronize the state of resources.

Standard API terminology

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" - they often represent operations rather than objects, such as a permission check (use a POST with a JSON-encoded body of `SubjectAccessReview` to the `subjectaccessreviews` resource). All objects will have a unique name to allow idempotent creation and retrieval, but virtual resource types may not have unique names if they are not retrievable or do not rely on idempotency.

Kubernetes generally leverages standard 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 in JSON (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 the resource type is called a **resource**

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. 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 of all namespaces with `GET /api/v1/namespaces` and details about a particular namespace with `GET /api/v1/namespaces/NAME`.

Almost all object resource types support the standard HTTP verbs - GET, POST, PUT, PATCH, and DELETE. 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**.

Some resource types will have one or more sub-resources, represented as sub paths below the resource:

- 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 documentation 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

To enable clients to build a model of the current state of a cluster, all Kubernetes object resource types are required to support consistent lists and an incremental change notification feed called a **watch**. Every Kubernetes object has a `resourceVersion` field representing the version of that resource as stored in the underlying database. When retrieving a collection of resources (either namespace or cluster scoped), the response from the server will contain a `resourceVersion` value that can be used to initiate a watch against the server. The server will return all changes (creates, deletes, and updates) that occur after the supplied `resourceVersion`. This allows a client to fetch the current state and then watch for changes without missing any updates. If the client watch is disconnected they can restart a new watch from the last returned `resourceVersion`, or perform a new collection request and begin again.

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 creates, deletes, or updates as individual JSON objects.

   ```
   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":
   {"resourceVersion": "10596", ...}, ...}
   }
   {
     "type": "MODIFIED",
     "object": {"kind": "Pod", "apiVersion": "v1", "metadata":
   {"resourceVersion": "11020", ...}, ...}
   ```

```
    }
    ...
```

A given Kubernetes server will only preserve a historical list of changes for a limited time. Clusters using etcd3 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 list operation, and starting the watch from the `resourceVersion` returned by that new list operation. Most client libraries offer some form of standard tool for this logic. (In Go this is called a `Reflector` and is located in the `k8s.io/client-go/cache` package.) To mitigate the impact of short history window, we introduced a concept of `bookmark` watch event. It is a special kind of event to pass an information that all changes up to a given `resourceVersion` client is requesting has already been send. Object returned in that event is of the type requested by the request, but only `resourceVersion` field is set, e.g.:

```
    GET /api/v1/namespaces/test/pods?
watch=1&resourceVersion=10245&allowWatchBookmarks=true
    ---
    200 OK
    Transfer-Encoding: chunked
    Content-Type: application/json
    {
      "type": "ADDED",
      "object": {"kind": "Pod", "apiVersion": "v1", "metadata":
{"resourceVersion": "10596", ...}, ...}
    }
    ...
    {
      "type": "BOOKMARK",
      "object": {"kind": "Pod", "apiVersion": "v1", "metadata":
{"resourceVersion": "12746"} }
    }
```

`Bookmark` events can be requested by `allowWatchBookmarks=true` option in watch requests, but clients shouldn't assume bookmarks are returned at any specific interval, nor may they assume the server will send any `bookmark` event. As of 1.15 release, it is an Alpha feature.

Retrieving large results sets in chunks

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 1-2kb 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. Starting in Kubernetes 1.9 the 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.

To retrieve a single list in chunks, two new parameters `limit` and `continue` are supported on collection requests and a new field `continue` is returned from all list operations in the list `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. The client can then pass this `continue` value to the server on the next request to instruct the server to return the next chunk of results. By continuing until the server returns an empty `continue` value the client can consume the full set of results.

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 the client wants to receive chunks of 500 pods at a time, they would 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",
       ...
     },
     "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",
       ...
     },
     "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 reached the end of the list
    ...
  },
  "items": [...] // returns pods 1001-1253
}
```

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

Receiving resources as Tables

`kubectl get` 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=v1beta1` with a `GET` call will request that the server return objects in the Table content type.

For example:

1. 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=v1beta1
   ---
   200 OK
   Content-Type: application/json
   {
       "kind": "Table",
       "apiVersion": "meta.k8s.io/v1beta1",
       ...
       "columnDefinitions": [
           ...
       ]
   }
   ```

For API resource types that do not have a custom Table definition on the server, a default Table response is returned by the server, consisting 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/v1beta1",
    ...
    "columnDefinitions": [
        {
            "name": "Name",
            "type": "string",
            ...
        },
        {
            "name": "Created At",
            "type": "date",
            ...
        }
    ]
}
```

Table responses are available beginning in version 1.10 of the kube-apiserver. As such, not all API resource types will support a Table response, specifically when using a client against older clusters. Clients that must work against all resource types, or can potentially deal with older clusters, should specify multiple content types in their `Accept` header to support fallback to non-Tabular JSON:

```
Accept: application/json;as=Table;g=meta.k8s.io;v=v1beta1,
application/json
```

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 return objects in the provided content type, while sending an object in Protobuf to the server for a `PUT` or `POST` call takes the `Content-Type` header. The server will return a `Content-Type` header if the requested format is supported, or the `406 Not acceptable` error if an invalid content type is provided.

See the API documentation 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 will support Protobuf, specifically those defined via Custom Resource Definitions or those that are API extensions. Clients that must work against all resource types should specify multiple content types in their `Accept` header to support fallback to JSON:

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

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" as set on the JSON object
    optional TypeMeta typeMeta = 1;

    // raw will hold the complete serialized object in protobuf.
See the protobuf definitions in the client libraries for a given
kind.
    optional bytes raw = 2;

    // contentEncoding is encoding used for the raw data.
Unspecified means no encoding.
    optional string contentEncoding = 3;

    // contentType is the serialization method used to serialize
'raw'. Unspecified means application/vnd.kubernetes.protobuf and
is usually
    // 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 exist for this object.
    optional string kind = 2;
  }
```

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.

Dry run

FEATURE STATE: `Kubernetes v1.13` beta
This feature is currently in a *beta* state, meaning:

- The version names contain beta (e.g. v2beta3).
- Code is well tested. Enabling the feature is considered safe. Enabled by default.
- Support for the overall feature will not be dropped, though details may change.
- The schema and/or semantics of objects may change in incompatible ways in a subsequent beta or stable release. When this happens, we will provide instructions for migrating to the next version. This may require deleting, editing, and re-creating API objects. The editing process may require some thought. This may require downtime for applications that rely on the feature.
- Recommended for only non-business-critical uses because of potential for incompatible changes in subsequent releases. If you have multiple clusters that can be upgraded independently, you may be able to relax this restriction.
- **Please do try our beta features and give feedback on them! After they exit beta, it may not be practical for us to make more changes.**

In version 1.13, the dry run beta feature is enabled by default. The modifying verbs (`POST`, `PUT`, `PATCH`, and `DELETE`) can accept requests 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. The system 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 in 1.13 the only accepted values are:

- `All`: Every stage runs as normal, except for the final storage stage. Admission controllers are run to check that the request is valid, mutating controllers mutate the request, 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 request would trigger an admission controller which would have 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 the sideEffects field to "None". If a webhook actually does have side effects, then the sideEffects field should be set to "NoneOnDryRun", and the webhook should also be modified to understand the `DryRun` field in AdmissionReview, and prevent side effects on dry run requests.
- Leave the value empty, which is also the default: Keep the default modifying behavior.

For example:

```
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 IPs that kube-apiserver assigns to v1.Service objects

Server Side Apply

FEATURE STATE: `Kubernetes v1.14` alpha
This feature is currently in a *alpha* state, meaning:

- The version names contain alpha (e.g. v1alpha1).
- Might be buggy. Enabling the feature may expose bugs. Disabled by default.
- Support for feature may be dropped at any time without notice.
- The API may change in incompatible ways in a later software release without notice.
- Recommended for use only in short-lived testing clusters, due to increased risk of bugs and lack of long-term support.

Server Side Apply allows clients other than kubectl to perform the Apply operation, and will eventually fully replace the complicated Client Side Apply logic that only exists in kubectl. If the Server Side Apply feature is enabled, the `PATCH` endpoint accepts the additional `application/apply-patch+yaml` content type. Users of Server Side Apply can send partially specified objects to this endpoint. An applied config should always include every field that the applier has an opinion about.

Enable the Server Side Apply alpha feature

Server Side Apply is an alpha feature, so it is disabled by default. To turn this feature gate on, you need to include the `--feature-gates ServerSideApply=true` flag when starting `kub` `e-apiserver`. If you have multiple `kube-apiserver` replicas, all should have the same flag setting.

Field Management

Compared to the `last-applied` annotation managed by `kubectl`, Server Side Apply uses a more declarative approach, which tracks a user'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.

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 either by changing the value with `POST`, `PUT`, or non-apply `PATCH`, or by including the field in a config sent to the Server Side Apply endpoint. Any applier that tries to change the field which is managed by someone else will get its request rejected (if not forced, see the Conflicts section below).

Field management is stored in a newly introduced `managedFields` field that is part of an object's [`metadata`].

A simple example of an object created by 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
    apiVersion: v1
    fields:
      f:metadata:
        f:labels:
          f:test-label: {}
      f:data:
        f:key: {}
data:
  key: some value
```

The above object contains a single manager in `metadata.managedFields`. The manager consists of basic information about the managing entity itself, like operation type, api version, and the fields managed by it.

> **Note:** This field is managed by the apiserver and should not be changed by the user.

Nevertheless it is possible to change `metadata.managedFields` through an `Update` operation. Doing so is highly discouraged, but might be a reasonable option to try if, for example, the `managedFields` get into an inconsistent state (which clearly should not happen).

Operations

The two operation types considered by this feature are `Apply` (`PATCH` with content type `application/apply-patch+yaml`) and `Update` (all other operations which modify the object). Both operations update the `managedFields`, but behave a little differently.

For instance, only the apply operation fails on conflicts while update does not. Also, apply operations are required to identify themselves by providing a `fieldManager` query parameter, while the query parameter is optional for update operations.

An example object with multiple managers could look like this:

```yaml
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 changed a value in the data field which caused the field's management to change to the `kube-controller-manager`.

> **Note:** If this update would have been an `Apply` operation, the operation 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 fact who manages them instead of overruling just based on values. This way it is intended to make it easier and more stable for multiple actors updating the same object by causing less unexpected interference.

When a user sends a partially specified object to the Server Side Apply endpoint, the server merges it with the live object favoring the value in the applied config 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 field manager is removed. For more information about how an object's schema is used to make decisions when merging, see sigs.k8s.io/structured-merge-diff.

Conflicts

A conflict is a special status error that occurs when an `Apply` operation tries to change a field, which another user 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 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 anymore, they can remove it from their config and make the request again. 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 the field, but doesn't want to overwrite it, they can change the value of the field in their config to match the value of the object on the server, and make the request again. This leaves the value unchanged, and causes the field's management to be shared by the applier and all other field managers that already claimed to manage it.

Comparison with Client Side Apply

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 config 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.

Custom Resources

Server Side Apply currently treats all custom resources as unstructured data. All keys are treated the same as struct fields, and all lists are considered atomic. In the future, it will use the validation field in Custom Resource Definitions to allow Custom Resource authors to define how to how to merge their own objects.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on July 06, 2019 at 4:20 AM PST by Remove myself from review of all files except what-is-kubernetes and the deprecation policy. All of the lists of suggested reviewers need to be overhauled, but that's a separate task. (#15227) (Page History)

Edit This Page

Client Libraries

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

- Officially-supported Kubernetes client libraries
- Community-maintained client libraries

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
Go	github.com/kubernetes/client-go/	browse
Python	github.com/kubernetes-client/python/	browse
Java	github.com/kubernetes-client/java	browse
dotnet	github.com/kubernetes-client/csharp	browse
JavaScript	github.com/kubernetes-client/javascript	browse

Community-maintained client libraries

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

Language	Client Library
Clojure	github.com/yanatan16/clj-kubernetes-api

Language	Client Library
Go	github.com/ericchiang/k8s
Java (OSGi)	bitbucket.org/amdatulabs/amdatu-kubernetes
Java (Fabric8, OSGi)	github.com/fabric8io/kubernetes-client
Lisp	github.com/brendandburns/cl-k8s
Lisp	github.com/xh4/cube
Node.js (TypeScript)	github.com/Goyoo/node-k8s-client
Node.js	github.com/tenxcloud/node-kubernetes-client
Node.js	github.com/godaddy/kubernetes-client
Node.js	github.com/ajpauwels/easy-k8s
Perl	metacpan.org/pod/Net::Kubernetes
PHP	github.com/maclof/kubernetes-client
PHP	github.com/allansun/kubernetes-php-client
PHP	github.com/travisghansen/kubernetes-client-php
Python	github.com/eldarion-gondor/pykube
Python	github.com/mnubo/kubernetes-py
Ruby	github.com/Ch00k/kuber
Ruby	github.com/abonas/kubeclient
Ruby	github.com/kontena/k8s-client
Rust	github.com/clux/kube-rs
Rust	github.com/ynqa/kubernetes-rust
Scala	github.com/doriordan/skuber
dotNet	github.com/tonnyeremin/kubernetes_gen
DotNet (RestSharp)	github.com/masroorhasan/Kubernetes.DotNet
Elixir	github.com/obmarg/kazan
Haskell	github.com/soundcloud/haskell-kubernetes

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 11, 2019 at 2:12 AM PST by add clux/kube-rs client (#14759) (Page History)

Edit This Page

Kubernetes Deprecation Policy

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

- Deprecating parts of the API
- Deprecating a flag or CLI
- Deprecating a feature or behavior
- Exceptions

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 until a new API version at least as stable is released.

GA API versions can replace GA API versions as well as beta and alpha API versions. Beta API versions *may not* replace GA API versions.

Rule #4a: Other than the most recent API versions in each track, older API versions must be supported after their announced deprecation for a duration of no less than:

- **GA: 12 months or 3 releases (whichever is longer)**
- **Beta: 9 months or 3 releases (whichever is longer)**
- **Alpha: 0 releases**

This covers the maximum supported version skew of 2 releases.

> **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 3 months (4 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	• v1alpha1 is removed, "action required" relnote

Release	API Versions	Preferred/ Storage Version	Notes
X+2	v1beta1	v1beta1	• v1alpha2 is removed, "action required" relnote
X+3	v1beta2, v1beta1 (deprecated)	v1beta1	• v1beta1 is deprecated, "action required" relnote
X+4	v1beta2, v1beta1 (deprecated)	v1beta2	
X+5	v1, v1beta1 (deprecated), v1beta2 (deprecated)	v1beta2	• v1beta2 is deprecated, "action required" relnote
X+6	v1, v1beta2 (deprecated)	v1	• v1beta1 is removed, "action required" relnote
X+7	v1, v1beta2 (deprecated)	v1	
X+8	v2alpha1, v1	v1	• v1beta2 is removed, "action required" relnote
X+9	v2alpha2, v1	v1	• v2alpha1 is removed, "action required" relnote
X+10	v2beta1, v1	v1	• v2alpha2 is removed, "action required" relnote
X+11	v2beta2, v2beta1 (deprecated), v1	v1	• v2beta1 is deprecated, "action required" relnote
X+12	v2, v2beta2 (deprecated), v2beta1 (deprecated), v1 (deprecated)	v1	• v2beta2 is deprecated, "action required" relnote • v1 is deprecated, "action required" relnote
X+13	v2, v2beta1 (deprecated), v2beta2 (deprecated), v1 (deprecated)	v2	
X+14	v2, v2beta2 (deprecated), v1 (deprecated)	v2	• v2beta1 is removed, "action required" relnote
X+15	v2, v1 (deprecated)	v2	• v2beta2 is removed, "action required" relnote
X+16	v2, v1 (deprecated)	v2	

Release	API Versions	Preferred/ Storage Version	Notes
X+17	v2	v2	• v1 is removed, "action required" relnote

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.

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 just like 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 #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.

This does not imply that all changes to the system are governed by this policy. This applies 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 #8: 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 #9: 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.

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.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

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 an enterprise.

- API Overview
- Default Roles and Role Bindings
- Privilege Escalation Prevention and Bootstrapping
- Command-line Utilities
- Service Account Permissions
- Upgrading from 1.5
- Permissive RBAC Permissions

`RBAC` uses the `rbac.authorization.k8s.io` API group to drive authorization decisions, allowing admins to dynamically configure policies through the Kubernetes API.

As of 1.8, RBAC mode is stable and backed by the rbac.authorization.k8s.io/v1 API.

To enable RBAC, start the apiserver with `--authorization-mode=RBAC`.

API Overview

The RBAC API declares four top-level types which will be covered in this section. Users can interact with these resources as they would with any other API resource (via `kubectl`, API calls, etc.). For instance, `kubectl apply -f (resource).yml` can be used with any of these examples, though readers who wish to follow along should review the section on bootstrapping first.

Role and ClusterRole

In the RBAC API, a role contains rules that represent a set of permissions. Permissions are purely additive (there are no "deny" rules). A role can be defined within a namespace with a `Role`, or cluster-wide with a `ClusterRole`.

A `Role` can only be used to grant access to resources within a single namespace. 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: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

A `ClusterRole` can be used to grant the same permissions as a `Role`, but because they are cluster-scoped, they can also be used to grant access to:

- cluster-scoped resources (like nodes)

- non-resource endpoints (like "/healthz")
- namespaced resources (like pods) across all namespaces (needed to run `kubectl get pods --all-namespaces`, for example)

The following `ClusterRole` 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: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

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. Permissions can be granted within a namespace with a `RoleBinding`, or cluster-wide with a `ClusterRoleBinding`.

A `RoleBinding` may reference a `Role` in the same namespace. The following `RoleBinding` grants the "pod-reader" role to the user "jane" within the "default" namespace. This allows "jane" to read pods in the "default" namespace.

`roleRef` is how you will actually create the binding. The `kind` will be either `Role` or `ClusterRole`, and the `name` will reference the name of the specific `Role` or `ClusterRole` you want. In the example below, this RoleBinding is using `roleRef` to bind the user "jane" to the `Role` created above named `pod-reader`.

```
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "jane" to read pods in the "default"
namespace.
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role #this must be Role or ClusterRole
  name: pod-reader # this must match the name of the Role or
ClusterRole you wish to bind to
  apiGroup: rbac.authorization.k8s.io
```

A `RoleBinding` may also reference a `ClusterRole` to grant the permissions to namespaced resources defined in the `ClusterRole` within the `RoleBinding`'s namespace. This allows administrators to define a set of common roles for the entire 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 (the namespace of the `RoleBinding`).

```
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "dave" to read secrets in the
"development" namespace.
kind: RoleBinding
metadata:
  name: read-secrets
  namespace: development # This only grants permissions within
the "development" namespace.
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
```

Finally, a `ClusterRoleBinding` may be used to grant permission at the cluster level and in all namespaces. 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 read secrets in any namespace.
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
```

You cannot modify which `Role` or `ClusterRole` a binding object refers to. Attempts to change the `roleRef` field of a binding object will result in a validation error. To change the `roleRef` field on an existing binding object, the binding object must be deleted and recreated. There are two primary reasons for this restriction:

1. 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 accidentally modifying just the roleRef without verifying all of the existing subjects should be given the new role's permissions).
2. Making `roleRef` immutable allows giving `update` permission on an existing binding object to a user, which lets them manage the list of subjects, without being able to change the role that is granted to those subjects.

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

Most resources are represented by a string representation of their name, such as "pods", just as it appears in the URL for the relevant API endpoint. However, some Kubernetes APIs involve a "subresource", such as the logs for a pod. The URL for the pods logs endpoint is:

```
GET /api/v1/namespaces/{namespace}/pods/{name}/log
```

In this case, "pods" is the namespaced resource, 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 both pods and pod logs, you would 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"]
```

Resources can also be referred to by name for certain requests through the `resourceNames` list. When specified, requests can be restricted to individual instances of a resource. To restrict a subject to only "get" and "update" a single configmap, you would write:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: configmap-updater
rules:
- apiGroups: [""]
  resources: ["configmaps"]
  resourceNames: ["my-configmap"]
  verbs: ["update", "get"]
```

Note that `create` requests cannot be restricted by resourceName, as the object name is not known at authorization time. The other exception is `deletecollection`.

Aggregated ClusterRoles

As of 1.9, ClusterRoles can be created by combining other ClusterRoles using an `aggregationRule`. The permissions of aggregated ClusterRoles are controller-managed, and filled in by unioning the rules of any ClusterRole that matches the provided label selector. An example aggregated ClusterRole:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
```

```yaml
metadata:
  name: monitoring
aggregationRule:
  clusterRoleSelectors:
  - matchLabels:
      rbac.example.com/aggregate-to-monitoring: "true"
rules: [] # Rules are automatically filled in by the controller
manager.
```

Creating a ClusterRole that matches the label selector will add rules to the aggregated ClusterRole. In this case rules can be added to the "monitoring" ClusterRole by creating another ClusterRole that has the label `rbac.example.com/aggregate-to-monitoring: true`.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: monitoring-endpoints
  labels:
    rbac.example.com/aggregate-to-monitoring: "true"
# These rules will be added to the "monitoring" role.
rules:
- apiGroups: [""]
  resources: ["services", "endpoints", "pods"]
  verbs: ["get", "list", "watch"]
```

The default user-facing roles (described below) use ClusterRole aggregation. This lets admins include rules for custom resources, such as those served by CustomResourceDefinitions or Aggregated API servers, on the default roles.

For example, the following ClusterRoles let the "admin" and "edit" default roles manage the custom resource "CronTabs" and the "view" role perform read-only actions on the resource.

```yaml
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", "d
elete"]
---
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

Only the `rules` section is shown in the following examples.

Allow reading the resource "pods" in the core API group:

```
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```

Allow reading/writing "deployments" in both the "extensions" and "apps" API groups:

```
rules:
- apiGroups: ["extensions", "apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

Allow reading "pods" and reading/writing "jobs":

```
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["batch", "extensions"]
  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: [""]
  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: [""]
  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 is a suffix glob match
  verbs: ["get", "post"]
```

Referring to Subjects

A `RoleBinding` or `ClusterRoleBinding` binds a role to *subjects*. Subjects can be groups, users or service accounts.

Users are represented by strings. These can be plain usernames, like "alice", email-style names, like "bob@example.com", or numeric IDs represented as a string. It is up to the Kubernetes admin to configure the authentication modules to produce usernames in the desired format. The RBAC authorization system does not require any particular format. However, the prefix `system:` is reserved for Kubernetes system use, and so the admin should ensure usernames do not contain this prefix by accident.

Group information in Kubernetes is currently provided by the Authenticator modules. Groups, like users, are represented as strings, and that string has no format requirements, other than that the prefix `system:` is reserved.

Service Accounts have usernames with the `system:serviceaccount:` prefix and belong to groups with the `system:serviceaccounts:` prefix.

Role Binding Examples

Only the `subjects` section of a `RoleBinding` is shown in the following examples.

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 everywhere:

```
subjects:
- kind: Group
  name: system:serviceaccounts
  apiGroup: rbac.authorization.k8s.io
```

For all authenticated users (version 1.5+):

```
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
```

For all unauthenticated users (version 1.5+):

```
subjects:
- kind: Group
  name: system:unauthenticated
  apiGroup: rbac.authorization.k8s.io
```

For all users (version 1.5+):

```
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 "owned" by the infrastructure. Modifications to these resources can result in non-functional clusters. One example is the `system:node` ClusterRole. This role defines permissions for kubelets. If the role is modified, it can prevent kubelets from working.

All of the default cluster roles and rolebindings are labeled with `kubernetes.io/bootstrapping=rbac-defaults`.

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 to keep roles and rolebindings up-to-date as permissions and subjects change in new releases.

To opt out of this reconciliation, set the `rbac.authorization.kubernetes.io/autoupdate` annotation on a default cluster role or rolebinding to `false`. Be aware that missing default permissions and subjects can result in non-functional clusters.

Auto-reconciliation is enabled in Kubernetes version 1.6+ when the RBAC authorizer is active.

Discovery Roles

Default 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` to the API server configuration.

To view the configuration of these roles via `kubectl` run:

```
kubectl get clusterroles system:discovery -o yaml
```

NOTE: editing the role is not recommended as changes will be overwritten on API server restart via auto-reconciliation (see above).

Default ClusterRole	Default ClusterRoleBinding	Description
system:basic-user	**system:authenticated** group	Allows a user read-only access to basic information about themselves. Prior to 1.14, this role was also bound to \`system:unauthenticated\` by default.
system:discovery	**system:authenticated** group	Allows read-only access to API discovery endpoints needed to discover and negotiate an API level. Prior to 1.14, this role was also bound to \`system:unauthenticated\` by default.
system:public-info-viewer	**system:authenticated** and **system:unauthenticated** groups	Allows read-only access to non-sensitive information about the cluster. Introduced in 1.14.

User-facing Roles

Some of the default roles 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 (`cluster-status`), and roles intended to be granted within particular namespaces using RoleBindings (`admin`, `edit`, `view`).

As of 1.9, user-facing roles use [ClusterRole Aggregation] to allow admins to include rules for custom resources on these roles. To add rules to the "admin", "edit", or "view" role, 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
cluster-admin	**system:masters** group	Allows super-user access to perform any action on any resource. When used in a **ClusterRoleBinding**, it gives full control over every resource in the cluster and in all namespaces. When used in a **RoleBinding**, it gives full control over every resource in the rolebinding's namespace, including the namespace itself.
admin	None	Allows admin access, intended to be granted within a namespace using a **RoleBinding**. If used in a **RoleBinding**, allows read/write access to most resources in a namespace, including the ability to create roles and rolebindings within the namespace. It does not allow write access to resource quota or to the namespace itself.
edit	None	Allows read/write access to most objects in a namespace. It does not allow viewing or modifying roles or rolebindings.
view	None	Allows read-only access to see most objects in a namespace. It does not allow viewing roles or rolebindings. It does not allow viewing secrets, since those are escalating.

Core Component Roles

Default ClusterRole	Default ClusterRoleBinding	Description
system:kube-scheduler	**system:kube-scheduler** user	Allows access to the resources required by the kube-scheduler component.
system:volume-scheduler	**system:kube-scheduler** user	Allows access to the volume resources required by the kube-scheduler component.
system:kube-controller-manager	**system:kube-controller-manager** user	Allows access to the resources required by the kube-controller-manager component. The permissions required by individual control loops are contained in the controller roles.
system:node	None in 1.8+	Allows access to resources required by the kubelet component, **including read access to all secrets, and write access to all pod status objects**. As of 1.7, use of the Node authorizer and NodeRestriction admission plugin is recommended instead of this role, and allow granting API access to kubelets based on the pods scheduled to run on them. Prior to 1.7, this role was automatically bound to the \`system:nodes\` group. In 1.7, this role was automatically bound to the \`system:nodes\` group if the \`Node\` authorization mode is not enabled. In 1.8+, no binding is automatically created.
system:node-proxier	**system:kube-proxy** user	Allows access to the resources required by the kube-proxy component.

Other Component Roles

Default ClusterRole	Default ClusterRoleBinding	Description

system:auth-delegator	None	Allows delegated authentication and authorization checks. This is commonly used by add-on API servers for unified authentication and authorization.
system:heapster	None	Role for the Heapster component.
system:kube-aggregator	None	Role for the kube-aggregator component.
system:kube-dns	**kube-dns** service account in the **kube-system** namespace	Role for the kube-dns component.
system:kubelet-api-admin	None	Allows full access to the kubelet API.
system:node-bootstrapper	None	Allows access to the resources required to perform Kubelet TLS bootstrapping.
system:node-problem-detector	None	Role for the node-problem-detector component.
system:persistent-volume-provisioner	None	Allows access to the resources required by most dynamic volume provisioners.

Controller Roles

The Kubernetes controller manager runs core control loops. When invoked with `--use-service-account-credentials`, each control loop is started using a separate service account. Corresponding roles exist for each control loop, 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.

A user can only create/update a role if at least one of the following things is true:

1. They 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. They are given explicit permission to perform the `escalate` verb on the `roles` or `clusterroles` resource in the `rbac.authorization.k8s.io` API group (Kubernetes 1.12 and newer)

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 the create/update:
    - 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)
    - 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 (Kubernetes 1.12 and newer)

A user can only create/update a role binding if they already have all the permissions contained in the referenced role (at the same scope as the role binding) *or* if they've been given explicit permission 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:
    - implicitly, by giving them the permissions contained in the role.
    - explicitly, by giving them permission to perform the `bind` verb on the particular role (or cluster role).

For example, this cluster role and role binding would allow "user-1" to grant other users the `admin`, `edit`, and `view` roles in the "user-1-namespace" 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"]
  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.
- If your API server runs with the insecure port enabled (`--insecure-port`), you can also make API calls via that port, which does not enforce authentication or authorization.

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 user to perform "get", "watch" and "list" on pods:

  ```
  kubectl create role pod-reader --verb=get --verb=list --verb=watch --resource=pods
  ```

- Create a `Role` named "pod-reader" with resourceNames specified:

  ```
  kubectl create role pod-reader --verb=get --resource=pods --resource-name=readablepod --resource-name=anotherpod
  ```

- Create a `Role` named "foo" with apiGroups specified:

  ```
  kubectl create role foo --verb=get,list,watch --resource=replicasets.apps
  ```

- Create a `Role` named "foo" with subresource permissions:

  ```
  kubectl create role foo --verb=get,list,watch --resource=pods,pods/status
  ```

- 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,watch,update --resource=lease --resource-name=my-component
```

`kubectl create clusterrole`

Creates a `ClusterRole` object. 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 --resource=pods
```

- Create a `ClusterRole` named "pod-reader" with resourceNames specified:

```
kubectl create clusterrole pod-reader --verb=get --resource=pods --resource-name=readablepod --resource-name=anotherpod
```

- Create a `ClusterRole` named "foo" with apiGroups specified:

```
kubectl create clusterrole foo --verb=get,list,watch --resource=replicasets.apps
```

- Create a `ClusterRole` named "foo" with subresource permissions:

```
kubectl create clusterrole foo --verb=get,list,watch --resource=pods,pods/status
```

- Create a `ClusterRole` name "foo" with nonResourceURL specified:

```
kubectl create clusterrole "foo" --verb=get --non-resource-url=/logs/*
```

- Create a `ClusterRole` name "monitoring" with aggregationRule specified:

```
kubectl create clusterrole monitoring --aggregation-rule="rbac.example.com/aggregate-to-monitoring=true"
```

`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=admin --user=bob --namespace=acme
```

- 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 --serviceaccount=acme:myapp --namespace=acme
  ```

- 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 --clusterrole=view --serviceaccount=myappnamespace:myapp --namespace=acme
  ```

`kubectl create clusterrolebinding`

Grants a `ClusterRole` across the entire cluster, including 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 --clusterrole=cluster-admin --user=root
  ```

- 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 --clusterrole=system:node-proxier --user=system:kube-proxy
  ```

- 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 --clusterrole=view --serviceaccount=acme:myapp
  ```

`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
  ```

- 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-subjects --remove-extra-permissions
```

See the CLI help for detailed usage.

Service Account 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 discovery permissions given to all authenticated users).

This allows you to grant particular roles to particular service accounts 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 service accounts, 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 currently 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.

> **Note:** Enabling this means the `kube-system` namespace contains secrets that grant super-user access to the 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 user with read access to secrets or the ability to create a pod to access super-user credentials.

```
kubectl create clusterrolebinding serviceaccounts-cluster-
admin \
  --clusterrole=cluster-admin \
  --group=system:serviceaccounts
```

Upgrading from 1.5

Prior to Kubernetes 1.6, many deployments used very 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 discovery permissions given to all authenticated users).

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=mypolicy.json
```

The RBAC authorizer will attempt to authorize requests first. If it denies an 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 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 apiserver log (prefixed with `RBAC DENY:`). 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 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
> ```

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on May 16, 2019 at 5:03 AM PST by List & watch can be authorized on named resources (#14339) (Page History)

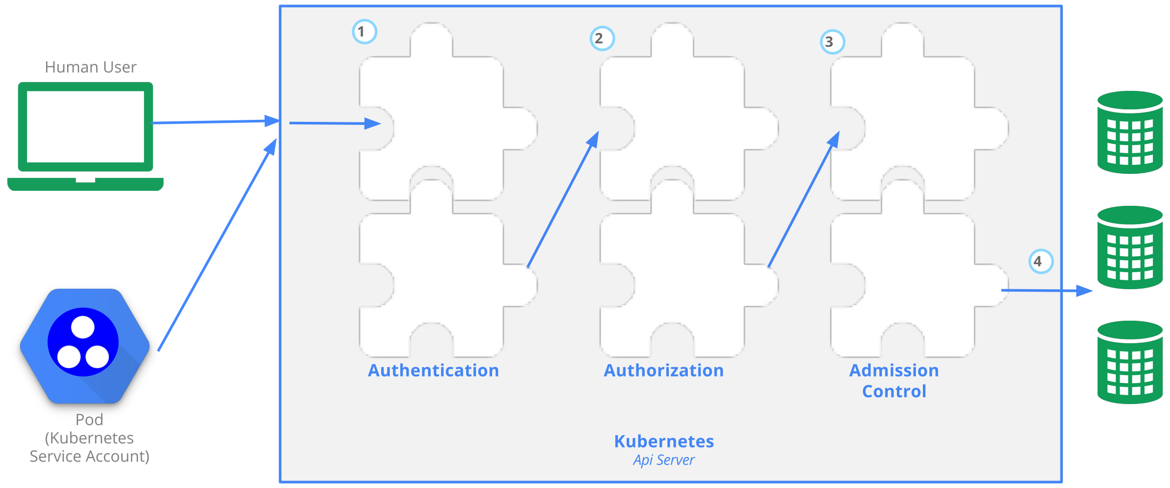
Edit This Page

Controlling Access to the Kubernetes API

This page provides an overview of controlling access to the Kubernetes API.

- Transport Security
- Authentication
- Authorization
- Admission Control
- API Server Ports and IPs

Users access the API using `kubectl`, client libraries, or by making REST requests. Both human users and Kubernetes service accounts can be authorized for API access. When a request reaches the API, it goes through several stages, illustrated in the following diagram:



Transport Security

In a typical Kubernetes cluster, the API serves on port 443. The API server presents a certificate. This certificate is often self-signed, so `$USER/.kube/config` on the user's machine typically contains the root certificate for the API server's certificate, which when specified is used in place of the system default root certificate. This certificate is typically automatically written into your `$USER/.kube/config` when you create a cluster yourself using `kube-up.sh`. If the cluster has multiple users, then the creator needs to share the certificate with other users.

Authentication

Once TLS is established, the HTTP request moves to the Authentication step. This is shown as step **1** in the diagram. The cluster creation script or cluster admin configures the API server to run one or more Authenticator Modules. Authenticators are described in more detail here.

The input to the authentication step is the entire HTTP request, however, it typically just examines the headers and/or client certificate.

Authentication modules include Client Certificates, Password, and Plain Tokens, Bootstrap Tokens, and JWT Tokens (used for service accounts).

Multiple authentication modules can be specified, in which case each one is tried in sequence, until one of them succeeds.

On GCE, Client Certificates, Password, Plain Tokens, and JWT Tokens are all enabled.

If the request cannot be authenticated, it is rejected with HTTP status code 401. Otherwise, the user is authenticated as a specific `username`, and the user name is available to subsequent steps to use in their decisions. Some authenticators also provide the group memberships of the user, while other authenticators do not.

While Kubernetes uses `usernames` for access control decisions and in request logging, it does not have a `user` object nor does it store usernames or other information about users in its object store.

Authorization

After the request is authenticated as coming from a specific user, the request must be authorized. This is shown as stepÂ **2**Â in the diagram.

A request must include the username of the requester, the requested action, and the object affected by the action. The request is authorized if an existing policy declares that the user has permissions to complete the requested action.

For example, if Bob has the policy below, then he can read pods only in the namespace `projectCaribou`:

```
{
    "apiVersion": "abac.authorization.kubernetes.io/v1beta1",
    "kind": "Policy",
    "spec": {
        "user": "bob",
```

```
        "namespace": "projectCaribou",
        "resource": "pods",
        "readonly": true
    }
}
```

If Bob makes the following request, the request is authorized because he is allowed to read objects in the `projectCaribou` namespace:

```
{
  "apiVersion": "authorization.k8s.io/v1beta1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "projectCaribou",
      "verb": "get",
      "group": "unicorn.example.org",
      "resource": "pods"
    }
  }
}
```

If Bob makes a request to write (`create` or `update`) to the objects in the `projectCaribou` namespace, his authorization is denied. If Bob makes a request to read (`get`) objects in a different namespace such as `projectFish`, then his authorization is denied.

Kubernetes authorization requires that you use common REST attributes to interact with existing organization-wide or cloud-provider-wide access control systems. It is important to use REST formatting because these control systems might interact with other APIs besides the Kubernetes API.

Kubernetes supports multiple authorization modules, such as ABAC mode, RBAC Mode, and Webhook mode. When an administrator creates a cluster, they configured the authorization modules that should be used in the API server. If more than one authorization modules are configured, Kubernetes checks each module, and if any module authorizes the request, then the request can proceed. If all of the modules deny the request, then the request is denied (HTTP status code 403).

To learn more about Kubernetes authorization, including details about creating policies using the supported authorization modules, see Authorization Overview.

Admission Control

Admission Control Modules are software modules that can modify or reject requests. In addition to the attributes available to Authorization Modules, Admission Control Modules can access the contents of the object that is being created or updated. They act on objects being created, deleted, updated or connected (proxy), but not reads.

Multiple admission controllers can be configured. Each is called in order.

This is shown as step **3** in the diagram.

Unlike Authentication and Authorization Modules, if any admission controller module rejects, then the request is immediately rejected.

In addition to rejecting objects, admission controllers can also set complex defaults for fields.

The available Admission Control Modules are described here.

Once a request passes all admission controllers, it is validated using the validation routines for the corresponding API object, and then written to the object store (shown as step **4**).

API Server Ports and IPs

The previous discussion applies to requests sent to the secure port of the API server (the typical case). The API server can actually serve on 2 ports:

By default the Kubernetes API server serves HTTP on 2 ports:

1. `Localhost Port`:
   - is intended for testing and bootstrap, and for other components of the master node (scheduler, controller-manager) to talk to the API
   - no TLS
   - default is port 8080, change with `--insecure-port` flag.
   - default IP is localhost, change with `--insecure-bind-address` flag.
   - request **bypasses** authentication and authorization modules.
   - request handled by admission control module(s).
   - protected by need to have host access
2. `Secure Port`:
   - use whenever possible
   - uses TLS. Set cert with `--tls-cert-file` and key with `--tls-private-key-file` flag.
   - default is port 6443, change with `--secure-port` flag.
   - default IP is first non-localhost network interface, change with `--bind-address` flag.
   - request handled by authentication and authorization modules.
   - request handled by admission control module(s).
   - authentication and authorization modules run.

When the cluster is created by `kube-up.sh`, on Google Compute Engine (GCE), and on several other cloud providers, the API server serves on port 443. On GCE, a firewall rule is configured on the project to allow external HTTPS access to the API. Other cluster setup methods vary.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Authenticating

This page provides an overview of authenticating.

- Users in Kubernetes
- Authentication strategies
- Anonymous requests
- User impersonation
- client-go credential plugins

Users in Kubernetes

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

Normal users are assumed to be managed by an outside, independent service. An admin distributing private keys, a user store like Keystone or Google Accounts, even 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.

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, an authenticating proxy, or HTTP basic auth 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 which associate users with a set of commonly grouped users.
- 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 Certs

Client certificate authentication is enabled by passing the `--client-ca-file=SOMEFILE` option to API server. The referenced file must contain one or more certificates 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/O=app1/O=app2"
```

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 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 THETOKEN`. 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

This feature is currently in **beta**.

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`.

Static Password File

Basic authentication is enabled by passing the `--basic-auth-file=SOMEFILE` option to API server. Currently, the basic auth credentials last indefinitely, and the password cannot be changed without restarting API server. Note that basic authentication is currently supported for convenience while we finish making the more secure modes described above easier to use.

The basic auth file is a csv file with a minimum of 3 columns: password, user name, user id. In Kubernetes version 1.6 and later, you can specify an optional fourth column containing comma-separated group names. If you have more than one group, you must enclose the fourth column value in double quotes ("). See the following example:

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

When using basic authentication from an http client, the API server expects an `Authorization` header with a value of `Basic BASE64ENCODED(USER:PASSWORD)`.

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` A file containing a PEM encoded key for signing bearer tokens. If unspecified, the API server's TLS private key will be 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 1.9
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
spec:
  replicas: 3
  template:
    metadata:
    # ...
    spec:
      serviceAccountName: bob-the-bot
      containers:
      - name: nginx
        image: nginx:1.7.9
```

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, simply use the `kubectl create serviceaccount (NAME)` command. This creates a service account in the current namespace and an associated secret.

```
kubectl create serviceaccount jenkins
```

```
serviceaccount "jenkins" created
```

Check an associated secret:

```
kubectl get serviceaccounts jenkins -o yaml
```

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  # ...
secrets:
- name: jenkins-token-1yvwg
```

The created secret holds the public CA of the API server and a signed JSON Web Token (JWT).

```
kubectl get secret jenkins-token-1yvwg -o yaml
```

```yaml
apiVersion: v1
data:
  ca.crt: (APISERVER'S CA BASE64 ENCODED)
  namespace: ZGVmYXVsdA==
  token: (BEARER TOKEN BASE64 ENCODED)
kind: Secret
metadata:
  # ...
type: kubernetes.io/service-account-token
```

> **Note:** Values are base64 encoded because secrets are always base64 encoded.

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 secrets are mounted into pods for in-cluster access to the API server, but can be used from outside the cluster as well.

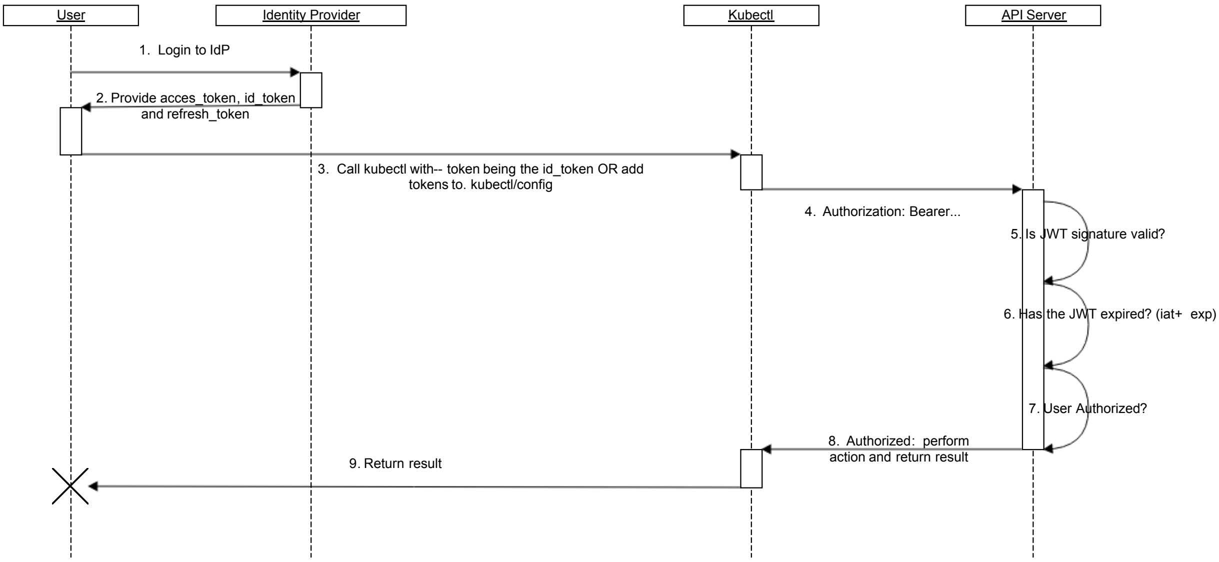
Service accounts authenticate with the username `system:serviceaccount:(NAMESPACE):(SERVICEACCOUNT)`, and are assigned to the groups `system:serviceaccounts` and `system:serviceaccounts:(NAMESPACE)`.

WARNING: Because service account tokens are stored in secrets, any user with read access to those secrets can authenticate as the service account. Be cautious when granting permissions to service accounts and read capabilities for secrets.

OpenID Connect Tokens

OpenID Connect is a flavor of OAuth2 supported by some OAuth2 providers, notably Azure Active Directory, 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.



1. Login 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 by checking against the certificate named in the configuration
6. Check to make sure the `id_token` hasn't expired
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. There's no easy way to authenticate to the Kubernetes dashboard without using the `kubectl proxy` command or a reverse proxy that injects the `id_token`.

Configuring the API Server

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

Parameter	Description	Example	Required
`--oidc-issuer-url`	URL of the provider which allows the API server to discover public signing keys. Only URLs which use the `https://` scheme are accepted. This is typically the provider's discovery URL without a path, for example "https://accounts.google.com" or "https://login.salesforce.com". This URL should point to the level below .well-known/openid-configuration	If the discovery URL is `https://accounts.google.com/.well-known/openid-configuration`, the value should be `https://accounts.google.com`	Yes
`--oidc-client-id`	A client id that all tokens must be issued for.	kubernetes	Yes
`--oidc-username-claim`	JWT claim to use as the user name. By default `sub`, which is expected to be a unique identifier of the end user. Admins can choose other claims, such as `email` or `name`, depending on their provider. However, claims other than `email` will be prefixed with the issuer URL to prevent naming clashes with other plugins.	sub	No
`--oidc-username-prefix`	Prefix prepended to username claims to prevent clashes with existing names (such as `system:` users). For example, the value `oidc:` will create usernames like `oidc:jane.doe`. If this flag isn't provided and `--oidc-user-claim` is a value other than `email` the prefix defaults to `( Issuer URL )#` where `( Issuer URL )` is the value of `--oidc-issuer-url`. The value `-` can be used to disable all prefixing.	`oidc:`	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.	groups	No
`--oidc-groups-prefix`	Prefix prepended to group claims to prevent clashes with existing names (such as `system:` groups). For example, the value `oidc:` will create group names like `oidc:engineering` and `oidc:infra`.	`oidc:`	No

Parameter	Description	Example	Required
`--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.	`claim=value`	No
`--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

Importantly, the API server is not an OAuth2 client, rather it can only be configured to trust a single issuer. This allows the use of public providers, such as Google, without trusting credentials issued to third parties. Admins who wish to utilize multiple OAuth clients should explore providers which support the `azp` (authorized party) claim, a mechanism for allowing one client to issue tokens on behalf of another.

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 CoreOS dex, Keycloak, CloudFoundry UAA, or Tremolo Security's OpenUnison.

For an identity provider to work with Kubernetes it must:

1. Support OpenID connect discovery; not all do.
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 this script from the CoreOS 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.

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 certificate ) \
   --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.lan:8443/auth/idp/OidcIdP  \
        --auth-provider-arg=client-id=kubernetes  \
        --auth-provider-arg=client-secret=1db158f6-177d-4d9c-8a8b
-d36869918ec5  \
        --auth-provider-arg=refresh-token=q1bKLFOyUiosTfawzA93TzZ
IDzH2TNa2SMm0zEiPKTUwME6BkEo6Sql5yUWVBSWpKUGphaWpxSVAfekBOZbBhaEW
+VlFUeVRGcluyVF5JT4+haZmPsluFoFu5XkpXk5BXqHega4GAXlF+ma+vmYpFcHe5
eZR+slBFpZKtQA= \
        --auth-provider-arg=idp-certificate-authority=/root/
ca.pem \
        --auth-provider-arg=id-token=eyJraWQiOiJDTj1vaWRjaWRwLnRy
ZW1vbG8ubGFuLCBPVT1EZW1vLCBPPVRybWVvbG8gU2VjdXJpdHksIEw9QXJsaW5nd
G9uLCBTVD1WaXJnaW5pYSwgQz1VUy1DTj1rdWJlLWNhLTEyMDIxNDc5MjEwMzYwNz
MyMTUyIiwiYWxnIjoiUlMyNTYifQ.eyJpc3MiOiJodHRwczovL29pZGNpZHAudHJl
bW9sby5sYW46ODQ0My9hdXRoL2lkcC9PaWRjSWRQIiwiYXVkIjoia3ViZXJuZXRlc
yIsImV4cCI6MTQ4MzU0OTUxMSwianRpIjoiMm96US15TXdFcHV4WDlHZUhQdy1hZy
IsImlhdCI6MTQ4MzU0OTQ1MSwibmJmIjoxNDgzNTQ5MzMxLCJzdWIiOiI0YWViMzd
iYS1iNjQ1LTQ4ZmQtYWIzMC0xYTAxZWU0MWUyMTgifQ.w6p4J_6qQ1HzTG9nrEOru
bxIMb9K5hzcMPxc9IxPx2K4xO9l-
oFiUw93daH3m5pluP6K7eOE6txBuRVfEcpJSwlelsOsW8gb8VJcnzMS9EnZpeA0tW
_p-mnkFc3VcfyXuhe5R3G7aa5d8uHv70yJ9Y3-
UhjiN9EhpMdfPAoEB9fYKKkJRzF7utTTIPGrSaSU6d2pcpfYKaxIwePzEkT4DfcQt
hoZdy9ucNvvLoi1DIC-
UocFD8HLs8LYKEqSxQvOcvnThbObJ9af71EwmuE21fO5KzMW20KtAeget1gnldOos
Ptz1G5EwvaQ401-RPQzPGMVBld0_zMCAwZttJ4knw
```

Which would produce the below configuration:

```yaml
users:
- name: mmosley
  user:
    auth-provider:
      config:
        client-id: kubernetes
        client-secret: 1db158f6-177d-4d9c-8a8b-d36869918ec5
```

```
        id-token: eyJraWQiOiJDTj1vaWRjaWRwLnRyZW1vbG8ubGFuLCBPVT1
EZW1vLCBPPVRybWVvbG8gU2VjdXJpdHksIEw9QXJsaW5ndG9uLCBTVD1WaXJnaW5p
YSwgQz1VUy1DTj1rdWJlLWNhLTEyMDIxNDc5MjEwMzYwNzMyMTUyIiwiYWxnIjoiU
lMyNTYifQ.eyJpc3MiOiJodHRwczovL29pZGNpZHAudHJlbW9sby5sYW46ODQ0My9
hdXRoL2lkcC9PaWRjSWRQIiwiYXVkIjoia3ViZXJuZXRlcyIsImV4cCI6MTQ4MzU0
OTUxMSwianRpIjoiMm96US15TXdFcHV4WDlHZUhQdy1hZyIsImlhdCI6MTQ4MzU0O
TQ1MSwibmJmIjoxNDgzNTQ5MzMxLCJzdWIiOiI0YWViMzdiYS1iNjQ1LTQ4ZmQtYW
IzMC0xYTAxZWU0MWUyMTgifQ.w6p4J_6qQ1HzTG9nrEOrubxIMb9K5hzcMPxc9IxP
x2K4xO9l-
oFiUw93daH3m5pluP6K7eOE6txBuRVfEcpJSwlelsOsW8gb8VJcnzMS9EnZpeA0tW
_p-mnkFc3VcfyXuhe5R3G7aa5d8uHv70yJ9Y3-
UhjiN9EhpMdfPAoEB9fYKKkJRzF7utTTIPGrSaSU6d2pcpfYKaxIwePzEkT4DfcQt
hoZdy9ucNvvLoi1DIC-
UocFD8HLs8LYKEqSxQvOcvnThbObJ9af71EwmuE21fO5KzMW20KtAeget1gnldOos
Ptz1G5EwvaQ401-RPQzPGMVBld0_zMCAwZttJ4knw
        idp-certificate-authority: /root/ca.pem
        idp-issuer-url: https://oidcidp.tremolo.lan:8443/auth/
idp/OidcIdP
        refresh-token: q1bKLFOyUiosTfawzA93TzZIDzH2TNa2SMm0zEiPKT
UwME6BkEo6Sql5yUWVBSWpKUGphaWpxSVAfekBOZbBhaEW+VlFUeVRGcluyVF5JT4
+haZmPsluFoFu5XkpXk5BXq
      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. Simply copy and paste the `id_token` into this option:

```
kubectl --token=eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL21sYi50
cmVtb2xvLmxhbjo4MDQzL2F1dGgvaWRwL29pZGMiLCJhdWQiOiJrdWJlcm5ldGVzI
iwiZXhwIjoxNDc0NTk2NjY5LCJqdGkiOiI2RDUzNXoxUEpFNjJOR3QxaWVyYm9RIi
wiaWF0IjoxNDc0NTk2MzY5LCJuYmYiOjE0NzQ1OTYyNDksInN1YiI6Im13aW5kdSI
sInVzZXJfcm9sZSI6WyJ1c2VycyIsIm5ldy1uYW1lc3BhY2Utdmlld2VyIl0sImVt
YWlsIjoibXdpbmR1QG5vbW9yZWplZGkuY29tIn0.f2As579n9VNoaKzoF-
dOQGmXkFKf1FMyNV0-va_B63jn-_n9LGSCca_6IVMP8pO-
Zb4KvRqGyTP0r3HkHxYy5c81AnIh8ijarruczl-
TK_yF5akjSTHFZD-0gRzlevBDiH8Q79NAr-
ky0P4iIXS8lY9Vnjch5MF74Zx0c3alKJHJUnnpjIACByfF2SCaYzbWFMUNat-
K1PaUk5-ujMBG7yYnr95xD-63n8CO8teGUAAEMx6zRjzfhnhbzX-
ajwZLGwGUBT4WqjMs70-6a7_8gZmLZb2az1cZynkFRj2BaCkVT3A2RrjeEwZEtGXl
MqKJ1_I2ulrOVsYx01_yD35-rw get nodes
```

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.

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
verifying the remote service.
      server: https://authn.example.com/authenticate # URL of
remote service to query. Must use 'https'.

# 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 to use
      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-sever
  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 `authentication.k8s.io/v1beta1` `TokenReview` object containing the token to the remote service. Kubernetes will not challenge a request that lacks such a header.

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 `authentication.k8s.io/v1beta1` API extensions group (`--runtime-config=authentication.k8s.io/v1beta1=true`).

The POST body will be of the following format:

```
{
  "apiVersion": "authentication.k8s.io/v1beta1",
  "kind": "TokenReview",
  "spec": {
```

```json
    "token": "(BEARERTOKEN)"
  }
}
```

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. A successful validation of the bearer token would return:

```json
{
  "apiVersion": "authentication.k8s.io/v1beta1",
  "kind": "TokenReview",
  "status": {
    "authenticated": true,
    "user": {
      "username": "janedoe@example.com",
      "uid": "42",
      "groups": [
        "developers",
        "qa"
      ],
      "extra": {
        "extrafield1": [
          "extravalue1",
          "extravalue2"
        ]
      }
    }
  }
}
```

An unsuccessful request would return:

```json
{
  "apiVersion": "authentication.k8s.io/v1beta1",
  "kind": "TokenReview",
  "status": {
    "authenticated": false
  }
}
```

HTTP status codes can be used to supply additional error context.

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 names (cn). If set, a valid client certificate with a Common Name (cn) in the specified list must be presented before the request headers are checked for user names. If empty, any Common Name 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.

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 )` should be lower-case, and any characters which aren't legal in HTTP header labels MUST be utf8 and percent-encoded.

  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.

An example set of headers:

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

```
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
```

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 cluster scope. (get nodes mynode)
```

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

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

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

To impersonate a user, group, or set extra fields, the impersonating user must have the ability to perform the "impersonate" verb on the kind of attribute being impersonated ("user", "group", 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"]
```

Extra fields are evaluated as sub-resources of the resource "userextras". To allow a user to use impersonation headers for the extra field "scopes," a user should be granted the following role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: scopes-impersonator
rules:
# Can set "Impersonate-Extra-scopes" header.
- apiGroups: ["authentication.k8s.io"]
  resources: ["userextras/scopes"]
  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"]
```

client-go credential plugins

FEATURE STATE: `Kubernetes v1.11` beta
This feature is currently in a *beta* state, meaning:

- The version names contain beta (e.g. v2beta3).
- Code is well tested. Enabling the feature is considered safe. Enabled by default.
- Support for the overall feature will not be dropped, though details may change.
- The schema and/or semantics of objects may change in incompatible ways in a subsequent beta or stable release. When this happens, we will provide instructions for migrating to the next version. This may require deleting, editing, and re-creating API objects. The editing process may require some thought. This may require downtime for applications that rely on the feature.
- Recommended for only non-business-critical uses because of potential for incompatible changes in subsequent releases. If you have multiple clusters that can be upgraded independently, you may be able to relax this restriction.
- **Please do try our beta features and give feedback on them! After they exit beta, it may not be practical for us to make more changes.**

`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.

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.

```
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/v1alpha1),
      # set an environment variable or pass an argument to the tool that indicates which version the exec plugin expects.
      apiVersion: "client.authentication.k8s.io/v1beta1"

      # 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"
clusters:
- name: my-cluster
  cluster:
    server: "https://172.17.4.100:6443"
    certificate-authority: "/etc/kubernetes/ca.pem"
```

```
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/v1beta1"
```

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`.

When run from an interactive session, `stdin` is exposed directly to the plugin. Plugins should use a TTY check to determine if it's appropriate to prompt a user interactively.

To use bearer token credentials, the plugin returns a token in the status of the `ExecCredential`.

```
{
  "apiVersion": "client.authentication.k8s.io/v1beta1",
  "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.

```
{
  "apiVersion": "client.authentication.k8s.io/v1beta1",
  "kind": "ExecCredential",
  "status": {
    "clientCertificateData": "-----BEGIN CERTIFICATE-----\n...
\n-----END CERTIFICATE-----",
    "clientKeyData": "-----BEGIN RSA PRIVATE KEY-----\n...\n-----
```

```
END RSA PRIVATE KEY-----"
  }
}
```

Optionally, the response can include the expiry of the credential formatted as a RFC3339 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.

```
{
  "apiVersion": "client.authentication.k8s.io/v1beta1",
  "kind": "ExecCredential",
  "status": {
    "token": "my-bearer-token",
    "expirationTimestamp": "2018-03-05T17:30:20-08:00"
  }
}
```

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on March 07, 2019 at 5:54 PM PST by Updated link to page (#12862) (Page History)

Edit This Page

Authenticating with Bootstrap Tokens

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
- Token Format
- Enabling Bootstrap Token Authentication
- Bootstrap Token Secret Format
- Token Management with kubeadm

- ConfigMap Signing

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.

FEATURE STATE: `Kubernetes v1.15` beta
This feature is currently in a *beta* state, meaning:

- The version names contain beta (e.g. v2beta3).
- Code is well tested. Enabling the feature is considered safe. Enabled by default.
- Support for the overall feature will not be dropped, though details may change.
- The schema and/or semantics of objects may change in incompatible ways in a subsequent beta or stable release. When this happens, we will provide instructions for migrating to the next version. This may require deleting, editing, and re-creating API objects. The editing process may require some thought. This may require downtime for applications that rely on the feature.
- Recommended for only non-business-critical uses because of potential for incompatible changes in subsequent releases. If you have multiple clusters that can be upgraded independently, you may be able to relax this restriction.
- **Please do try our beta features and give feedback on them! After they exit beta, it may not be practical for us to make more changes.**

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 init'."

  # 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:bootstrappers:"
  auth-extra-groups: system:bootstrappers:worker,system:bootstrap
pers:ingress
```

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
..tYEfbo6zDNo40MQE07aZcQX2m3EB2rO3NuXtxVMYm9U
  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 just 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 security model section for more information.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on November 06, 2018 at 11:33 AM PST by Add admonition type to shortcode (#9482) (Page History)

Edit This Page

Using Admission Controllers

This page provides an overview of Admission Controllers.

- What are they?
- Why do I need them?
- How do I turn on an admission controller?
- How do I turn off an admission controller?
- Which plugins are enabled by default?
- What does each admission controller do?
- Is there a recommended set of admission controllers to use?

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. The controllers 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 controllers may be "validating", "mutating", or both. Mutating controllers may modify the objects they admit; validating controllers may not.

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?

Many advanced features in 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,Limi
tRanger ...
```

> **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,Always
Deny ...
```

Which plugins are enabled by default?

To see which admission plugins are enabled:

```
kube-apiserver -h | grep enable-admission-plugins
```

In 1.14, they are:

```
NamespaceLifecycle, LimitRanger, ServiceAccount,
TaintNodesByCondition, Priority, DefaultTolerationSeconds,
DefaultStorageClass, PersistentVolumeClaimResize,
MutatingAdmissionWebhook, ValidatingAdmissionWebhook,
ResourceQuota
```

What does each admission controller do?

AlwaysAdmit

FEATURE STATE: `Kubernetes v1.13` deprecated
This feature is *deprecated*. For more information on this state, see the Kubernetes Deprecation Policy.

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.

AlwaysPullImages

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 simply 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.

AlwaysDeny

FEATURE STATE: `Kubernetes v1.13` deprecated
This feature is *deprecated*. For more information on this state, see the Kubernetes Deprecation Policy.

Rejects all requests. AlwaysDeny is DEPRECATED as no real meaning.

DefaultStorageClass

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

This admission controller sets the default forgiveness toleration for pods to tolerate the taints `notready:NoExecute` and `unreachable:NoExecute` for 5 minutes, if the pods don't already have toleration for taints `node.kubernetes.io/not-ready:NoExecute` or `node.alpha.kubernetes.io/unreachable:NoExecute`.

DenyExecOnPrivileged

FEATURE STATE: `Kubernetes v1.13` deprecated
This feature is *deprecated*. For more information on this state, see the Kubernetes Deprecation Policy.

This admission controller will intercept all requests to exec a command in a pod if that pod has a privileged container.

This functionality has been merged into DenyEscalatingExec. The DenyExecOnPrivileged admission plugin is deprecated and will be removed in v1.18.

Use of a policy-based admission plugin (like PodSecurityPolicy or a custom admission plugin) which can be targeted at specific users or Namespaces and also protects against creation of overly privileged Pods is recommended instead.

DenyEscalatingExec

FEATURE STATE: `Kubernetes v1.13` deprecated
This feature is *deprecated*. For more information on this state, see the Kubernetes Deprecation Policy.

This admission controller will deny exec and attach commands to pods that run with escalated privileges that allow host access. This includes pods that run as privileged, have access to the host IPC namespace, and have access to the host PID namespace.

The DenyEscalatingExec admission plugin is deprecated and will be removed in v1.18.

Use of a policy-based admission plugin (like PodSecurityPolicy or a custom admission plugin) which can be targeted at specific users or Namespaces and also protects against creation of overly privileged Pods is recommended instead.

EventRateLimit

FEATURE STATE: `Kubernetes v1.13` alpha

This feature is currently in a *alpha* state, meaning:

- The version names contain alpha (e.g. v1alpha1).
- Might be buggy. Enabling the feature may expose bugs. Disabled by default.
- Support for feature may be dropped at any time without notice.
- The API may change in incompatible ways in a later software release without notice.
- Recommended for use only in short-lived testing clusters, due to increased risk of bugs and lack of long-term support.

This admission controller mitigates the problem where the API server gets flooded by event requests. The cluster admin can specify event rate limits by:

- Ensuring that `eventratelimit.admission.k8s.io/v1alpha1=true` is included in the `--runtime-config` flag for the API server;
- 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.k8s.io/v1alpha1
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 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 proposal for more details.

ExtendedResourceToleration

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.

ImagePolicyWebhook

The ImagePolicyWebhook admission controller allows a backend webhook to make admission decisions.

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.k8s.io/v1alpha1
kind: AdmissionConfiguration
plugins:
- name: ImagePolicyWebhook
  path: imagepolicyconfig.yaml
...
```

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 remote service.
    server: https://images.example.com/policy # URL of remote service to query. Must use 'https'.

# 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
```

```
admission controller to use
    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 `*.image-policy.k8s.io/*`.

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 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:beb6bd6a68f114c1dc2ea4b28db81bdf91de202a9014972bec
5e4d9171d90ed"
      }
    ],
    "annotations":{
      "mycluster.image-policy.k8s.io/ticket-1234": "break-glass"
    },
    "namespace":"mynamespace"
  }
}
```


The remote service is expected to fill the `ImageReviewStatus` 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 objects and `plugin/pkg/admission/imagepolicy/admission.go`.

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. In the future, if an annotation is determined to be widely useful, it may be promoted to a named field of `ImageReviewSpec`.

LimitPodHardAntiAffinityTopology

This admission controller denies any pod that defines `AntiAffinity` topology key other than `kubernetes.io/hostname` in `requiredDuringSchedulingRequiredDuringExecution`.

LimitRanger

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 design doc and the example of Limit Range for more details.

MutatingAdmissionWebhook

FEATURE STATE: `Kubernetes v1.13` beta
This feature is currently in a *beta* state, meaning:

- The version names contain beta (e.g. v2beta3).

- Code is well tested. Enabling the feature is considered safe. Enabled by default.
- Support for the overall feature will not be dropped, though details may change.
- The schema and/or semantics of objects may change in incompatible ways in a subsequent beta or stable release. When this happens, we will provide instructions for migrating to the next version. This may require deleting, editing, and re-creating API objects. The editing process may require some thought. This may require downtime for applications that rely on the feature.
- Recommended for only non-business-critical uses because of potential for incompatible changes in subsequent releases. If you have multiple clusters that can be upgraded independently, you may be able to relax this restriction.
- **Please do try our beta features and give feedback on them! After they exit beta, it may not be practical for us to make more changes.**

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/v1beta1` group/version via the `--runtime-config` flag (both are on by default in versions >= 1.9).

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.
- This is a beta feature. Future versions of Kubernetes may restrict the types of mutations these webhooks can make.
- 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

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

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

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

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. In Kubernetes 1.11+, kubelets are not allowed to update or remove taints from their `Node` API object.

In Kubernetes 1.13+, 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`
    - `failure-domain.beta.kubernetes.io/region`
    - `failure-domain.beta.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

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.

PersistentVolumeLabel

FEATURE STATE: `Kubernetes v1.13` deprecated
This feature is *deprecated*. For more information on this state, see the Kubernetes Deprecation Policy.

This admission controller automatically attaches region or zone labels to PersistentVolumes as defined by the cloud provider (for example, GCE or AWS). It helps ensure the Pods and the PersistentVolumes mounted are in the same region and/or zone. If the admission controller doesn't support automatic labelling your PersistentVolumes, you may need to add the labels manually to prevent pods from mounting volumes from a different zone. PersistentVolumeLabel is DEPRECATED and labeling persistent volumes has been taken over by cloud controller manager. Starting from 1.11, this admission controller is disabled by default.

PodNodeSelector

This admission controller defaults and limits what node selectors may be used within a namespace by reading a namespace annotation and a global configuration.

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.k8s.io/v1alpha1
kind: AdmissionConfiguration
plugins:
- name: PodNodeSelector
  path: podnodeselector.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-selector
  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 whitelist 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.

PersistentVolumeClaimResize

This admission controller implements additional validations for checking incoming `PersistentVolumeClaim` resize requests.

> **Note:** Support for volume resizing is available as an alpha feature. Admins must set the feature gate `ExpandPersistentVolumes` to `true` to enable resizing.

After enabling the `ExpandPersistentVolumes` feature gate, enabling the `PersistentVolumeClaimResize` admission controller is recommended, too. 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.

PodPreset

This admission controller injects a pod with the fields specified in a matching PodPreset. See also PodPreset concept and Inject Information into Pods Using a PodPreset for more information.

PodSecurityPolicy

This admission controller acts on creation and modification of the pod and determines if it should be admitted based on the requested security context and the available Pod Security Policies.

For Kubernetes < 1.6.0, the API Server must enable the extensions/v1beta1/podsecuritypolicy API extensions group (`--runtime-config=extensions/v1beta1/podsecuritypolicy=true`).

See also Pod Security Policy documentation for more information.

PodTolerationRestriction

This admission controller first verifies any conflict between a pod's tolerations and its namespace's tolerations, and rejects the pod request if there is a conflict. It then merges the namespace's tolerations into the pod's tolerations. The resulting tolerations are checked against the namespace's whitelist of tolerations. If the check succeeds, the pod request is admitted otherwise rejected.

If the pod's namespace does not have any associated default or whitelist of tolerations, then the cluster-level default or whitelist of tolerations are used instead if specified.

Tolerations to a namespace are assigned via the `scheduler.alpha.kubernetes.io/defaultTolerations` and `scheduler.alpha.kubernetes.io/tolerationsWhitelist` annotation keys.

Priority

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

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 design doc and the example of Resource Quota for more details.

SecurityContextDeny

This admission controller will deny any pod that attempts to set certain escalating SecurityContext fields. This should be enabled if a cluster doesn't utilize pod security policies to restrict the set of values a security context can take.

ServiceAccount

This admission controller implements automation for serviceAccounts. We strongly recommend using this admission controller if you intend to make use of Kubernetes `ServiceAccount` objects.

StorageObjectInUseProtection

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.

ValidatingAdmissionWebhook

FEATURE STATE: `Kubernetes v1.13` beta
This feature is currently in a *beta* state, meaning:

- The version names contain beta (e.g. v2beta3).
- Code is well tested. Enabling the feature is considered safe. Enabled by default.
- Support for the overall feature will not be dropped, though details may change.
- The schema and/or semantics of objects may change in incompatible ways in a subsequent beta or stable release. When this happens, we will provide instructions for migrating to the next version. This may require deleting, editing, and re-creating API objects. The editing process may require some thought. This may require downtime for applications that rely on the feature.
- Recommended for only non-business-critical uses because of potential for incompatible changes in subsequent releases. If you have multiple clusters that can be upgraded independently, you may be able to relax this restriction.
- **Please do try our beta features and give feedback on them! After they exit beta, it may not be practical for us to make more changes.**

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/v1beta1` group/version via the `--runtime-config` flag (both are on by default in versions 1.9 and later).

Is there a recommended set of admission controllers to use?

Yes. For Kubernetes version 1.10 and later, 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**).

> **Note:** `--admission-control` was deprecated in 1.10 and replaced with `--enable-admission-plugins`.

For Kubernetes 1.9 and earlier, we recommend running the following set of admission controllers using the `--admission-control` flag (**order matters**).

- v1.9

```
  --admission-control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,DefaultTolerationSeconds,MutatingAdmissionWebhook,ValidatingAdmissionWebhook,ResourceQuota
```

- It's worth reiterating that in 1.9, these happen in a mutating phase and a validating phase, and that e.g. `ResourceQuota` runs in the validating phase, and therefore is the last admission controller to run. `MutatingAdmissionWebhook` appears before it in this list, because it runs in the mutating phase.

  For earlier versions, there was no concept of validating vs mutating and the admission controllers ran in the exact order specified.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on July 06, 2019 at 7:16 AM PST by correct request payload for imagepolicy webhook (#15228) (Page History)

Edit This Page

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, and use admission webhooks.

- What are admission webhooks?
- Experimenting with admission webhooks
- Webhook request and response
- Webhook configuration

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 the Kubernetes cluster is at least as new as v1.9.
- 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/v1beta1 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 apiservers, 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 apiservers. If you need mutual TLS or other ways to authenticate the clients, see how to authenticate apiservers.

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 client 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/v1beta1
kind: ValidatingWebhookConfiguration
metadata:
  name: <name of this configuration object>
webhooks:
- name: <webhook name, e.g., pod-policy.example.io>
  rules:
  - apiGroups:
    - ""
    apiVersions:
    - v1
    operations:
    - CREATE
    resources:
    - pods
    scope: "Namespaced"
  clientConfig:
    service:
      namespace: <namespace of the front-end service>
      name: <name of the front-end service>
    caBundle: <pem encoded ca cert that signs the server cert
used by the webhook>
  admissionReviewVersions:
  - v1beta1
  timeoutSeconds: 1
```

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 30 seconds but starting in kubernetes 1.14 you can set the timeout and it is encouraged to use a very small timeout for webhooks. If the webhook call times out, the request is handled according to the webhook's failure policy.

When an apiserver receives a request that matches one of the `rules`, the apiserver 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 apiservers

If your admission webhooks require authentication, you can configure the apiservers 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 apiserver, 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:

```
apiVersion: apiserver.k8s.io/v1alpha1
kind: AdmissionConfiguration
plugins:
- name: ValidatingAdmissionWebhook
  configuration:
    apiVersion: apiserver.config.k8s.io/v1alpha1
    kind: WebhookAdmission
    kubeConfigFile: <path-to-kubeconfig-file>
- name: MutatingAdmissionWebhook
  configuration:
    apiVersion: apiserver.config.k8s.io/v1alpha1
    kind: WebhookAdmission
    kubeConfigFile: <path-to-kubeconfig-file>
```

The schema of `admissionConfiguration` is defined here. See the webhook configuration section for details about each config field.

- In the kubeConfig file, provide the credentials:

```
apiVersion: v1
kind: Config
users:
# DNS name of webhook service, i.e., <service name>.<namespace>.svc, or the URL
# of the webhook server.
- name: 'webhook1.ns1.svc'
  user:
    client-certificate-data: <pem encoded certificate>
    client-key-data: <pem encoded key>
# The `name` supports using * to wildmatch 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 authentications.

Webhook request and response

Request

Webhooks are sent a POST request, 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/v1beta1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  admissionReviewVersions: ["v1beta1"]
  ...
```

If no `admissionReviewVersions` are specified, the default when creating `admissionregistration.k8s.io/v1beta1` webhook configurations is `v1beta1`.

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/v1beta1",
  "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 m
odified
    "resource": {"group":"apps","version":"v1","resource":"deploy
ments"},
    // subresource, if the request is to a subresource
    "subResource": "scale",
```

```
    // Fully-qualified group/version/kind of the incoming object in the original request to the API server.
    // This only differs from `kind` if the webhook specified `matchPolicy: Equivalent` and the
    // original request to the API server was converted to a version the webhook registered for.
    // Only sent by v1.15+ API servers.
    "requestKind": {"group":"autoscaling","version":"v1","kind":"Scale"},
    // Fully-qualified group/version/kind of the resource being modified in the original request to the API server.
    // This only differs from `resource` if the webhook specified `matchPolicy: Equivalent` and the
    // original request to the API server was converted to a version the webhook registered for.
    // Only sent by v1.15+ API servers.
    "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: Equivalent` and the
    // original request to the API server was converted to a version the webhook registered for.
    // Only sent by v1.15+ API servers.
    "requestSubResource": "scale",

    // Name of the resource being modified
    "name": "my-deployment",
    // Namespace of the resource being modified, if the resource is namespaced (or is a Namespace object)
    "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 server
      "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 to the API server
      "groups": ["system:authenticated","my-admin-group"],
      // Arbitrary extra info associated with the user making the request to the API server.
      // This is populated by the API server authentication layer and should be included
      // 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 (and for DELETE operations in API servers prior to v1.15.0)
    "oldObject": {"apiVersion":"autoscaling/v1","kind":"Scale",...},
    // options contains the options for the operation being admitted, like meta.k8s.io/v1 CreateOptions, UpdateOptions, or DeleteOptions.
    // It is null for CONNECT operations.
    // Only sent by v1.15+ API servers.
    "options": {"apiVersion":"meta.k8s.io/v1","kind":"UpdateOptions",...},

    // dryRun indicates the API request is running in dry run mode and will not be persisted.
    // Webhooks with side effects should avoid actuating those side effects when dryRun is true.
    // See http://k8s.io/docs/reference/using-api/api-concepts/#make-a-dry-run-request for more details.
    "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/v1beta1",
  "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/v1beta1",
  "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 [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/v1beta1",
  "kind": "AdmissionReview",
  "response": {
    "uid": "<value from request.uid>",
    "allowed": false,
    "status": {
      "code": 403,
      "message": "You cannot do this because it is Tuesday and
your name starts with A"
    }
  }
}
```

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 `W3sib3AiOiAiYWRkIiwgInBhdGgiOiAiL3NwZWMvcmVwbGljYXMiLCAidmFsdWUiOiAzfV0=`

So a webhook response to add that label would be:

```
{
  "apiVersion": "admission.k8s.io/v1beta1",
  "kind": "AdmissionReview",
  "response": {
    "uid": "<value from request.uid>",
    "allowed": true,
    "patchType": "JSONPatch",
    "patch": "W3sib3AiOiAiYWRkIiwgInBhdGgiOiAiL3NwZWMvcmVwbGljYXM
iLCAidmFsdWUiOiAzfV0="
  }
}
```

Webhook configuration

To register admission webhooks, create `MutatingWebhookConfiguration` or `ValidatingWebhookConfiguration` API objects. Each configuration can contain one or more webhooks. 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. Supported in v1.14+. Default is `"*"`, matching pre-1.14 behavior.
  - `"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`:

```yaml
apiVersion: admissionregistration.k8s.io/v1beta1
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/v1beta1
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/v1beta1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  rules:
  - operations: ["UPDATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*/status"]
    scope: "*"
  ...
```

Matching requests: objectSelector

In v1.15+, 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 with the label `foo: bar`:

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  objectSelector:
```

```
    matchLabels:
      foo: bar
  rules:
  - operations: ["CREATE"]
    apiGroups: ["*"]
    apiVersions: ["*"]
    resources: ["*"]
    scope: "*"
  ...
```

See https://kubernetes.io/docs/concepts/overview/working-with-objects/labels 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/v1beta1
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/v1beta1
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 https://kubernetes.io/docs/concepts/overview/working-with-objects/labels for more examples of label selectors.

Matching requests: matchPolicy

API servers can make objects available via multiple API groups or versions. For example, the Kubernetes API server allows creating and modifying `Deployment` objects via `extensions/v1beta1`, `apps/v1beta1`, `apps/v1beta2`, and `apps/v1` APIs.

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.

In v1.15+, `matchPolicy` lets a webhook define how its `rules` are used to match incoming requests. Allowed values are `Exact` or `Equivalent`. The default in `v1beta1` is `Exact`.

- `Exact` means a request should be intercepted only if it exactly matches a specified rule.
- `Equivalent` means a request should be intercepted if 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, deprecated `extensions/v1beta1` deployments are scheduled to stop being served by default in v1.16. Once that occurs, a webhook with a `apiGroups:["extensions"], apiVersions:["v1beta1"], resources:["deployments"]` rule would no longer 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/v1beta1
kind: ValidatingWebhookConfiguration
```

```yaml
...
webhooks:
- name: my-webhook.example.com
  matchPolicy: Equivalent
  rules:
  - operations: ["CREATE","UPDATE","DELETE"]
    apiGroups: ["apps"]
    apiVersions: ["v1"]
    resources: ["deployments"]
    scope: "Namespaced"
  ...
```

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 apiservers (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 apiserver which might need to make calls to this webhook. Such installs are likely to be non-portable, i.e., not easy to turn up in a new cluster.

The scheme must be "https"; the URL must begin with "https://".

Attempting to use a user or basic auth e.g. "user:password@" is not allowed. Fragments ("#â€¦") and query parameters ("?â€¦") 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):

```yaml
apiVersion: admissionregistration.k8s.io/v1beta1
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/v1beta1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  clientConfig:
    caBundle: "Ci0tLS0tQk...<base64-encoded PEM bundle>...tLS0K"
    service:
      namespace: my-service-namespace
      name: my-service-name
      path: /my-path
      port: 1234
  ...
```

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 reconcilation 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 should 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. `sideEffects` may be set to `Unknown`, `None`, `Some`, `NoneOnDryRun`. The default is `Unknown`.

- `Unknown`: no information is known about the side effects of calling the webhook. If a request with `dryRun: true` would trigger a call to this webhook, the request will instead fail, and the webhook will not be called.
- `None`: calling the webhook will have no side effects.
- `Some`: calling the webhook will possibly have side effects. If a request with the dry-run attribute would trigger a call to this webhook, the request will instead fail, and the webhook will not be called.

- `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/v1beta1
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, and defaults to 30 seconds.

Here is an example of a validating webhook with a custom timeout of 2 seconds:

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  timeoutSeconds: 2
  ...
```

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).

In v1.15+, 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/v1beta1
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`. Defaults to `Ignore` in v1beta1.

- `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/v1beta1
kind: MutatingWebhookConfiguration
...
webhooks:
- name: my-webhook.example.com
  failurePolicy: Fail
  ...
```

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on July 03, 2019 at 7:22 PM PST by Fix typo (#15265) (Page History)

Edit This Page

Managing Service Accounts

This is a Cluster Administrator guide to service accounts. It assumes knowledge of the User Guide to Service Accounts.

Support for authorization and user accounts is planned but incomplete. Sometimes incomplete features are referred to in order to better describe service accounts.

- User accounts vs service accounts
- Service account automation

User accounts vs 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 processes, which run in pods.
- User accounts are intended to be global. Names must be unique across all namespaces of a cluster, future user resource will not be namespaced. Service accounts are namespaced.
- Typically, a cluster's User accounts might be synced from a corporate database, where new user account creation requires special privileges and is tied to complex business processes. Service account creation is intended to be more lightweight, allowing cluster users to create service accounts for specific tasks (i.e. principle of least privilege).
- Auditing considerations for humans and service accounts may differ.
- A config bundle for a complex system may include definition of various service accounts for components of that system. Because service accounts can be created ad-hoc and have namespaced names, such config is portable.

Service account automation

Three separate components cooperate to implement the automation around service accounts:

- A Service account admission controller
- A Token controller
- A Service account controller

Service Account Admission Controller

The modification of pods is implemented via a plugin called an Admission Controller. It is part of the apiserver. It acts synchronously to modify pods as they are created or updated. When this plugin is active (and it is by default on most distributions), then it does the following when a pod is created or modified:

1. If the pod does not have a `ServiceAccount` set, it sets the `ServiceAccount` to `default`.
2. It ensures that the `ServiceAccount` referenced by the pod exists, and otherwise rejects it.
3. If the pod does not contain any `ImagePullSecrets`, then `ImagePullSecrets` of the `ServiceAccount` are added to the pod.
4. It adds a `volume` to the pod which contains a token for API access.
5. It adds a `volumeSource` to each container of the pod mounted at `/var/run/secrets/kubernetes.io/serviceaccount`.

Starting from v1.13, you can migrate a service account volume to a projected volume when the `BoundServiceAccountTokenVolume` feature gate is enabled. The service account token will expire after 1 hour or the pod is deleted. See more details about projected volume.

Token Controller

TokenController runs as part of controller-manager. It acts asynchronously. It:

- observes serviceAccount creation and creates a corresponding Secret to allow API access.
- observes serviceAccount deletion and deletes all corresponding ServiceAccountToken Secrets.
- observes secret addition, and ensures the referenced ServiceAccount exists, and adds a token to the secret if needed.
- observes 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 controller-manager by using the `--service-account-private-key-file` option. The private key will be 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` option. The public key will be used to verify the tokens during authentication.

To create additional API tokens

A controller loop ensures a secret with an API token exists for each service account. To create additional API tokens for a service account, create a secret of type `ServiceAccountToken` with an annotation referencing the service account, and the controller will update it with a generated token:

secret.json:

```
{
    "kind": "Secret",
    "apiVersion": "v1",
    "metadata": {
        "name": "mysecretname",
```

```
        "annotations": {
            "kubernetes.io/service-account.name": "myserviceaccou
nt"
        }
    },
    "type": "kubernetes.io/service-account-token"
}
```

```
kubectl create -f ./secret.json
kubectl describe secret mysecretname
```

To delete/invalidate a service account token

```
kubectl delete secret mysecretname
```

Service Account Controller

Service Account Controller manages ServiceAccount inside namespaces, and ensures a ServiceAccount named "default" exists in every active namespace.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on May 06, 2019 at 11:55 AM PST by Fix broken link in service-accounts-admin.md (#14178) (Page History)

Edit This Page

Authorization Overview

Learn more about Kubernetes authorization, including details about creating policies using the supported authorization modules.

- Determine Whether a Request is Allowed or Denied
- Review Your Request Attributes
- Determine the Request Verb
- Authorization Modules
- Using Flags for Your Authorization Module
- Privilege escalation via pod creation
- What's next

In Kubernetes, you must be authenticated (logged in) before your request can be authorized (granted permission to access). For information about authentication, see Accessing Control Overview.

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.

Determine Whether a Request is Allowed or Denied

Kubernetes authorizes API requests using the API server. It evaluates all of the request attributes against all policies and allows or denies the request. All parts of an API request must be allowed by some policy in order to proceed. This means that permissions are denied by default.

(Although Kubernetes uses the API server, access controls and policies that depend on specific fields of specific kinds of objects are handled by Admission Controllers.)

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. A deny returns an HTTP status code 403.

Review Your Request Attributes

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 `get`, `list`, `create`, `update`, `patch`, `watch`, `proxy`, `redirect`, `delete`, and `deletecollection` are used for resource requests. To determine the request verb for a resource API endpoint, see Determine the request verb.
- **HTTP request verb** - HTTP verbs `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.

Determine the Request Verb

To determine the request verb for a resource API endpoint, review the HTTP verb used and whether or not the request acts on an individual resource or a collection of resources:

HTTP verb	request verb
POST	create
GET, HEAD	get (for individual resources), list (for collections)
PUT	update
PATCH	patch
DELETE	delete (for individual resources), deletecollection (for collections)

Kubernetes sometimes checks authorization for additional permissions using specialized verbs. For example:

- PodSecurityPolicy checks for authorization of the `use` verb on `podsecuritypolicies` resources in the `policy` API group.
- RBAC checks for authorization of the `bind` verb on `roles` and `clusterroles` resources in the `rbac.authorization.k8s.io` API group.
- Authentication layer checks for authorization of the `impersonate` verb on `users`, `groups`, and `serviceaccounts` in the core API group, and the `userextras` in the `authentication.k8s.io` API group.

Authorization Modules

- **Node** - A special-purpose authorizer that grants permissions to kubelets based on the pods they are scheduled to run. To learn more about using the Node authorization mode, see Node Authorization.
- **ABAC** - Attribute-based access control (ABAC) defines an access control paradigm whereby access rights are granted to users through the use ofÂ policiesÂ which combine attributes together. TheÂ policiesÂ can use any type of attributes (user attributes, resource attributes, object, environment attributes, etc). To learn more about using the ABAC mode, see ABAC Mode.
- **RBAC** - Role-based access controlÂ (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. To learn more about using the RBAC mode, see RBAC Mode
    - When specified RBAC (Role-Based Access Control) uses the `rbac.authorization.k8s.io` API group to drive authorization decisions, allowing admins to dynamically configure permission policies through the Kubernetes API.
    - To enable RBAC, start the apiserver with `--authorization-mode=RBAC`.
- **Webhook** - AÂ WebHookÂ is an HTTP callback: an HTTP POST that occurs when something happens; a simple event-notification via HTTP POST. A web application implementingÂ WebHooksÂ will POST a message to a URL when certain things happen. To learn more about using the Webhook mode, see Webhook Mode.

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
```

```
yes
```

```
kubectl auth can-i create deployments --namespace prod
```

```
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
```

```
no
```

`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 just 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.

```
kubectl create -f - -o yaml << EOF
```

```
apiVersion: authorization.k8s.io/v1
kind: SelfSubjectAccessReview
spec:
  resourceAttributes:
    group: apps
    resource: deployments
    verb: create
    namespace: dev
EOF

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
```

Using Flags for Your Authorization Module

You must include a flag in your policy to indicate which authorization module your policies include:

The following flags can be used:

- `--authorization-mode=ABAC` Attribute-Based Access Control (ABAC) mode allows you to configure policies using local files.
- `--authorization-mode=RBAC` Role-based access controlÂ (RBAC) mode allows you to create and store policies using the Kubernetes API.
- `--authorization-mode=Webhook` WebHook is an HTTP callback mode that allows you to manage authorization using a remote REST endpoint.
- `--authorization-mode=Node` Node authorization is a special-purpose authorization mode that specifically authorizes API requests made by kubelets.
- `--authorization-mode=AlwaysDeny` This flag blocks all requests. Use this flag only for testing.
- `--authorization-mode=AlwaysAllow` This flag allows all requests. Use this flag only if you do not require authorization for your API requests.

You can choose more than one authorization module. Modules are checked in order so an earlier module has higher priority to allow or deny a request.

Privilege escalation via pod creation

Users who have the ability to create pods in a namespace can potentially escalate their privileges within that namespace. They can create pods that access their privileges within that namespace. They can create pods that access secrets the user cannot themselves read, or that run under a service account with different/greater permissions.

> **Caution:** System administrators, use care when granting access to pod creation. A user granted permission to create pods (or controllers that create pods) in the namespace can: read all secrets in the namespace; read all config maps in the namespace; and impersonate any service account in the namespace and take any action the account could take. This applies regardless of authorization mode.

What's next

- To learn more about Authentication, see **Authentication** in [Controlling Access to the Kubernetes API](#).
- To learn more about Admission Control, see [Using Admission Controllers](#).

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on [Stack Overflow](#). Open an issue in the GitHub repo if you want to [report a problem](#) or [suggest an improvement](#).

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
- Authorization Algorithm
- Kubectl
- Examples
- A quick note on service accounts

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, just 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: `extensions`
    - 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`

        - 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/`.
    - `readonly`, type boolean, when true, means that the Resource-matching policy only applies to get, list, and watch operations, Non-resource-matching policy only applies to get operation.

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 api-server 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", "spec": {"user": "alice", "namespace": "*", "resource": "*", "apiGroup": "*"}}
   ```

2. The Kubelet can read any pods:

   ```
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "kubelet", "namespace": "*", "resource": "pods", "readonly": true}}
   ```

3. The Kubelet can read and write events:

   ```
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "kubelet", "namespace": "*", "resource": "events"}}
   ```

4. Bob can just read pods in namespace "projectCaribou":

   ```
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "bob", "namespace": "projectCaribou", "resource": "pods", "readonly": true}}
   ```

5. Anyone can make read-only requests to all non-resource paths:

   ```
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"group": "system:authenticated", "readonly": true, "nonResourcePath": "*"}}
   {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"group": "system:unauthenticated", "readonly": true, "nonResourcePath": "*"}}
   ```

Complete file example

A quick note on service accounts

Every service account has a corresponding ABAC username, and that service account's user name 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","spec":{"user":"system:serviceaccount:kube-system:default","namespace":"*","resource":"*","apiGroup":"*"}}
```

The apiserver will need to be restarted to pickup the new policy lines.

> **Note:**
>
> **FEATURE STATE:** `Kubernetes 1.6` deprecated
> This feature is *deprecated*. For more information on this state, see the Kubernetes Deprecation Policy.
>
> The ABAC Authorization feature has been considered deprecated from the Kubernetes 1.6 release.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on April 10, 2019 at 11:04 AM PST by Update abac.md (#13067) (Page History)

Edit This Page

Using Node Authorization

Node authorization is a special-purpose authorization mode that specifically authorizes API requests made by kubelets.

- Overview
- Migration considerations

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

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 certificationsigningrequests 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, as well as cloud provider overrides, see the kubelet options reference and the cloud provider details.

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.

Upgrades from previous versions using RBAC

Upgraded pre-1.7 clusters using RBAC will continue functioning as-is because the `system:nodes` group binding will already exist.

If a cluster admin wishes to start using the `Node` authorizer and `NodeRestriction` admission plugin to limit node access to the API, that can be done non-disruptively:

1. Enable the `Node` authorization mode (`--authorization-mode=Node,RBAC`) and the `NodeRestriction` admission plugin
2. Ensure all kubelets' credentials conform to the group/username requirements
3. Audit apiserver logs to ensure the `Node` authorizer is not rejecting requests from kubelets (no persistent `NODE DENY` messages logged)
4. Delete the `system:node` cluster role binding

RBAC Node Permissions

In 1.6, the `system:node` cluster role was automatically bound to the `system:nodes` group when using the RBAC Authorization mode.

In 1.7, the automatic binding of the `system:nodes` group to the `system:node` role is deprecated because the node authorizer accomplishes the same purpose with the benefit of additional restrictions on secret and configmap access. If the `Node` and `RBAC` authorization modes are both enabled, the automatic binding of the `system:nodes` group to the `system:node` role is not created in 1.7.

In 1.8, the binding will not be created at all.

When using RBAC, the `system:node` cluster role will continue to be created, for compatibility with deployment methods that bind other users or groups to that role.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

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.

- Configuration File Format
- Request Payloads

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 parameters.
      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 to use
      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:


```json
{
  "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:


```json
{
  "apiVersion": "authorization.k8s.io/v1beta1",
  "kind": "SubjectAccessReview",
  "status": {
    "allowed": true
  }
}
```


To disallow access, the remote service would return:


```json
{
  "apiVersion": "authorization.k8s.io/v1beta1",
  "kind": "SubjectAccessReview",
```

```json
  "status": {
    "allowed": false,
    "reason": "user does not have read access to the namespace"
  }
}
```

Access to non-resource paths are sent as:

```json
{
  "apiVersion": "authorization.k8s.io/v1beta1",
  "kind": "SubjectAccessReview",
  "spec": {
    "nonResourceAttributes": {
      "path": "/debug",
      "verb": "get"
    },
    "user": "jane",
    "group": [
      "group1",
      "group2"
    ]
  }
}
```

Non-resource paths include: `/api`, `/apis`, `/metrics`, `/resetMetrics`, `/logs`, `/debug`, `/healthz`, `/swagger-ui/`, `/swaggerapi/`, `/ui`, 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 documentation refer to the authorization.v1beta1 API objects and webhook.go.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Well-Known Labels, Annotations and Taints

Kubernetes reserves all labels and annotations in the kubernetes.io namespace.

This document serves both as a reference to the values, and as a coordination point for assigning values.

- kubernetes.io/arch
- kubernetes.io/os
- beta.kubernetes.io/arch (deprecated)
- beta.kubernetes.io/os (deprecated)
- kubernetes.io/hostname
- beta.kubernetes.io/instance-type
- failure-domain.beta.kubernetes.io/region
- failure-domain.beta.kubernetes.io/zone

kubernetes.io/arch

Example: `kubernetes.io/arch=amd64`

Used on: Node

Kubelet populates this with `runtime.GOARCH` as defined by Go. This can be handy if you are mixing arm and x86 nodes, for example.

kubernetes.io/os

Example: `kubernetes.io/os=linux`

Used on: Node

Kubelet populates this with `runtime.GOOS` as defined by Go. This can be handy if you are mixing operating systems in your cluster (e.g., mixing Linux and Windows nodes).

beta.kubernetes.io/arch (deprecated)

This label has been deprecated. Please use `kubernetes.io/arch` instead.

beta.kubernetes.io/os (deprecated)

This label has been deprecated. Please use `kubernetes.io/os` instead.

kubernetes.io/hostname

Example: `kubernetes.io/hostname=ip-172-20-114-199.ec2.internal`

Used on: Node

Kubelet populates this with the hostname. Note that the hostname can be changed from the "actual" hostname by passing the `--hostname-override` flag to kubelet.

beta.kubernetes.io/instance-type

Example: `beta.kubernetes.io/instance-type=m3.medium`

Used on: Node

Kubelet populates this with the instance type as defined by the `cloudprovider`. It will not be set if not using a cloudprovider. This can be 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, and you should aim to schedule based on properties rather than on instance types (e.g. require a GPU, instead of requiring a `g2.2xlarge`)

failure-domain.beta.kubernetes.io/region

See [failure-domain.beta.kubernetes.io/zone](#).

failure-domain.beta.kubernetes.io/zone

Example:

`failure-domain.beta.kubernetes.io/region=us-east-1`

`failure-domain.beta.kubernetes.io/zone=us-east-1c`

Used on: Node, PersistentVolume

On the Node: Kubelet populates this with the zone information as defined by the `cloudprovider`. It will not be set if not using a `cloudprovider`, but you should consider setting it on the nodes if it makes sense in your topology.

On the PersistentVolume: The `PersistentVolumeLabel` admission controller will automatically add zone labels to PersistentVolumes, on GCE and AWS.

Kubernetes will automatically spread the pods in a replication controller or service across nodes in a single-zone cluster (to reduce the impact of failures). With multiple-zone clusters, this spreading behaviour is extended across zones (to reduce the impact of zone failures). This is achieved via SelectorSpreadPriority.

This is a best-effort placement, and so if the zones in your cluster are heterogeneous (e.g. different numbers of nodes, different types of nodes, or different pod resource requirements), this might prevent equal spreading of your pods across zones. If desired, you can use homogenous zones (same number and types of nodes) to reduce the probability of unequal spreading.

The scheduler (via the VolumeZonePredicate predicate) will also ensure that pods that claim a given volume are only placed into the same zone as that volume, as volumes cannot be attached across zones.

The actual values of zone and region don't matter, and nor is the meaning of the hierarchy rigidly defined. The expectation is that failures of nodes in different zones should be uncorrelated unless the entire region has failed. For example, zones should typically avoid sharing a single network switch. The exact mapping depends on your particular infrastructure - a three-rack installation will choose a very different setup to a multi-datacenter configuration.

If `PersistentVolumeLabel` does not support automatic labeling of your PersistentVolumes, you should consider adding the labels manually (or adding support to `PersistentVolumeLabel`), if you want the scheduler to prevent 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.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on March 25, 2019 at 5:06 PM PST by Official 1.14 Release Docs (#13174) (Page History)

Edit This Page

v1.15

Kubernetes API v1.15

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 20, 2019 at 9:48 PM PST by generate and update api reference to 1.15 (#15042) (Page History)

Edit This Page

kubeadm reset

This command reverts any changes made by `kubeadm init` or `kubeadm join`.

- What's next

Run this to revert any changes made to this host by â€˜kubeadm init' or â€˜kubeadm join'

Synopsis

Run this to revert any changes made to this host by â€˜kubeadm init' or â€˜kubeadm join'

The "reset" command executes the following phases:

```
preflight               Run reset pre-flight checks
update-cluster-status   Remove this node from the ClusterStatus
object.
remove-etcd-member      Remove a local etcd member.
cleanup-node            Run cleanup node.
```

```
kubeadm reset [flags]
```

Options

```
      --cert-dir string                   The path to the
directory where the certificates are stored. If specified, clean
this directory. (default "/etc/kubernetes/pki")
      --cri-socket string                 Path to the CRI socket
to connect. If empty kubeadm will try to auto-detect this value;
use this option only if you have more than one CRI installed or
if you have non-standard CRI socket.
  -f, --force                             Reset the node without
prompting for confirmation.
  -h, --help                              help for reset
      --ignore-preflight-errors strings   A list of checks whose
errors will be shown as warnings. Example:
'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all
checks.
      --kubeconfig string                 The kubeconfig file to
use when talking to the cluster. If the flag is not set, a set
of standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --skip-phases strings               List of phases to be
skipped
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Reset workflow

`kubeadm reset` is responsible for cleaning up a node local file system from files that were created using the `kubeadm init` or `kubeadm join` commands. For control-plane nodes `reset` also removes the local stacked etcd member of this node from the etcd cluster and also removes this node's information from the kubeadm `ClusterStatus` object. `ClusterStatus` is a kubeadm managed Kubernetes API object that holds a list of kube-apiserver endpoints.

`kubeadm reset phase` can be used to execute the separate phases of the above workflow. To skip a list of phases you can use the `--skip-phases` flag, which works in a similar way to the `kubeadm join` and `kubeadm init` phase runners.

External etcd clean up

`kubeadm reset` will not delete any etcd data if external etcd is used. This means that if you run `kubeadm init` again using the same etcd endpoints, you will see state from previous clusters.

To wipe etcd data it is recommended you use a client like etcdctl, such as:

```
etcdctl del "" --prefix
```

See the etcd documentation for more information.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to bootstrap a Kubernetes worker node and join it to the cluster

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 11, 2019 at 4:56 AM PST by kubeadm: update the reference documentation for 1.15 (#14596) (Page History)

Edit This Page

Overview of kubeadm

-
    - How to install
    - What's next

Kubeadm is a tool built to provide `kubeadm init` and `kubeadm join` as best-practice "fast paths" for creating Kubernetes clusters.

kubeadm performs the actions necessary to get a minimum viable cluster up and running. By design, it cares only about bootstrapping, not about provisioning machines. Likewise, installing various nice-to-have addons, like the Kubernetes Dashboard, monitoring solutions, and cloud-specific addons, is not in scope.

Instead, we expect higher-level and more tailored tooling to be built on top of kubeadm, and ideally, using kubeadm as the basis of all deployments will make it easier to create conformant clusters.

How to install

To install kubeadm, see the installation guide.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to bootstrap a Kubernetes worker node and join it to the cluster
- kubeadm upgrade to upgrade a Kubernetes cluster to a newer version
- kubeadm config if you initialized your cluster using kubeadm v1.7.x or lower, to configure your cluster for `kubeadm upgrade`
- kubeadm token to manage tokens for `kubeadm join`
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`
- kubeadm version to print the kubeadm version
- kubeadm alpha to preview a set of features made available for gathering feedback from the community

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on July 07, 2019 at 4:56 PM PST by add how to install kubeadm (#15174) (Page History)

Edit This Page

kubeadm init

This command initializes a Kubernetes control-plane node.

- What's next

Run this command in order to set up the Kubernetes control plane

Synopsis

Run this command in order to set up the Kubernetes control plane

The "init" command executes the following phases:

```
preflight                    Run pre-flight checks
kubelet-start                Write kubelet settings and (re)start
the kubelet
certs                        Certificate generation
  /etcd-ca                     Generate the self-signed CA to
provision identities for etcd
  /apiserver-etcd-client       Generate the certificate the
apiserver uses to access etcd
  /etcd-healthcheck-client     Generate the certificate for
liveness probes to healtcheck etcd
  /etcd-server                 Generate the certificate for
serving etcd
  /etcd-peer                   Generate the certificate for etcd
nodes to communicate with each other
  /ca                          Generate the self-signed Kubernetes
CA to provision identities for other Kubernetes components
  /apiserver                   Generate the certificate for
serving the Kubernetes API
  /apiserver-kubelet-client    Generate the certificate for the
API server to connect to kubelet
  /front-proxy-ca              Generate the self-signed CA to
provision identities for front proxy
  /front-proxy-client          Generate the certificate for the
front proxy client
  /sa                          Generate a private key for signing
service account tokens along with its public key
kubeconfig                   Generate all kubeconfig files
necessary to establish the control plane and the admin
kubeconfig file
  /admin                       Generate a kubeconfig file for the
admin to use and for kubeadm itself
  /kubelet                     Generate a kubeconfig file for the
kubelet to use *only* for cluster bootstrapping purposes
  /controller-manager          Generate a kubeconfig file for the
controller manager to use
  /scheduler                   Generate a kubeconfig file for the
scheduler to use
control-plane                Generate all static Pod manifest
files necessary to establish the control plane
  /apiserver                   Generates the kube-apiserver static
Pod manifest
  /controller-manager          Generates the kube-controller-
manager static Pod manifest
  /scheduler                   Generates the kube-scheduler static
Pod manifest
etcd                         Generate static Pod manifest file for
local etcd
```

```
  /local                     Generate the static Pod manifest
file for a local, single-node local etcd instance
upload-config              Upload the kubeadm and kubelet
configuration to a ConfigMap
  /kubeadm                   Upload the kubeadm
ClusterConfiguration to a ConfigMap
  /kubelet                   Upload the kubelet component config
to a ConfigMap
upload-certs               Upload certificates to kubeadm-certs
mark-control-plane         Mark a node as a control-plane
bootstrap-token            Generates bootstrap tokens used to
join a node to a cluster
addon                      Install required addons for passing
Conformance tests
  /coredns                   Install the CoreDNS addon to a
Kubernetes cluster
  /kube-proxy                Install the kube-proxy addon to a
Kubernetes cluster
```

```
kubeadm init [flags]
```

Options

```
      --apiserver-advertise-address string   The IP address the
API Server will advertise it's listening on. If not set the
default network interface will be used.
      --apiserver-bind-port int32            Port for the API
Server to bind to. (default 6443)
      --apiserver-cert-extra-sans strings    Optional extra
Subject Alternative Names (SANs) to use for the API Server
serving certificate. Can be both IP addresses and DNS names.
      --cert-dir string                      The path where to
save and store the certificates. (default "/etc/kubernetes/pki")
      --certificate-key string               Key used to encrypt
the control-plane certificates in the kubeadm-certs Secret.
      --config string                        Path to a kubeadm
configuration file.
      --cri-socket string                    Path to the CRI
socket to connect. If empty kubeadm will try to auto-detect this
value; use this option only if you have more than one CRI
installed or if you have non-standard CRI socket.
      --dry-run                              Don't apply any
changes; just output what would be done.
      --feature-gates string                 A set of key=value
pairs that describe feature gates for various features. No
feature gates are available in this release.
  -h, --help                                 help for init
      --ignore-preflight-errors strings      A list of checks
whose errors will be shown as warnings. Example:
'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all
checks.
      --image-repository string              Choose a container
registry to pull control plane images from (default "k8s.gcr.io")
```

```
      --kubernetes-version string            Choose a specific
Kubernetes version for the control plane. (default "stable-1")
      --node-name string                     Specify the node
name.
      --pod-network-cidr string              Specify range of IP
addresses for the pod network. If set, the control plane will
automatically allocate CIDRs for every node.
      --service-cidr string                  Use alternative
range of IP address for service VIPs. (default "10.96.0.0/12")
      --service-dns-domain string            Use alternative
domain for services, e.g. "myorg.internal". (default
"cluster.local")
      --skip-certificate-key-print           Don't print the key
used to encrypt the control-plane certificates.
      --skip-phases strings                  List of phases to
be skipped
      --skip-token-print                     Skip printing of
the default bootstrap token generated by 'kubeadm init'.
      --token string                         The token to use
for establishing bidirectional trust between nodes and control-
plane nodes. The format is [a-z0-9]{6}\.[a-z0-9]{16} - e.g.
abcdef.0123456789abcdef
      --token-ttl duration                   The duration before
the token is automatically deleted (e.g. 1s, 2m, 3h). If set to
'0', the token will never expire (default 24h0m0s)
      --upload-certs                         Upload control-
plane certificates to the kubeadm-certs Secret.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Init workflow

`kubeadm init` bootstraps a Kubernetes control-plane node by executing the following steps:

1. Runs a series of pre-flight checks to validate the system state before making changes. Some checks only trigger warnings, others are considered errors and will exit kubeadm until the problem is corrected or the user specifies `--ignore-preflight-errors=<list-of-errors>`.

2. Generates a self-signed CA (or using an existing one if provided) to set up identities for each component in the cluster. If the user has provided their own CA cert and/or key by dropping it in the cert directory configured via `--cert-dir` (`/etc/kubernetes/pki` by default) this step is skipped as described in the Using custom certificates document. The APIServer certs will have additional SAN entries for any `--apiserver-cert-extra-sans` arguments, lowercased if necessary.

3. Writes kubeconfig files in `/etc/kubernetes/` for the kubelet, the controller-manager and the scheduler to use to connect to the API server, each with its own identity, as well as an additional kubeconfig file for administration named `admin.conf`.

4. Generates static Pod manifests for the API server, controller manager and scheduler. In case an external etcd is not provided, an additional static Pod manifest is generated for etcd.

Static Pod manifests are written to `/etc/kubernetes/manifests`; the kubelet watches this directory for Pods to create on startup.

Once control plane Pods are up and running, the `kubeadm init` sequence can continue.

1. Apply labels and taints to the control-plane node so that no additional workloads will run there.

2. Generates the token that additional nodes can use to register themselves with a control-plane in the future. Optionally, the user can provide a token via `--token`, as described in the kubeadm token docs.

3. Makes all the necessary configurations for allowing node joining with the Bootstrap Tokens and TLS Bootstrap mechanism:

    - Write a ConfigMap for making available all the information required for joining, and set up related RBAC access rules.

    - Let Bootstrap Tokens access the CSR signing API.

    - Configure auto-approval for new CSR requests.

See kubeadm join for additional info.

1. Installs a DNS server (CoreDNS) and the kube-proxy addon components via the API server. In Kubernetes version 1.11 and later CoreDNS is the default DNS server. To install kube-dns instead of CoreDNS, the DNS addon has to be configured in the kubeadm `ClusterConfiguration`. For more information about the configuration see the section `Using kubeadm init with a configuration file` below. Please note that although the DNS server is deployed, it will not be scheduled until CNI is installed.

Using init phases with kubeadm

Kubeadm allows you create a control-plane node in phases. In 1.13 the `kubeadm init phase` command has graduated to GA from it's previous alpha state under `kubeadm alpha phase`.

To view the ordered list of phases and sub-phases you can call `kubeadm init --help`. The list will be located at the top of the help screen and each phase will have a description next to it. Note that by calling `kubeadm init` all of the phases and sub-phases will be executed in this exact order.

Some phases have unique flags, so if you want to have a look at the list of available options add `--help`, for example:

```
sudo kubeadm init phase control-plane controller-manager --help
```

You can also use `--help` to see the list of sub-phases for a certain parent phase:

```
sudo kubeadm init phase control-plane --help
```

`kubeadm init` also exposes a flag called `--skip-phases` that can be used to skip certain phases. The flag accepts a list of phase names and the names can be taken from the above ordered list.

An example:

```
sudo kubeadm init phase control-plane all --config=configfile.yaml
sudo kubeadm init phase etcd local --config=configfile.yaml
# you can now modify the control plane and etcd manifest files
sudo kubeadm init --skip-phases=control-plane,etcd --config=configfile.yaml
```

What this example would do is write the manifest files for the control plane and etcd in `/etc/kubernetes/manifests` based on the configuration in `configfile.yaml`. This allows you to modify the files and then skip these phases using `--skip-phases`. By calling the last command you will create a control plane node with the custom manifest files.

Using kubeadm init with a configuration file

> **Caution:** The config file is still considered beta and may change in future versions.

It's possible to configure `kubeadm init` with a configuration file instead of command line flags, and some more advanced features may only be available as configuration file options. This file is passed in the `--config` option.

In Kubernetes 1.11 and later, the default configuration can be printed out using the kubeadm config print command.

It is **recommended** that you migrate your old `v1beta1` configuration to `v1beta2` using the kubeadm config migrate command.

For more details on each field in the `v1beta2` configuration you can navigate to our API reference pages.

Adding kube-proxy parameters

For information about kube-proxy parameters in the kubeadm configuration see: - kube-proxy

For information about enabling IPVS mode with kubeadm see: - IPVS

Passing custom flags to control plane components

For information about passing flags to control plane components see: - control-plane-flags

Using custom images

By default, kubeadm pulls images from `k8s.gcr.io`, unless the requested Kubernetes version is a CI version. In this case, `gcr.io/kubernetes-ci-images` is used.

You can override this behavior by using kubeadm with a configuration file. Allowed customization are:

- To provide an alternative `imageRepository` to be used instead of `k8s.gcr.io`.
- To set `useHyperKubeImage` to `true` to use the HyperKube image.
- To provide a specific `imageRepository` and `imageTag` for etcd or DNS add-on.

Please note that the configuration field `kubernetesVersion` or the command line flag `--kubernetes-version` affect the version of the images.

Uploading control-plane certificates to the cluster

By adding the flag `--upload-certs` to `kubeadm init` you can temporary upload the control-plane certificates to a Secret in the cluster. Please note that this Secret will expire automatically after 2 hours. The certificates are encrypted using a 32byte key that can be specified using `--certificate-key`. The same key can be used to download the certificates when additional control-plane nodes are joining, by passing `--control-plane` and `--certificate-key` to `kubeadm join`.

The following phase command can be used to re-upload the certificates after expiration:

```
kubeadm init phase upload-certs --upload-certs --certificate-key=SOME_VALUE
```

If the flag `--certificate-key` is not passed to `kubeadm init` and `kubeadm init phase upload-certs` a new key will be generated automatically.

The following command can be used to generate a new key on demand:

```
kubeadm alpha certs certificate-key
```

Using custom certificates

By default, kubeadm generates all the certificates needed for a cluster to run. You can override this behavior by providing your own certificates.

To do so, you must place them in whatever directory is specified by the `--cert-dir` flag or `CertificatesDir` configuration file key. By default this is `/etc/kubernetes/pki`.

If a given certificate and private key pair exists, kubeadm skips the generation step and existing files are used for the prescribed use case. This means you can, for example, copy an existing CA into `/etc/kubernetes/pki/ca.crt` and `/etc/kubernetes/pki/ca.key`, and kubeadm will use this CA for signing the rest of the certs.

External CA mode

It is also possible to provide just the `ca.crt` file and not the `ca.key` file (this is only available for the root CA file, not other cert pairs). If all other certificates and kubeconfig files are in place, kubeadm recognizes this condition and activates the "External CA" mode. kubeadm will proceed without the CA key on disk.

Instead, run the controller-manager standalone with `--controllers=csrsigner` and point to the CA certificate and key.

Managing the kubeadm drop-in file for the kubelet

The kubeadm package ships with configuration for how the kubelet should be run. Note that the `kubeadm` CLI command never touches this drop-in file. This drop-in file belongs to the kubeadm deb/rpm package.

This is what it looks like:

```
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/
kubernetes/bootstrap-kubelet.conf
--kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/
config.yaml"
# This is a file that "kubeadm init" and "kubeadm join"
generates at runtime, populating
the KUBELET_KUBEADM_ARGS variable dynamically
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the
kubelet args as a last resort. Preferably,
#the user should use the .NodeRegistration.KubeletExtraArgs
object in the configuration files instead.
# KUBELET_EXTRA_ARGS should be sourced from this file.
EnvironmentFile=-/etc/default/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS
$KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS $KUBELET_EXTRA_ARGS
```

Here's a breakdown of what/why:

- `--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf` path to a kubeconfig file that is used to get client certificates for kubelet during node join. On success, a kubeconfig file is written to the path specified by `--kubeconfig`.
- `--kubeconfig=/etc/kubernetes/kubelet.conf` points to the kubeconfig file that tells the kubelet where the API server is. This file also has the kubelet's credentials.
- `--pod-manifest-path=/etc/kubernetes/manifests` specifies from where to read static Pod manifests used for starting the control plane.
- `--allow-privileged=true` allows this kubelet to run privileged Pods.
- `--network-plugin=cni` uses CNI networking.
- `--cni-conf-dir=/etc/cni/net.d` specifies where to look for the CNI spec file(s).
- `--cni-bin-dir=/opt/cni/bin` specifies where to look for the actual CNI binaries.
- `--cluster-dns=10.96.0.10` use this cluster-internal DNS server for `nameserver` entries in Pods' `/etc/resolv.conf`.
- `--cluster-domain=cluster.local` uses this cluster-internal DNS domain for `search` entries in Pods' `/etc/resolv.conf`.
- `--client-ca-file=/etc/kubernetes/pki/ca.crt` authenticates requests to the Kubelet API using this CA certificate.
- `--authorization-mode=Webhook` authorizes requests to the Kubelet API by `POST`-ing a `SubjectAccessReview` to the API server.
- `--rotate-certificates` auto rotate the kubelet client certificates by requesting new certificates from the `kube-apiserver` when the certificate expiration approaches.
- `--cert-dir`the directory where the TLS certs are located.

Use kubeadm with CRI runtimes

Since v1.6.0, Kubernetes has enabled the use of CRI, Container Runtime Interface, by default. The container runtime used by default is Docker, which is enabled through the built-in `dockershim` CRI implementation inside of the `kubelet`.

Other CRI-based runtimes include:

- cri-containerd
- cri-o
- frakti
- rkt

Refer to the CRI installation instructions for more information.

After you have successfully installed `kubeadm` and `kubelet`, execute these two additional steps:

1. Install the runtime shim on every node, following the installation document in the runtime shim project listing above.

2. Configure kubelet to use the remote CRI runtime. Please remember to change `RUNTIME_ENDPOINT` to your own value like `/var/run/{your_runtime}.sock`:

```
cat > /etc/systemd/system/kubelet.service.d/20-cri.conf <<EOF
[Service]
Environment="KUBELET_EXTRA_ARGS=--container-runtime=remote --container-runtime-endpoint=$RUNTIME_ENDPOINT"
EOF
systemctl daemon-reload
```

Now `kubelet` is ready to use the specified CRI runtime, and you can continue with the `kubeadm init` and `kubeadm join` workflow to deploy Kubernetes cluster.

You may also want to set `--cri-socket` to `kubeadm init` and `kubeadm reset` when using an external CRI implementation.

Setting the node name

By default, `kubeadm` assigns a node name based on a machine's host address. You can override this setting with the `--node-name`flag. The flag passes the appropriate `--hostname-override` to the kubelet.

Be aware that overriding the hostname can interfere with cloud providers.

Running kubeadm without an internet connection

For running kubeadm without an internet connection you have to pre-pull the required control-plane images.

In Kubernetes 1.11 and later, you can list and pull the images using the `kubeadm config images` sub-command:

```
kubeadm config images list
kubeadm config images pull
```

In Kubernetes 1.12 and later, the `k8s.gcr.io/kube-*`, `k8s.gcr.io/etcd` and `k8s.gcr.io/pause` images don't require an `-${ARCH}` suffix.

Automating kubeadm

Rather than copying the token you obtained from `kubeadm init` to each node, as in the basic kubeadm tutorial, you can parallelize the token distribution for easier automation. To implement this automation, you must know the IP address that the control-plane node will have after it is started.

1. Generate a token. This token must have the form `<6 character string>.<16 character string>`. More formally, it must match the regex: `[a-z0-9]{6}\.[a-z0-9]{16}`.

   kubeadm can generate a token for you:

   ```
   kubeadm token generate
   ```

2. Start both the control-plane node and the worker nodes concurrently with this token. As they come up they should find each other and form the cluster. The same `--token` argument can be used on both `kubeadm init` and `kubeadm join`.

3. Similar can be done for `--certificate-key` when joining additional control-plane nodes. The key can be generated using:

   ```
   kubeadm alpha certs certificate-key
   ```

Once the cluster is up, you can grab the admin credentials from the control-plane node at `/etc/kubernetes/admin.conf` and use that to talk to the cluster.

Note that this style of bootstrap has some relaxed security guarantees because it does not allow the root CA hash to be validated with `--discovery-token-ca-cert-hash` (since it's not generated when the nodes are provisioned). For details, see the kubeadm join.

What's next

- kubeadm init phase to understand more about `kubeadm init` phases
- kubeadm join to bootstrap a Kubernetes worker node and join it to the cluster
- kubeadm upgrade to upgrade a Kubernetes cluster to a newer version
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 13, 2019 at 9:50 PM PST by fix some typos in kubeadm init (#14864) (Page History)

Edit This Page

kubeadm join

This command initializes a Kubernetes worker node and joins it to the cluster.

- What's next

Run this on any machine you wish to join an existing cluster

Synopsis

When joining a kubeadm initialized cluster, we need to establish bidirectional trust. This is split into discovery (having the Node trust the Kubernetes Control Plane) and TLS bootstrap (having the Kubernetes Control Plane trust the Node).

There are 2 main schemes for discovery. The first is to use a shared token along with the IP address of the API server. The second is to provide a file - a subset of the standard kubeconfig file. This file can be a local file or downloaded via an HTTPS URL. The forms are kubeadm join -discovery-token abcdef.1234567890abcdef 1.2.3.4:6443, kubeadm join -discovery-file path/to/file.conf, or kubeadm join -discovery-file https://url/file.conf. Only one form can be used. If the discovery information is loaded from a URL, HTTPS must be used. Also, in that case the host installed CA bundle is used to verify the connection.

If you use a shared token for discovery, you should also pass the -discovery-token-ca-cert-hash flag to validate the public key of the root certificate authority (CA) presented by the Kubernetes Control Plane. The value of this flag is specified as ":", where the supported hash type is "sha256". The hash is calculated over the bytes of the Subject Public Key Info (SPKI) object (as in RFC7469). This value is available in the output of "kubeadm init" or can be calculated using standard tools. The -discovery-token-ca-cert-hash flag may be repeated multiple times to allow more than one public key.

If you cannot know the CA public key hash ahead of time, you can pass the -discovery-token-unsafe-skip-ca-verification flag to disable this verification. This weakens the kubeadm security model since other nodes can potentially impersonate the Kubernetes Control Plane.

The TLS bootstrap mechanism is also driven via a shared token. This is used to temporarily authenticate with the Kubernetes Control Plane to submit a certificate signing request (CSR) for a locally created key pair. By default, kubeadm will set up the Kubernetes Control Plane to automatically approve these signing requests. This token is passed in with the -tls-bootstrap-token abcdef.1234567890abcdef flag.

Often times the same token is used for both parts. In this case, the -token flag can be used instead of specifying each token individually.

The "join [api-server-endpoint]" command executes the following phases:

```
preflight              Run join pre-flight checks
control-plane-prepare  Prepare the machine for serving a control
plane
  /download-certs        [EXPERIMENTAL] Download certificates
shared among control-plane nodes from the kubeadm-certs Secret
  /certs                 Generate the certificates for the new
control plane components
  /kubeconfig            Generate the kubeconfig for the new
control plane components
  /control-plane         Generate the manifests for the new
control plane components
kubelet-start          Write kubelet settings, certificates and
(re)start the kubelet
control-plane-join     Join a machine as a control plane instance
  /etcd                  Add a new local etcd member
  /update-status         Register the new control-plane node
into the ClusterStatus maintained in the kubeadm-config ConfigMap
  /mark-control-plane    Mark a node as a control-plane
```

```
kubeadm join [api-server-endpoint] [flags]
```

Options

```
      --apiserver-advertise-address string            If the
node should host a new control plane instance, the IP address
the API Server will advertise it's listening on. If not set the
default network interface will be used.
      --apiserver-bind-port int32                     If the
node should host a new control plane instance, the port for the
API Server to bind to. (default 6443)
      --certificate-key string                        Use this
key to decrypt the certificate secrets uploaded by init.
      --config string                                 Path to
kubeadm config file.
      --control-plane                                 Create a
new control plane instance on this node
      --cri-socket string                             Path to
the CRI socket to connect. If empty kubeadm will try to auto-
detect this value; use this option only if you have more than
one CRI installed or if you have non-standard CRI socket.
      --discovery-file string                         For file-
based discovery, a file or URL from which to load cluster
information.
      --discovery-token string                        For token-
based discovery, the token used to validate cluster information
fetched from the API server.
      --discovery-token-ca-cert-hash strings          For token-
based discovery, validate that the root CA public key matches
```

```
this hash (format: "<type>:<value>").
      --discovery-token-unsafe-skip-ca-verification   For token-based discovery, allow joining without --discovery-token-ca-cert-hash pinning.
      --experimental-control-plane                    Create a new control plane instance on this node
  -h, --help                                          help for join
      --ignore-preflight-errors strings               A list of checks whose errors will be shown as warnings. Example: 'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all checks.
      --node-name string                              Specify the node name.
      --skip-phases strings                           List of phases to be skipped
      --tls-bootstrap-token string                    Specify the token used to temporarily authenticate with the Kubernetes Control Plane while joining the node.
      --token string                                  Use this token for both discovery-token and tls-bootstrap-token when those values are not provided.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

The join workflow

`kubeadm join` bootstraps a Kubernetes worker node or a control-plane node and adds it to the cluster. This action consists of the following steps for worker nodes:

1. kubeadm downloads necessary cluster information from the API server. By default, it uses the bootstrap token and the CA key hash to verify the authenticity of that data. The root CA can also be discovered directly via a file or URL.

2. Once the cluster information is known, kubelet can start the TLS bootstrapping process.

The TLS bootstrap uses the shared token to temporarily authenticate with the Kubernetes API server to submit a certificate signing request (CSR); by default the control plane signs this CSR request automatically.

1. Finally, kubeadm configures the local kubelet to connect to the API server with the definitive identity assigned to the node.

For control-plane nodes additional steps are performed:

1. Downloading certificates shared among control-plane nodes from the cluster (if explicitly requested by the user).

2. Generating control-plane component manifests, certificates and kubeconfig.

3. Adding new local etcd member.

4. Adding this node to the ClusterStatus of the kubeadm cluster.

Using join phases with kubeadm

Kubeadm allows you join a node to the cluster in phases. The `kubeadm join phase` command was added in v1.14.0.

To view the ordered list of phases and sub-phases you can call `kubeadm join --help`. The list will be located at the top of the help screen and each phase will have a description next to it. Note that by calling `kubeadm join` all of the phases and sub-phases will be executed in this exact order.

Some phases have unique flags, so if you want to have a look at the list of available options add `--help`, for example:

```
kubeadm join phase kubelet-start --help
```

Similar to the kubeadm init phase command, `kubadm join phase` allows you to skip a list of phases using the `--skip-phases` flag.

For example:

```
sudo kubeadm join --skip-phases=preflight --config=config.yaml
```

Discovering what cluster CA to trust

The kubeadm discovery has several options, each with security tradeoffs. The right method for your environment depends on how you provision nodes and the security expectations you have about your network and node lifecycles.

Token-based discovery with CA pinning

This is the default mode in Kubernetes 1.8 and above. In this mode, kubeadm downloads the cluster configuration (including root CA) and validates it using the token as well as validating that the root CA public key matches the provided hash and that the API server certificate is valid under the root CA.

The CA key hash has the format `sha256:<hex_encoded_hash>`. By default, the hash value is returned in the `kubeadm join` command printed at the end of `kubeadm init` or in the output of `kubeadm token create --print-join-command`. It is in a standard format (see RFC7469) and can also be calculated by 3rd party tools or provisioning systems. For example, using the OpenSSL CLI:

```
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl
rsa -pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex
| sed 's/^.* //'
```

Example `kubeadm join` commands:

For worker nodes:

```
kubeadm join --discovery-token abcdef.1234567890abcdef --
discovery-token-ca-cert-hash sha256:1234..cdef 1.2.3.4:6443
```

For control-plane nodes:

```
kubeadm join --discovery-token abcdef.1234567890abcdef --discovery-token-ca-cert-hash sha256:1234..cdef --control-plane 1.2.3.4:6443
```

You can also call `join` for a control-plane node with `--certificate-key` to copy certificates to this node, if the `kubeadm init` command was called with `--upload-certs`.

Advantages:

- Allows bootstrapping nodes to securely discover a root of trust for the control-plane node even if other worker nodes or the network are compromised.
- Convenient to execute manually since all of the information required fits into a single `kubeadm join` command that is easy to copy and paste.

Disadvantages:

- The CA hash is not normally known until the control-plane node has been provisioned, which can make it more difficult to build automated provisioning tools that use kubeadm. By generating your CA in beforehand, you may workaround this limitation though.

Token-based discovery without CA pinning

This was the default in Kubernetes 1.7 and earlier, but comes with some important caveats. This mode relies only on the symmetric token to sign (HMAC-SHA256) the discovery information that establishes the root of trust for the control-plane. It's still possible in Kubernetes 1.8 and above using the `--discovery-token-unsafe-skip-ca-verification` flag, but you should consider using one of the other modes if possible.

Example `kubeadm join` command:

```
kubeadm join --token abcdef.1234567890abcdef --discovery-token-unsafe-skip-ca-verification 1.2.3.4:6443`
```

Advantages:

- Still protects against many network-level attacks.
- The token can be generated ahead of time and shared with the control-plane node and worker nodes, which can then bootstrap in parallel without coordination. This allows it to be used in many provisioning scenarios.

Disadvantages:

- If an attacker is able to steal a bootstrap token via some vulnerability, they can use that token (along with network-level access) to impersonate the control-plane node to other bootstrapping nodes. This may or may not be an appropriate tradeoff in your environment.

File or HTTPS-based discovery

This provides an out-of-band way to establish a root of trust between the control-plane node and bootstrapping nodes. Consider using this mode if you are building automated provisioning using kubeadm.

Example `kubeadm join` commands:

- `kubeadm join --discovery-file path/to/file.conf` (local file)
- `kubeadm join --discovery-file https://url/file.conf` (remote HTTPS URL)

Advantages:

- Allows bootstrapping nodes to securely discover a root of trust for the control-plane node even if the network or other worker nodes are compromised.

Disadvantages:

- Requires that you have some way to carry the discovery information from the control-plane node to the bootstrapping nodes. This might be possible, for example, via your cloud provider or provisioning tool. The information in this file is not secret, but HTTPS or equivalent is required to ensure its integrity.

Securing your installation even more

The defaults for kubeadm may not work for everyone. This section documents how to tighten up a kubeadm installation at the cost of some usability.

Turning off auto-approval of node client certificates

By default, there is a CSR auto-approver enabled that basically approves any client certificate request for a kubelet when a Bootstrap Token was used when authenticating. If you don't want the cluster to automatically approve kubelet client certs, you can turn it off by executing this command:

```
kubectl delete clusterrolebinding kubeadm:node-autoapprove-
bootstrap
```

After that, `kubeadm join` will block until the admin has manually approved the CSR in flight:

```
kubectl get csr
```

The output is similar to this:

```
NAME                                                       AGE
REQUESTOR                 CONDITION
node-csr-c69HXe7aYcqkS1bKmH4faEnHAWxn6i2bHZ2mD04jZyQ   18s
system:bootstrap:878f07   Pending
```

```
kubectl certificate approve node-csr-
c69HXe7aYcqkS1bKmH4faEnHAWxn6i2bHZ2mD04jZyQ
```

The output is similar to this:

```
certificatesigningrequest "node-csr-
c69HXe7aYcqkS1bKmH4faEnHAWxn6i2bHZ2mD04jZyQ" approved
```

```
kubectl get csr
```

The output is similar to this:

```
NAME                                                    AGE
REQUESTOR                 CONDITION
node-csr-c69HXe7aYcqkS1bKmH4faEnHAWxn6i2bHZ2mD04jZyQ   1m
system:bootstrap:878f07   Approved,Issued
```

Only after `kubectl certificate approve` has been run, `kubeadm join` can proceed.

Turning off public access to the cluster-info ConfigMap

In order to achieve the joining flow using the token as the only piece of validation information, a ConfigMap with some data needed for validation of the control-plane node's identity is exposed publicly by default. While there is no private data in this ConfigMap, some users might wish to turn it off regardless. Doing so will disable the ability to use the `--discovery-token` flag of the `kubeadm join` flow. Here are the steps to do so:

- Fetch the `cluster-info` file from the API Server:

```
kubectl -n kube-public get cm cluster-info -o yaml | grep "kubeco
nfig:" -A11 | grep "apiVersion" -A10 | sed "s/    //" | tee
cluster-info.yaml
```

The output is similar to this:

```
apiVersion: v1
kind: Config
clusters:
- cluster:
    certificate-authority-data: <ca-cert>
    server: https://<ip>:<port>
  name: ""
contexts: []
current-context: ""
preferences: {}
users: []
```

- Use the `cluster-info.yaml` file as an argument to `kubeadm join --discovery-file`.

- Turn off public access to the `cluster-info` ConfigMap:

```
kubectl -n kube-public delete rolebinding kubeadm:bootstrap-
signer-clusterinfo
```

These commands should be run after `kubeadm init` but before `kubeadm join`.

Using kubeadm join with a configuration file

> **Caution:** The config file is still considered alpha and may change in future versions.

It's possible to configure `kubeadm join` with a configuration file instead of command line flags, and some more advanced features may only be available as configuration file options. This file is passed using the `--config` flag and it must contain a `JoinConfiguration` structure.

To print the default values of `JoinConfiguration` run the following command:

```
kubeadm config print join-defaults
```

For details on individual fields in `JoinConfiguration` see the godoc.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm token to manage tokens for `kubeadm join`
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 10, 2019 at 7:42 PM PST by update kubectl "$" format (#13256) (Page History)

Edit This Page

kubeadm upgrade

`kubeadm upgrade` is a user-friendly command that wraps complex upgrading logic behind one command, with support for both planning an upgrade and actually performing it. `kubeadm upgrade` can also be used for downgrading cluster if necessary.

- kubeadm upgrade guidance
- kubeadm upgrade plan
- kubeadm upgrade apply
- kubeadm upgrade diff
- kubeadm upgrade node

- What's next

kubeadm upgrade guidance

Every upgrade process might be a bit different, so we've documented each minor upgrade process individually. For more version-specific upgrade guidance, see the following resources:

- 1.12 to 1.13 upgrades
- 1.13 to 1.14 upgrades
- 1.14 to 1.15 upgrades

For older versions, please refer to older documentation sets on the Kubernetes website.

In Kubernetes v1.11.0 and later, you can use `kubeadm upgrade diff` to see the changes that would be applied to static pod manifests.

To use kube-dns with upgrades in Kubernetes v1.13.0 and later please follow this guide.

In Kubernetes v1.15.0 and later, `kubeadm upgrade apply` and `kubeadm upgrade node` will also automatically renew the kubeadm managed certificates on this node, including those stored in kubeconfig files. To opt-out, it is possible to pass the flag `--certificate-renewal=false`. For more details about certificate renewal see the certificate management documentation.

kubeadm upgrade plan

Check which versions are available to upgrade to and validate whether your current cluster is upgradeable. To skip the internet check, pass in the optional [version] parameter

Synopsis

Check which versions are available to upgrade to and validate whether your current cluster is upgradeable. To skip the internet check, pass in the optional [version] parameter

```
kubeadm upgrade plan [version] [flags]
```

Options

```
      --allow-experimental-upgrades        Show unstable
versions of Kubernetes as an upgrade alternative and allow
upgrading to an alpha/beta/release candidate versions of
Kubernetes.
      --allow-release-candidate-upgrades   Show release
candidate versions of Kubernetes as an upgrade alternative and
allow upgrading to a release candidate versions of Kubernetes.
      --config string                      Path to a kubeadm
configuration file.
      --feature-gates string               A set of key=value
pairs that describe feature gates for various features. No
feature gates are available in this release.
  -h, --help                               help for plan
      --ignore-preflight-errors strings    A list of checks
whose errors will be shown as warnings. Example:
```

```
'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all
checks.
      --kubeconfig string                  The kubeconfig file
to use when talking to the cluster. If the flag is not set, a
set of standard locations can be searched for an existing
kubeconfig file. (default "/etc/kubernetes/admin.conf")
      --print-config                       Specifies whether the
configuration file that will be used in the upgrade should be
printed or not.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm upgrade apply

Upgrade your Kubernetes cluster to the specified version

Synopsis

Upgrade your Kubernetes cluster to the specified version

```
kubeadm upgrade apply [version]
```

Options

```
      --allow-experimental-upgrades        Show unstable
versions of Kubernetes as an upgrade alternative and allow
upgrading to an alpha/beta/release candidate versions of
Kubernetes.
      --allow-release-candidate-upgrades   Show release
candidate versions of Kubernetes as an upgrade alternative and
allow upgrading to a release candidate versions of Kubernetes.
      --certificate-renewal                Perform the renewal
of certificates used by component changed during upgrades.
(default true)
      --config string                      Path to a kubeadm
configuration file.
      --dry-run                            Do not change any
state, just output what actions would be performed.
      --etcd-upgrade                       Perform the upgrade
of etcd. (default true)
      --feature-gates string               A set of key=value
pairs that describe feature gates for various features. No
feature gates are available in this release.
  -f, --force                              Force upgrading
although some requirements might not be met. This also implies
non-interactive mode.
  -h, --help                               help for apply
      --ignore-preflight-errors strings    A list of checks
whose errors will be shown as warnings. Example:
```

```
'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all
checks.
      --image-pull-timeout duration        The maximum amount of
time to wait for the control plane pods to be downloaded.
(default 15m0s)
      --kubeconfig string                  The kubeconfig file
to use when talking to the cluster. If the flag is not set, a
set of standard locations can be searched for an existing
kubeconfig file. (default "/etc/kubernetes/admin.conf")
      --print-config                       Specifies whether the
configuration file that will be used in the upgrade should be
printed or not.
  -y, --yes                                Perform the upgrade
and do not prompt for confirmation (non-interactive mode).
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm upgrade diff

Show what differences would be applied to existing static pod manifests. See also: kubeadm upgrade apply -dry-run

Synopsis

Show what differences would be applied to existing static pod manifests. See also: kubeadm upgrade apply -dry-run

```
kubeadm upgrade diff [version] [flags]
```

Options

```
      --api-server-manifest string          path to API server
manifest (default "/etc/kubernetes/manifests/kube-
apiserver.yaml")
      --config string                       Path to a kubeadm
configuration file.
  -c, --context-lines int                   How many lines of
context in the diff (default 3)
      --controller-manager-manifest string   path to controller
manifest (default "/etc/kubernetes/manifests/kube-controller-
manager.yaml")
  -h, --help                                help for diff
      --scheduler-manifest string           path to scheduler
manifest (default "/etc/kubernetes/manifests/kube-
scheduler.yaml")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

kubeadm upgrade node

Upgrade commands for a node in the cluster

Synopsis

Upgrade commands for a node in the cluster

The "node" command executes the following phases:

```
control-plane   Upgrade the control plane instance deployed on this node, if any
kubelet-config  Upgrade the kubelet configuration for this node
```

```
kubeadm upgrade node [flags]
```

Options

```
      --dry-run                    Do not change any state, just output the actions that would be performed.
  -h, --help                       help for node
      --kubeconfig string          The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file. (default "/etc/kubernetes/admin.conf")
      --kubelet-version string     The *desired* version for the kubelet config after the upgrade. If not specified, the KubernetesVersion from the kubeadm-config ConfigMap will be used
      --skip-phases strings        List of phases to be skipped
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

What's next

- [kubeadm config](#) if you initialized your cluster using kubeadm v1.7.x or lower, to configure your cluster for `kubeadm upgrade`

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 20, 2019 at 5:02 AM PST by update the upgrade method of kubelet / kubectl of worker nodes (#14871) (Page History)

Edit This Page

kubeadm config

Beginning with v1.8.0, kubeadm uploads the configuration of your cluster to a ConfigMap called `kubeadm-config` in the `kube-system` namespace, and later reads the ConfigMap when upgrading. This enables correct configuration of system components, and provides a seamless user experience.

You can execute `kubeadm config view` to view the ConfigMap. If you initialized your cluster using kubeadm v1.7.x or lower, you must use `kubeadm config upload` to create the ConfigMap before you may use `kubeadm upgrade`.

In Kubernetes v1.11.0, some new commands were added. You can use `kubeadm config print-default` to print the default configuration and `kubeadm config migrate` to convert your old configuration files to a newer version. `kubeadm config images list` and `kubeadm config images pull` can be used to list and pull the images that kubeadm requires.

In Kubernetes v1.13.0 and later to list/pull kube-dns images instead of the CoreDNS image the `--config` method described here has to be used.

- kubeadm config view
- kubeadm config print init-defaults
- kubeadm config print join-defaults
- kubeadm config migrate
- kubeadm config images list
- kubeadm config images pull
- What's next

kubeadm config view

View the kubeadm configuration stored inside the cluster

Synopsis

Using this command, you can view the ConfigMap in the cluster where the configuration for kubeadm is located.

The configuration is located in the "kube-system" namespace in the "kubeadm-config" ConfigMap.

```
kubeadm config view [flags]
```

Options

```
  -h, --help   help for view
```

Options inherited from parent commands

```
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --rootfs string       [EXPERIMENTAL] The path to the
'real' host root filesystem.
```

kubeadm config print init-defaults

Print default init configuration, that can be used for â€˜kubeadm init'

Synopsis

This command prints objects such as the default init configuration that is used for â€˜kubeadm init'.

Note that sensitive values like the Bootstrap Token fields are replaced with placeholder values like {"abcdef.0123456789abcdef" "" "nil" [] []} in order to pass validation but not perform the real computation for creating a token.

```
kubeadm config print init-defaults [flags]
```

Options

```
      --component-configs strings   A comma-separated list for
component config API objects to print the default values for.
Available values: [KubeProxyConfiguration KubeletConfiguration].
If this flag is not set, no component configs will be printed.
  -h, --help                        help for init-defaults
```

Options inherited from parent commands

```
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --rootfs string       [EXPERIMENTAL] The path to the
'real' host root filesystem.
```

kubeadm config print join-defaults

Print default join configuration, that can be used for â€˜kubeadm join'

Synopsis

This command prints objects such as the default join configuration that is used for â€˜kubeadm join'.

Note that sensitive values like the Bootstrap Token fields are replaced with placeholder values like {"abcdef.0123456789abcdef" "" "nil" [] []} in order to pass validation but not perform the real computation for creating a token.

```
kubeadm config print join-defaults [flags]
```

Options

```
      --component-configs strings   A comma-separated list for
component config API objects to print the default values for.
Available values: [KubeProxyConfiguration KubeletConfiguration].
If this flag is not set, no component configs will be printed.
  -h, --help                        help for join-defaults
```

Options inherited from parent commands

```
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --rootfs string       [EXPERIMENTAL] The path to the
'real' host root filesystem.
```

kubeadm config migrate

Read an older version of the kubeadm configuration API types from a file, and output the similar config object for the newer version

Synopsis

This command lets you convert configuration objects of older versions to the latest supported version, locally in the CLI tool without ever touching anything in the cluster. In this version of kubeadm, the following API versions are supported:

- kubeadm.k8s.io/v1beta1
- kubeadm.k8s.io/v1beta2

Further, kubeadm can only write out config of version "kubeadm.k8s.io/v1beta2", but read both types. So regardless of what version you pass to the -old-config parameter here, the API object will be read, deserialized, defaulted, converted, validated, and re-serialized when written to stdout or -new-config if specified.

In other words, the output of this command is what kubeadm actually would read internally if you submitted this file to "kubeadm init"

```
kubeadm config migrate [flags]
```

Options

```
  -h, --help                help for migrate
      --new-config string   Path to the resulting equivalent
kubeadm config file using the new API version. Optional, if not
specified output will be sent to STDOUT.
      --old-config string   Path to the kubeadm config file that
is using an old API version and should be converted. This flag
is mandatory.
```

Options inherited from parent commands

```
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --rootfs string       [EXPERIMENTAL] The path to the
'real' host root filesystem.
```

kubeadm config images list

Print a list of images kubeadm will use. The configuration file is used in case any images or image repositories are customized

Synopsis

Print a list of images kubeadm will use. The configuration file is used in case any images or image repositories are customized

```
kubeadm config images list [flags]
```

Options

```
      --config string               Path to kubeadm config file.
      --feature-gates string        A set of key=value pairs
that describe feature gates for various features. No feature
gates are available in this release.
  -h, --help                        help for list
      --image-repository string     Choose a container registry
to pull control plane images from (default "k8s.gcr.io")
      --kubernetes-version string   Choose a specific Kubernetes
version for the control plane. (default "stable-1")
```

Options inherited from parent commands

```
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --rootfs string       [EXPERIMENTAL] The path to the
'real' host root filesystem.
```

kubeadm config images pull

Pull images used by kubeadm

Synopsis

Pull images used by kubeadm

```
kubeadm config images pull [flags]
```

Options

```
      --config string               Path to kubeadm config file.
      --cri-socket string           Path to the CRI socket to
connect. If empty kubeadm will try to auto-detect this value;
use this option only if you have more than one CRI installed or
if you have non-standard CRI socket.
      --feature-gates string        A set of key=value pairs
that describe feature gates for various features. No feature
gates are available in this release.
  -h, --help                        help for pull
      --image-repository string     Choose a container registry
to pull control plane images from (default "k8s.gcr.io")
      --kubernetes-version string   Choose a specific Kubernetes
version for the control plane. (default "stable-1")
```

Options inherited from parent commands

```
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --rootfs string       [EXPERIMENTAL] The path to the
'real' host root filesystem.
```

What's next

- kubeadm upgrade to upgrade a Kubernetes cluster to a newer version

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 11, 2019 at 4:56 AM PST by kubeadm: update the reference documentation for 1.15 (#14596) (Page History)

Edit This Page

kubeadm token

Bootstrap tokens are used for establishing bidirectional trust between a node joining the cluster and a control-plane node, as described in authenticating with bootstrap tokens.

`kubeadm init` creates an initial token with a 24-hour TTL. The following commands allow you to manage such a token and also to create and manage new ones.

- kubeadm token create
- kubeadm token delete
- kubeadm token generate
- kubeadm token list
- What's next

kubeadm token create

Create bootstrap tokens on the server

Synopsis

This command will create a bootstrap token for you. You can specify the usages for this token, the "time to live" and an optional human friendly description.

The [token] is the actual token to write. This should be a securely generated random token of the form "[a-z0-9]{6}.[a-z0-9]{16}". If no [token] is given, kubeadm will generate a random token instead.

```
kubeadm token create [token]
```

Options

```
      --config string        Path to a kubeadm configuration
file.
      --description string   A human friendly description of how
this token is used.
      --groups strings       Extra groups that this token will
authenticate as when used for authentication. Must match "\
\Asystem:bootstrappers:[a-z0-9:-]{0,255}[a-z0-9]\\z" (default
[system:bootstrappers:kubeadm:default-node-token])
  -h, --help                 help for create
      --print-join-command   Instead of printing only the token,
print the full 'kubeadm join' flag needed to join the cluster
using the token.
```

```
      --ttl duration         The duration before the token is
automatically deleted (e.g. 1s, 2m, 3h). If set to '0', the
token will never expire (default 24h0m0s)
      --usages strings       Describes the ways in which this
token can be used. You can pass --usages multiple times or
provide a comma separated list of options. Valid options:
[signing,authentication] (default [signing,authentication])
```

Options inherited from parent commands

```
      --dry-run             Whether to enable dry-run mode or not
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --rootfs string       [EXPERIMENTAL] The path to the
'real' host root filesystem.
```

kubeadm token delete

Delete bootstrap tokens on the server

Synopsis

This command will delete a list of bootstrap tokens for you.

The [token-value] is the full Token of the form "[a-z0-9]{6}.[a-z0-9]{16}" or the Token ID of the form "[a-z0-9]{6}" to delete.

```
kubeadm token delete [token-value] ...
```

Options

```
  -h, --help   help for delete
```

Options inherited from parent commands

```
      --dry-run             Whether to enable dry-run mode or not
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --rootfs string       [EXPERIMENTAL] The path to the
'real' host root filesystem.
```

kubeadm token generate

Generate and print a bootstrap token, but do not create it on the server

Synopsis

This command will print out a randomly-generated bootstrap token that can be used with the "init" and "join" commands.

You don't have to use this command in order to generate a token. You can do so yourself as long as it is in the format "[a-z0-9]{6}.[a-z0-9]{16}". This command is provided for convenience to generate tokens in the given format.

You can also use "kubeadm init" without specifying a token and it will generate and print one for you.

```
kubeadm token generate [flags]
```

Options

```
  -h, --help   help for generate
```

Options inherited from parent commands

```
      --dry-run             Whether to enable dry-run mode or not
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --rootfs string       [EXPERIMENTAL] The path to the
'real' host root filesystem.
```

kubeadm token list

List bootstrap tokens on the server

Synopsis

This command will list all bootstrap tokens for you.

```
kubeadm token list [flags]
```

Options

```
  -h, --help   help for list
```

Options inherited from parent commands

```
      --dry-run             Whether to enable dry-run mode or not
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --rootfs string       [EXPERIMENTAL] The path to the
'real' host root filesystem.
```

What's next

- kubeadm join to bootstrap a Kubernetes worker node and join it to the cluster

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on December 03, 2018 at 7:21 PM PST by Official 1.13 Release Docs (#11401) (Page History)

Edit This Page

kubeadm version

This command prints the version of kubeadm.

Print the version of kubeadm

Synopsis

Print the version of kubeadm

```
kubeadm version [flags]
```

Options

```
  -h, --help            help for version
  -o, --output string   Output format; available options are
'yaml', 'json' and 'short'
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the
'real' host root filesystem.
```

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on May 21, 2018 at 11:55 PM PST by Last batch change approvers to reviewers (#8644) (Page History)

Edit This Page

kubeadm alpha

-
  - kubeadm alpha certs renew
  - kubeadm alpha certs certificate-key
  - kubeadm alpha certs check-expiration
  - kubeadm alpha kubeconfig user
  - kubeadm alpha kubelet config
  - kubeadm alpha selfhosting pivot
  - What's next

Caution: `kubeadm alpha` provides a preview of a set of features made available for gathering feedback from the community. Please try it out and give us feedback!

kubeadm alpha certs renew

You can renew all Kubernetes certificates using the `all` subcommand or renew them selectively. For more details about certificate expiration and renewal see the certificate management documentation.

- renew
- all
- admin.conf
- apiserver-etcd-client
- apiserver-kubelet-client
- apiserver
- controller-manager.conf
- etcd-healthcheck-client
- etcd-peer
- etcd-server
- front-proxy-client
- scheduler.conf

Renew certificates for a Kubernetes cluster

Synopsis

This command is not meant to be run on its own. See list of available subcommands.

```
kubeadm alpha certs renew [flags]
```

Options

```
  -h, --help   help for renew
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Renew all available certificates

Synopsis

Renew all known certificates necessary to run the control plane. Renewals are run unconditionally, regardless of expiration date. Renewals can also be run individually for more control.

```
kubeadm alpha certs renew all [flags]
```

Options

```
      --cert-dir string     The path where to save the certificates (default "/etc/kubernetes/pki")
      --config string       Path to a kubeadm configuration file.
      --csr-dir string      The path to output the CSRs and private keys to
      --csr-only            Create CSRs instead of generating certificates
  -h, --help                help for all
      --kubeconfig string   The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file. (default "/etc/kubernetes/admin.conf")
      --use-api             Use the Kubernetes certificate API to renew certificates
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Renew the certificate embedded in the kubeconfig file for the admin to use and for kubeadm itself

Synopsis

Renew the certificate embedded in the kubeconfig file for the admin to use and for kubeadm itself.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm alpha certs renew admin.conf [flags]
```

Options

```
      --cert-dir string     The path where to save the
certificates (default "/etc/kubernetes/pki")
      --config string       Path to a kubeadm configuration file.
      --csr-dir string      The path to output the CSRs and
private keys to
      --csr-only            Create CSRs instead of generating
certificates
  -h, --help                help for admin.conf
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --use-api             Use the Kubernetes certificate API
to renew certificates
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Renew the certificate the apiserver uses to access etcd

Synopsis

Renew the certificate the apiserver uses to access etcd.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm alpha certs renew apiserver-etcd-client [flags]
```

Options

```
      --cert-dir string     The path where to save the
certificates (default "/etc/kubernetes/pki")
```

```
      --config string       Path to a kubeadm configuration file.
      --csr-dir string      The path to output the CSRs and
private keys to
      --csr-only            Create CSRs instead of generating
certificates
  -h, --help                help for apiserver-etcd-client
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --use-api             Use the Kubernetes certificate API
to renew certificates
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Renew the certificate for the API server to connect to kubelet

Synopsis

Renew the certificate for the API server to connect to kubelet.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm alpha certs renew apiserver-kubelet-client [flags]
```

Options

```
      --cert-dir string     The path where to save the
certificates (default "/etc/kubernetes/pki")
      --config string       Path to a kubeadm configuration file.
      --csr-dir string      The path to output the CSRs and
private keys to
      --csr-only            Create CSRs instead of generating
certificates
  -h, --help                help for apiserver-kubelet-client
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --use-api             Use the Kubernetes certificate API
to renew certificates
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Renew the certificate for serving the Kubernetes API

Synopsis

Renew the certificate for serving the Kubernetes API.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm alpha certs renew apiserver [flags]
```

Options

```
      --cert-dir string     The path where to save the certificates (default "/etc/kubernetes/pki")
      --config string       Path to a kubeadm configuration file.
      --csr-dir string      The path to output the CSRs and private keys to
      --csr-only            Create CSRs instead of generating certificates
  -h, --help                help for apiserver
      --kubeconfig string   The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file. (default "/etc/kubernetes/admin.conf")
      --use-api             Use the Kubernetes certificate API to renew certificates
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Renew the certificate embedded in the kubeconfig file for the controller manager to use

Synopsis

Renew the certificate embedded in the kubeconfig file for the controller manager to use.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm alpha certs renew controller-manager.conf [flags]
```

Options

```
      --cert-dir string     The path where to save the
certificates (default "/etc/kubernetes/pki")
      --config string       Path to a kubeadm configuration file.
      --csr-dir string      The path to output the CSRs and
private keys to
      --csr-only            Create CSRs instead of generating
certificates
  -h, --help                help for controller-manager.conf
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --use-api             Use the Kubernetes certificate API
to renew certificates
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Renew the certificate for liveness probes to healtcheck etcd

Synopsis

Renew the certificate for liveness probes to healtcheck etcd.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm alpha certs renew etcd-healthcheck-client [flags]
```

Options

```
      --cert-dir string     The path where to save the
certificates (default "/etc/kubernetes/pki")
      --config string       Path to a kubeadm configuration file.
      --csr-dir string      The path to output the CSRs and
private keys to
      --csr-only            Create CSRs instead of generating
certificates
  -h, --help                help for etcd-healthcheck-client
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --use-api             Use the Kubernetes certificate API
to renew certificates
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Renew the certificate for etcd nodes to communicate with each other

Synopsis

Renew the certificate for etcd nodes to communicate with each other.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm alpha certs renew etcd-peer [flags]
```

Options

```
      --cert-dir string     The path where to save the
certificates (default "/etc/kubernetes/pki")
      --config string       Path to a kubeadm configuration file.
      --csr-dir string      The path to output the CSRs and
private keys to
      --csr-only            Create CSRs instead of generating
certificates
  -h, --help                help for etcd-peer
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
```

```
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --use-api            Use the Kubernetes certificate API
to renew certificates
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Renew the certificate for serving etcd

Synopsis

Renew the certificate for serving etcd.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm alpha certs renew etcd-server [flags]
```

Options

```
      --cert-dir string     The path where to save the
certificates (default "/etc/kubernetes/pki")
      --config string       Path to a kubeadm configuration file.
      --csr-dir string      The path to output the CSRs and
private keys to
      --csr-only            Create CSRs instead of generating
certificates
  -h, --help                help for etcd-server
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --use-api             Use the Kubernetes certificate API
to renew certificates
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Renew the certificate for the front proxy client

Synopsis

Renew the certificate for the front proxy client.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm alpha certs renew front-proxy-client [flags]
```

Options

```
      --cert-dir string     The path where to save the
certificates (default "/etc/kubernetes/pki")
      --config string       Path to a kubeadm configuration file.
      --csr-dir string      The path to output the CSRs and
private keys to
      --csr-only            Create CSRs instead of generating
certificates
  -h, --help                help for front-proxy-client
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --use-api             Use the Kubernetes certificate API
to renew certificates
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Renew the certificate embedded in the kubeconfig file for the scheduler manager to use

Synopsis

Renew the certificate embedded in the kubeconfig file for the scheduler manager to use.

Renewals run unconditionally, regardless of certificate expiration date; extra attributes such as SANs will be based on the existing file/certificates, there is no need to resupply them.

Renewal by default tries to use the certificate authority in the local PKI managed by kubeadm; as alternative it is possible to use K8s certificate API for certificate renewal, or as a last option, to generate a CSR request.

After renewal, in order to make changes effective, is is required to restart control-plane components and eventually re-distribute the renewed certificate in case the file is used elsewhere.

```
kubeadm alpha certs renew scheduler.conf [flags]
```

Options

```
      --cert-dir string     The path where to save the
certificates (default "/etc/kubernetes/pki")
      --config string       Path to a kubeadm configuration file.
      --csr-dir string      The path to output the CSRs and
private keys to
      --csr-only            Create CSRs instead of generating
certificates
  -h, --help                help for scheduler.conf
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --use-api             Use the Kubernetes certificate API
to renew certificates
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm alpha certs certificate-key

This command can be used to generate a new control-plane certificate key. The key can be passed as `--certificate-key` to `kubeadm init` and `kubeadm join` to enable the automatic copy of certificates when joining additional control-plane nodes.

- certificate-key

Generate certificate keys

Synopsis

This command will print out a secure randomly-generated certificate key that can be used with the "init" command.

You can also use "kubeadm init -experimental-upload-certs" without specifying a certificate key and it will generate and print one for you.

```
kubeadm alpha certs certificate-key [flags]
```

Options

```
  -h, --help   help for certificate-key
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

kubeadm alpha certs check-expiration

This command checks expiration for the certificates in the local PKI managed by kubeadm. For more details about certificate expiration and renewal see the [certificate management documentation]().

- [check-expiration]()

Check certificates expiration for a Kubernetes cluster

Synopsis

Checks expiration for the certificates in the local PKI managed by kubeadm.

```
kubeadm alpha certs check-expiration [flags]
```

Options

```
      --cert-dir string   The path where to save the certificates (default "/etc/kubernetes/pki")
      --config string     Path to a kubeadm configuration file.
  -h, --help              help for check-expiration
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

kubeadm alpha kubeconfig user

The `user` subcommand can be used for the creation of kubeconfig files for additional users.

- [kubeconfig]()
- [user]()

Kubeconfig file utilities

Synopsis

Kubeconfig file utilities.

Alpha Disclaimer: this command is currently alpha.

Options

```
  -h, --help   help for kubeconfig
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Output a kubeconfig file for an additional user

Synopsis

Output a kubeconfig file for an additional user.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm alpha kubeconfig user [flags]
```

Examples

```
  # Output a kubeconfig file for an additional user named foo
  kubeadm alpha kubeconfig user --client-name=foo
```

Options

```
      --apiserver-advertise-address string   The IP address the API server is accessible on
      --apiserver-bind-port int32            The port the API server is accessible on (default 6443)
      --cert-dir string                      The path where certificates are stored (default "/etc/kubernetes/pki")
      --client-name string                   The name of user. It will be used as the CN if client certificates are created
  -h, --help                                 help for user
      --org strings                          The orgnizations of the client certificate. It will be used as the O if client certificates are created
      --token string                         The token that should be used as the authentication mechanism for this kubeconfig, instead of client certificates
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

kubeadm alpha kubelet config

Use the following commands to either download the kubelet configuration from the cluster or to enable the DynamicKubeletConfiguration feature.

- kubelet
- download
- enable-dynamic

Commands related to handling the kubelet

Synopsis

This command is not meant to be run on its own. See list of available subcommands.

Options

```
  -h, --help   help for kubelet
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Download the kubelet configuration from the cluster ConfigMap kubelet-config-1.X, where X is the minor version of the kubelet

Synopsis

Download the kubelet configuration from a ConfigMap of the form "kubelet-config-1.X" in the cluster, where X is the minor version of the kubelet. Either kubeadm autodetects the kubelet version by exec-ing "kubelet -version" or respects the -kubelet-version parameter.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm alpha kubelet config download [flags]
```

Examples

```
  # Download the kubelet configuration from the ConfigMap in the
cluster. Autodetect the kubelet version.
  kubeadm alpha phase kubelet config download

  # Download the kubelet configuration from the ConfigMap in the
cluster. Use a specific desired kubelet version.
  kubeadm alpha phase kubelet config download --kubelet-version
1.14.0
```

Options

```
  -h, --help                     help for download
      --kubeconfig string        The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --kubelet-version string   The desired version for the
kubelet. Defaults to being autodetected from 'kubelet --version'.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Download the kubelet configuration from the cluster ConfigMap kubelet-config-1.X, where X is the minor version of the kubelet

Synopsis

Download the kubelet configuration from a ConfigMap of the form "kubelet-config-1.X" in the cluster, where X is the minor version of the kubelet. Either kubeadm autodetects the kubelet version by exec-ing "kubelet -version" or respects the -kubelet-version parameter.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm alpha kubelet config download [flags]
```

Examples

```
  # Download the kubelet configuration from the ConfigMap in the cluster. Autodetect the kubelet version.
  kubeadm alpha phase kubelet config download

  # Download the kubelet configuration from the ConfigMap in the cluster. Use a specific desired kubelet version.
  kubeadm alpha phase kubelet config download --kubelet-version 1.14.0
```

Options

```
  -h, --help                     help for download
      --kubeconfig string        The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file. (default "/etc/kubernetes/admin.conf")
      --kubelet-version string   The desired version for the kubelet. Defaults to being autodetected from 'kubelet --version'.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

kubeadm alpha selfhosting pivot

The subcommand `pivot` can be used to convert a static Pod-hosted control plane into a self-hosted one.

[Documentation]

- [selfhosting]
- [pivot]

Make a kubeadm cluster self-hosted

Synopsis

This command is not meant to be run on its own. See list of available subcommands.

Options

```
  -h, --help   help for selfhosting
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Convert a static Pod-hosted control plane into a self-hosted one

Synopsis

Convert static Pod files for control plane components into self-hosted DaemonSets configured via the Kubernetes API.

See the documentation for self-hosting limitations.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm alpha selfhosting pivot [flags]
```

Examples

```
  # Convert a static Pod-hosted control plane into a self-hosted one.

  kubeadm alpha phase self-hosting convert-from-staticpods
```

Options

```
      --cert-dir string          The path where certificates are stored (default "/etc/kubernetes/pki")
      --config string            Path to a kubeadm configuration file.
  -f, --force                    Pivot the cluster without prompting for confirmation
  -h, --help                     help for pivot
      --kubeconfig string        The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig
```

```
file. (default "/etc/kubernetes/admin.conf")
  -s, --store-certs-in-secrets   Enable storing certs in secrets
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to connect a node to the cluster
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 11, 2019 at 4:56 AM PST by kubeadm: update the reference documentation for 1.15 (#14596) (Page History)

Edit This Page

kubeadm init phase

- - kubeadm init phase preflight
  - kubeadm init phase certs
  - kubeadm init phase kubeconfig
  - kubeadm init phase kubelet-start
  - kubeadm init phase control-plane
  - kubeadm init phase etcd
  - kubeadm init phase upload-certs
  - kubeadm init phase mark-control-plane
  - kubeadm init phase bootstrap-token
  - kubeadm init phase upload-config
  - kubeadm init phase addon
  - What's next

In v1.8.0, kubeadm introduced the `kubeadm alpha phase` command with the aim of making kubeadm more modular. In v1.13.0 this command graduated to `kubeadm init phase`. This modularity enables you to invoke atomic sub-steps of the bootstrap process. Hence, you can let kubeadm do some parts and fill in yourself where you need customizations.

`kubeadm init phase` is consistent with the kubeadm init workflow, and behind the scene both use the same code.

kubeadm init phase preflight

Using this command you can execute preflight checks on a control-plane node.

- preflight

Run pre-flight checks

Synopsis

Run pre-flight checks for kubeadm init.

```
kubeadm init phase preflight [flags]
```

Examples

```
  # Run pre-flight checks for kubeadm init using a config file.
  kubeadm init phase preflight --config kubeadm-config.yml
```

Options

```
      --config string                        Path to a kubeadm
configuration file.
  -h, --help                                 help for preflight
      --ignore-preflight-errors strings   A list of checks whose
errors will be shown as warnings. Example:
'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all
checks.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm init phase certs

Can be used to create all required certificates by kubeadm.

- certs
- all
- apiserver-etcd-client
- apiserver-kubelet-client
- apiserver
- ca
- etcd-ca
- healthcheck-client

- etcd-peer
- etcd-server
- front-proxy-ca
- front-proxy-client
- sa

Certificate generation

Synopsis

This command is not meant to be run on its own. See list of available subcommands.

```
kubeadm init phase certs [flags]
```

Options

```
  -h, --help   help for certs
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Generate all certificates

Synopsis

Generate all certificates

```
kubeadm init phase certs all [flags]
```

Options

```
      --apiserver-advertise-address string   The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
      --apiserver-cert-extra-sans strings    Optional extra Subject Alternative Names (SANs) to use for the API Server serving certificate. Can be both IP addresses and DNS names.
      --cert-dir string                      The path where to save and store the certificates. (default "/etc/kubernetes/pki")
      --config string                        Path to a kubeadm configuration file.
  -h, --help                                 help for all
      --service-cidr string                  Use alternative range of IP address for service VIPs. (default "10.96.0.0/12")
      --service-dns-domain string            Use alternative domain for services, e.g. "myorg.internal". (default "cluster.local")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Generate the certificate the apiserver uses to access etcd

Synopsis

Generate the certificate the apiserver uses to access etcd, and save them into apiserver-etcd-client.cert and apiserver-etcd-client.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm init phase certs apiserver-etcd-client [flags]
```

Options

```
      --cert-dir string   The path where to save and store the certificates. (default "/etc/kubernetes/pki")
      --config string     Path to a kubeadm configuration file.
      --csr-dir string    The path to output the CSRs and private keys to
      --csr-only          Create CSRs instead of generating certificates
  -h, --help              help for apiserver-etcd-client
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Generate the certificate for the API server to connect to kubelet

Synopsis

Generate the certificate for the API server to connect to kubelet, and save them into apiserver-kubelet-client.cert and apiserver-kubelet-client.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm init phase certs apiserver-kubelet-client [flags]
```

Options

```
      --cert-dir string   The path where to save and store the certificates. (default "/etc/kubernetes/pki")
      --config string     Path to a kubeadm configuration file.
```

```
      --csr-dir string    The path to output the CSRs and
private keys to
      --csr-only          Create CSRs instead of generating
certificates
  -h, --help              help for apiserver-kubelet-client
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generate the certificate for serving the Kubernetes API

Synopsis

Generate the certificate for serving the Kubernetes API, and save them into apiserver.cert and apiserver.key files.

Default SANs are kubernetes, kubernetes.default, kubernetes.default.svc, kubernetes.default.svc.cluster.local, 10.96.0.1, 127.0.0.1

If both files already exist, kubeadm skips the generation step and existing files will be used.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm init phase certs apiserver [flags]
```

Options

```
      --apiserver-advertise-address string   The IP address the
API Server will advertise it's listening on. If not set the
default network interface will be used.
      --apiserver-cert-extra-sans strings    Optional extra
Subject Alternative Names (SANs) to use for the API Server
serving certificate. Can be both IP addresses and DNS names.
      --cert-dir string                      The path where to
save and store the certificates. (default "/etc/kubernetes/pki")
      --config string                        Path to a kubeadm
configuration file.
      --csr-dir string                       The path to output
the CSRs and private keys to
      --csr-only                             Create CSRs instead
of generating certificates
  -h, --help                                 help for apiserver
      --service-cidr string                  Use alternative
range of IP address for service VIPs. (default "10.96.0.0/12")
      --service-dns-domain string            Use alternative
domain for services, e.g. "myorg.internal". (default
"cluster.local")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Generate the self-signed Kubernetes CA to provision identities for other Kubernetes components

Synopsis

Generate the self-signed Kubernetes CA to provision identities for other Kubernetes components, and save them into ca.cert and ca.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm init phase certs ca [flags]
```

Options

```
      --cert-dir string   The path where to save and store the certificates. (default "/etc/kubernetes/pki")
      --config string     Path to a kubeadm configuration file.
  -h, --help              help for ca
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Generate the self-signed CA to provision identities for etcd

Synopsis

Generate the self-signed CA to provision identities for etcd, and save them into etcd/ca.cert and etcd/ca.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm init phase certs etcd-ca [flags]
```

Options

```
      --cert-dir string   The path where to save and store the certificates. (default "/etc/kubernetes/pki")
      --config string     Path to a kubeadm configuration file.
  -h, --help              help for etcd-ca
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' 
host root filesystem.
```

Generate the certificate for liveness probes to healtcheck etcd

Synopsis

Generate the certificate for liveness probes to healtcheck etcd, and save them into etcd/healthcheck-client.cert and etcd/healthcheck-client.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm init phase certs etcd-healthcheck-client [flags]
```

Options

```
      --cert-dir string   The path where to save and store the 
certificates. (default "/etc/kubernetes/pki")
      --config string     Path to a kubeadm configuration file.
      --csr-dir string    The path to output the CSRs and 
private keys to
      --csr-only          Create CSRs instead of generating 
certificates
  -h, --help              help for etcd-healthcheck-client
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' 
host root filesystem.
```

Generate the certificate for etcd nodes to communicate with each other

Synopsis

Generate the certificate for etcd nodes to communicate with each other, and save them into etcd/peer.cert and etcd/peer.key files.

Default SANs are localhost, 127.0.0.1, 127.0.0.1, ::1

If both files already exist, kubeadm skips the generation step and existing files will be used.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm init phase certs etcd-peer [flags]
```

Options

```
      --cert-dir string   The path where to save and store the
certificates. (default "/etc/kubernetes/pki")
      --config string     Path to a kubeadm configuration file.
      --csr-dir string    The path to output the CSRs and
private keys to
      --csr-only          Create CSRs instead of generating
certificates
  -h, --help              help for etcd-peer
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generate the certificate for serving etcd

Synopsis

Generate the certificate for serving etcd, and save them into etcd/server.cert and etcd/server.key files.

Default SANs are localhost, 127.0.0.1, 127.0.0.1, ::1

If both files already exist, kubeadm skips the generation step and existing files will be used.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm init phase certs etcd-server [flags]
```

Options

```
      --cert-dir string   The path where to save and store the
certificates. (default "/etc/kubernetes/pki")
      --config string     Path to a kubeadm configuration file.
      --csr-dir string    The path to output the CSRs and
private keys to
      --csr-only          Create CSRs instead of generating
certificates
  -h, --help              help for etcd-server
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generate the self-signed CA to provision identities for front proxy

Synopsis

Generate the self-signed CA to provision identities for front proxy, and save them into front-proxy-ca.cert and front-proxy-ca.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm init phase certs front-proxy-ca [flags]
```

Options

```
      --cert-dir string   The path where to save and store the
certificates. (default "/etc/kubernetes/pki")
      --config string     Path to a kubeadm configuration file.
  -h, --help              help for front-proxy-ca
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generate the certificate for the front proxy client

Synopsis

Generate the certificate for the front proxy client, and save them into front-proxy-client.cert and front-proxy-client.key files.

If both files already exist, kubeadm skips the generation step and existing files will be used.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm init phase certs front-proxy-client [flags]
```

Options

```
      --cert-dir string   The path where to save and store the
certificates. (default "/etc/kubernetes/pki")
      --config string     Path to a kubeadm configuration file.
      --csr-dir string    The path to output the CSRs and
private keys to
      --csr-only          Create CSRs instead of generating
certificates
  -h, --help              help for front-proxy-client
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generate a private key for signing service account tokens along with its public key

Synopsis

Generate the private key for signing service account tokens along with its public key, and save them into sa.key and sa.pub files. If both files already exist, kubeadm skips the generation step and existing files will be used.

Alpha Disclaimer: this command is currently alpha.

```
kubeadm init phase certs sa [flags]
```

Options

```
      --cert-dir string   The path where to save and store the certificates. (default "/etc/kubernetes/pki")
  -h, --help              help for sa
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

kubeadm init phase kubeconfig

You can create all required kubeconfig files by calling the `all` subcommand or call then individually.

- kubeconfig
- all
- admin
- controller-manager
- kubelet
- scheduler

Generate all kubeconfig files necessary to establish the control plane and the admin kubeconfig file

Synopsis

This command is not meant to be run on its own. See list of available subcommands.

```
kubeadm init phase kubeconfig [flags]
```

Options

```
  -h, --help   help for kubeconfig
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Generate all kubeconfig files

Synopsis

Generate all kubeconfig files

```
kubeadm init phase kubeconfig all [flags]
```

Options

```
      --apiserver-advertise-address string   The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
      --apiserver-bind-port int32            Port for the API Server to bind to. (default 6443)
      --cert-dir string                      The path where to save and store the certificates. (default "/etc/kubernetes/pki")
      --config string                        Path to a kubeadm configuration file.
  -h, --help                                 help for all
      --kubeconfig-dir string                The path where to save the kubeconfig file. (default "/etc/kubernetes")
      --node-name string                     Specify the node name.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Generate a kubeconfig file for the admin to use and for kubeadm itself

Synopsis

Generate the kubeconfig file for the admin and for kubeadm itself, and save it to admin.conf file.

```
kubeadm init phase kubeconfig admin [flags]
```

Options

```
      --apiserver-advertise-address string   The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
      --apiserver-bind-port int32            Port for the API Server to bind to. (default 6443)
      --cert-dir string                      The path where to save and store the certificates. (default "/etc/kubernetes/pki")
```

```
      --config string                             Path to a kubeadm
configuration file.
  -h, --help                                      help for admin
      --kubeconfig-dir string                     The path where to
save the kubeconfig file. (default "/etc/kubernetes")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generate a kubeconfig file for the controller manager to use

Synopsis

Generate the kubeconfig file for the controller manager to use and save it to controller-manager.conf file

```
kubeadm init phase kubeconfig controller-manager [flags]
```

Options

```
      --apiserver-advertise-address string   The IP address the
API Server will advertise it's listening on. If not set the
default network interface will be used.
      --apiserver-bind-port int32            Port for the API
Server to bind to. (default 6443)
      --cert-dir string                      The path where to
save and store the certificates. (default "/etc/kubernetes/pki")
      --config string                        Path to a kubeadm
configuration file.
  -h, --help                                 help for controller-
manager
      --kubeconfig-dir string                The path where to
save the kubeconfig file. (default "/etc/kubernetes")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generate a kubeconfig file for the kubelet to use *only* for cluster bootstrapping purposes

Synopsis

Generate the kubeconfig file for the kubelet to use and save it to kubelet.conf file.

Please note that this should only be used for cluster bootstrapping purposes. After your control plane is up, you should request all kubelet credentials from the CSR API.

```
kubeadm init phase kubeconfig kubelet [flags]
```

Options

```
      --apiserver-advertise-address string   The IP address the
API Server will advertise it's listening on. If not set the
default network interface will be used.
      --apiserver-bind-port int32            Port for the API
Server to bind to. (default 6443)
      --cert-dir string                      The path where to
save and store the certificates. (default "/etc/kubernetes/pki")
      --config string                        Path to a kubeadm
configuration file.
  -h, --help                                 help for kubelet
      --kubeconfig-dir string                The path where to
save the kubeconfig file. (default "/etc/kubernetes")
      --node-name string                     Specify the node
name.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generate a kubeconfig file for the scheduler to use

Synopsis

Generate the kubeconfig file for the scheduler to use and save it to scheduler.conf file.

```
kubeadm init phase kubeconfig scheduler [flags]
```

Options

```
      --apiserver-advertise-address string   The IP address the
API Server will advertise it's listening on. If not set the
default network interface will be used.
      --apiserver-bind-port int32            Port for the API
Server to bind to. (default 6443)
      --cert-dir string                      The path where to
save and store the certificates. (default "/etc/kubernetes/pki")
      --config string                        Path to a kubeadm
configuration file.
  -h, --help                                 help for scheduler
      --kubeconfig-dir string                The path where to
save the kubeconfig file. (default "/etc/kubernetes")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm init phase kubelet-start

This phase will write the kubelet configuration file and environment file and then start the kubelet.

- kubelet-start

Write kubelet settings and (re)start the kubelet

Synopsis

Write a file with KubeletConfiguration and an environment file with node specific kubelet settings, and then (re)start kubelet.

```
kubeadm init phase kubelet-start [flags]
```

Examples

```
  # Writes a dynamic environment file with kubelet flags from a
InitConfiguration file.
  kubeadm init phase kubelet-start --config config.yaml
```

Options

```
      --config string       Path to a kubeadm configuration file.
      --cri-socket string   Path to the CRI socket to connect.
If empty kubeadm will try to auto-detect this value; use this
option only if you have more than one CRI installed or if you
have non-standard CRI socket.
  -h, --help                help for kubelet-start
      --node-name string    Specify the node name.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm init phase control-plane

Using this phase you can create all required static Pod files for the control plane components.

- control-plane
- all
- apiserver
- controller-manager
- scheduler

Generate all static Pod manifest files necessary to establish the control plane

Synopsis

This command is not meant to be run on its own. See list of available subcommands.

```
kubeadm init phase control-plane [flags]
```

Options

```
  -h, --help   help for control-plane
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generate all static Pod manifest files

Synopsis

Generate all static Pod manifest files

```
kubeadm init phase control-plane all [flags]
```

Examples

```
  # Generates all static Pod manifest files for control plane
components,
  # functionally equivalent to what is generated by kubeadm init.
  kubeadm init phase control-plane all

  # Generates all static Pod manifest files using options read
from a configuration file.
  kubeadm init phase control-plane all --config config.yaml
```

Options

```
      --apiserver-advertise-address string              The IP
address the API Server will advertise it's listening on. If not
set the default network interface will be used.
      --apiserver-bind-port int32                       Port for
the API Server to bind to. (default 6443)
      --apiserver-extra-args mapStringString            A set of
extra flags to pass to the API Server or override default ones
in form of <flagname>=<value>
      --cert-dir string                                 The path
where to save and store the certificates. (default "/etc/
kubernetes/pki")
      --config string                                   Path to
a kubeadm configuration file.
      --controller-manager-extra-args mapStringString   A set of
extra flags to pass to the Controller Manager or override
default ones in form of <flagname>=<value>
```

```
      --feature-gates string                            A set of
key=value pairs that describe feature gates for various
features. No feature gates are available in this release.
  -h, --help                                            help for
all
      --image-repository string                         Choose a
container registry to pull control plane images from (default
"k8s.gcr.io")
      --kubernetes-version string                       Choose a
specific Kubernetes version for the control plane. (default
"stable-1")
      --pod-network-cidr string                         Specify
range of IP addresses for the pod network. If set, the control
plane will automatically allocate CIDRs for every node.
      --scheduler-extra-args mapStringString            A set of
extra flags to pass to the Scheduler or override default ones in
form of <flagname>=<value>
      --service-cidr string                             Use
alternative range of IP address for service VIPs. (default
"10.96.0.0/12")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generates the kube-apiserver static Pod manifest

Synopsis

Generates the kube-apiserver static Pod manifest

```
kubeadm init phase control-plane apiserver [flags]
```

Options

```
      --apiserver-advertise-address string     The IP address
the API Server will advertise it's listening on. If not set the
default network interface will be used.
      --apiserver-bind-port int32              Port for the API
Server to bind to. (default 6443)
      --apiserver-extra-args mapStringString   A set of extra
flags to pass to the API Server or override default ones in form
of <flagname>=<value>
      --cert-dir string                        The path where to
save and store the certificates. (default "/etc/kubernetes/pki")
      --config string                          Path to a kubeadm
configuration file.
      --feature-gates string                   A set of
key=value pairs that describe feature gates for various
features. No feature gates are available in this release.
  -h, --help                                   help for apiserver
      --image-repository string                Choose a
```

```
container registry to pull control plane images from (default
"k8s.gcr.io")
      --kubernetes-version string              Choose a specific
Kubernetes version for the control plane. (default "stable-1")
      --service-cidr string                    Use alternative
range of IP address for service VIPs. (default "10.96.0.0/12")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generates the kube-controller-manager static Pod manifest

Synopsis

Generates the kube-controller-manager static Pod manifest

```
kubeadm init phase control-plane controller-manager [flags]
```

Options

```
      --cert-dir string                                 The path
where to save and store the certificates. (default "/etc/
kubernetes/pki")
      --config string                                   Path to
a kubeadm configuration file.
      --controller-manager-extra-args mapStringString   A set of
extra flags to pass to the Controller Manager or override
default ones in form of <flagname>=<value>
  -h, --help                                            help for
controller-manager
      --image-repository string                         Choose a
container registry to pull control plane images from (default
"k8s.gcr.io")
      --kubernetes-version string                       Choose a
specific Kubernetes version for the control plane. (default
"stable-1")
      --pod-network-cidr string                         Specify
range of IP addresses for the pod network. If set, the control
plane will automatically allocate CIDRs for every node.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generates the kube-scheduler static Pod manifest

Synopsis

Generates the kube-scheduler static Pod manifest

```
kubeadm init phase control-plane scheduler [flags]
```

Options

```
      --cert-dir string                         The path where to
save and store the certificates. (default "/etc/kubernetes/pki")
      --config string                           Path to a kubeadm
configuration file.
  -h, --help                                    help for scheduler
      --image-repository string                 Choose a
container registry to pull control plane images from (default
"k8s.gcr.io")
      --kubernetes-version string               Choose a specific
Kubernetes version for the control plane. (default "stable-1")
      --scheduler-extra-args mapStringString    A set of extra
flags to pass to the Scheduler or override default ones in form
of <flagname>=<value>
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm init phase etcd

Use the following phase to create a local etcd instance based on a static Pod file.

- etcd
- local

Generate static Pod manifest file for local etcd

Synopsis

This command is not meant to be run on its own. See list of available subcommands.

```
kubeadm init phase etcd [flags]
```

Options

```
  -h, --help   help for etcd
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Generate the static Pod manifest file for a local, single-node local etcd instance

Synopsis

Generate the static Pod manifest file for a local, single-node local etcd instance

```
kubeadm init phase etcd local [flags]
```

Examples

```
  # Generates the static Pod manifest file for etcd, functionally
  # equivalent to what is generated by kubeadm init.
  kubeadm init phase etcd local

  # Generates the static Pod manifest file for etcd using options
  # read from a configuration file.
  kubeadm init phase etcd local --config config.yaml
```

Options

```
      --cert-dir string           The path where to save and
store the certificates. (default "/etc/kubernetes/pki")
      --config string             Path to a kubeadm
configuration file.
  -h, --help                      help for local
      --image-repository string   Choose a container registry to
pull control plane images from (default "k8s.gcr.io")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm init phase upload-certs

Use the following phase to upload control-plane certificates to the cluster. By default the certs and encryption key expire after two hours.

- upload-certs

Upload certificates to kubeadm-certs

Synopsis

This command is not meant to be run on its own. See list of available subcommands.

```
kubeadm init phase upload-certs [flags]
```

Options

```
      --certificate-key string       Key used to encrypt the
control-plane certificates in the kubeadm-certs Secret.
      --config string                Path to a kubeadm
```

```
configuration file.
  -h, --help                            help for upload-certs
      --skip-certificate-key-print   Don't print the key used to
encrypt the control-plane certificates.
      --upload-certs                    Upload control-plane
certificates to the kubeadm-certs Secret.
```

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

kubeadm init phase mark-control-plane

Use the following phase to label and taint the node with the `node-role.kubernetes.io/master=""` key-value pair.

- mark-control-plane

Mark a node as a control-plane

Synopsis

Mark a node as a control-plane

```
kubeadm init phase mark-control-plane [flags]
```

Examples

```
  # Applies control-plane label and taint to the current node,
functionally equivalent to what executed by kubeadm init.
  kubeadm init phase mark-control-plane --config config.yml

  # Applies control-plane label and taint to a specific node
  kubeadm init phase mark-control-plane --node-name myNode
```

Options

```
      --config string       Path to a kubeadm configuration file.
  -h, --help                help for mark-control-plane
      --node-name string    Specify the node name.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm init phase bootstrap-token

Use the following phase to configure bootstrap tokens.

- bootstrap-token

Generates bootstrap tokens used to join a node to a cluster

Synopsis

Bootstrap tokens are used for establishing bidirectional trust between a node joining the cluster and a the control-plane node.

This command makes all the configurations required to make bootstrap tokens works and then creates an initial token.

```
kubeadm init phase bootstrap-token [flags]
```

Examples

```
  # Make all the bootstrap token configurations and create an
initial token, functionally
  # equivalent to what generated by kubeadm init.
  kubeadm init phase bootstrap-token
```

Options

```
      --config string       Path to a kubeadm configuration file.
  -h, --help                help for bootstrap-token
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --skip-token-print    Skip printing of the default
bootstrap token generated by 'kubeadm init'.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm init phase upload-config

You can use this command to upload the kubeadm configuration to your cluster. Alternatively, you can use kubeadm config.

- upload-config
- all
- kubeadm
- kubelet

Upload the kubeadm and kubelet configuration to a ConfigMap

Synopsis

This command is not meant to be run on its own. See list of available subcommands.

```
kubeadm init phase upload-config [flags]
```

Options

```
  -h, --help   help for upload-config
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Upload all configuration to a config map

Synopsis

Upload all configuration to a config map

```
kubeadm init phase upload-config all [flags]
```

Options

```
      --config string       Path to a kubeadm configuration file.
  -h, --help                help for all
      --kubeconfig string   The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file. (default "/etc/kubernetes/admin.conf")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Upload the kubeadm ClusterConfiguration to a ConfigMap

Synopsis

Upload the kubeadm ClusterConfiguration to a ConfigMap called kubeadm-config in the kube-system namespace. This enables correct configuration of system components and a seamless user experience when upgrading.

Alternatively, you can use kubeadm config.

```
kubeadm init phase upload-config kubeadm [flags]
```

Examples

```
  # upload the configuration of your cluster
  kubeadm init phase upload-config --config=myConfig.yaml
```

Options

```
      --config string       Path to a kubeadm configuration file.
  -h, --help                help for kubeadm
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Upload the kubelet component config to a ConfigMap

Synopsis

Upload kubelet configuration extracted from the kubeadm InitConfiguration object to a ConfigMap of the form kubelet-config-1.X in the cluster, where X is the minor version of the current (API Server) Kubernetes version.

```
kubeadm init phase upload-config kubelet [flags]
```

Examples

```
  # Upload the kubelet configuration from the kubeadm Config
file to a ConfigMap in the cluster.
  kubeadm init phase upload-config kubelet --config kubeadm.yaml
```

Options

```
      --config string       Path to a kubeadm configuration file.
  -h, --help                help for kubelet
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm init phase addon

You can install all the available addons with the `all` subcommand, or install them selectively.

- addon
- all
- kube-proxy
- coredns

Install required addons for passing Conformance tests

Synopsis

This command is not meant to be run on its own. See list of available subcommands.

```
kubeadm init phase addon [flags]
```

Options

```
  -h, --help   help for addon
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

Install all the addons

Synopsis

Install all the addons

```
kubeadm init phase addon all [flags]
```

Options

```
      --apiserver-advertise-address string   The IP address the API Server will advertise it's listening on. If not set the default network interface will be used.
      --apiserver-bind-port int32            Port for the API Server to bind to. (default 6443)
      --config string                        Path to a kubeadm configuration file.
      --feature-gates string                 A set of key=value pairs that describe feature gates for various features. No feature gates are available in this release.
  -h, --help                                 help for all
      --image-repository string              Choose a container registry to pull control plane images from (default "k8s.gcr.io")
      --kubeconfig string                    The kubeconfig file to use when talking to the cluster. If the flag is not set, a
```

```
set of standard locations can be searched for an existing
kubeconfig file. (default "/etc/kubernetes/admin.conf")
      --kubernetes-version string            Choose a specific
Kubernetes version for the control plane. (default "stable-1")
      --pod-network-cidr string              Specify range of IP
addresses for the pod network. If set, the control plane will
automatically allocate CIDRs for every node.
      --service-cidr string                  Use alternative
range of IP address for service VIPs. (default "10.96.0.0/12")
      --service-dns-domain string            Use alternative
domain for services, e.g. "myorg.internal". (default
"cluster.local")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Install the kube-proxy addon to a Kubernetes cluster

Synopsis

Install the kube-proxy addon components via the API server.

```
kubeadm init phase addon kube-proxy [flags]
```

Options

```
      --apiserver-advertise-address string   The IP address the
API Server will advertise it's listening on. If not set the
default network interface will be used.
      --apiserver-bind-port int32            Port for the API
Server to bind to. (default 6443)
      --config string                        Path to a kubeadm
configuration file.
  -h, --help                                 help for kube-proxy
      --image-repository string              Choose a container
registry to pull control plane images from (default "k8s.gcr.io")
      --kubeconfig string                    The kubeconfig file
to use when talking to the cluster. If the flag is not set, a
set of standard locations can be searched for an existing
kubeconfig file. (default "/etc/kubernetes/admin.conf")
      --kubernetes-version string            Choose a specific
Kubernetes version for the control plane. (default "stable-1")
      --pod-network-cidr string              Specify range of IP
addresses for the pod network. If set, the control plane will
automatically allocate CIDRs for every node.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Install the CoreDNS addon to a Kubernetes cluster

Synopsis

Install the CoreDNS addon components via the API server. Please note that although the DNS server is deployed, it will not be scheduled until CNI is installed.

```
kubeadm init phase addon coredns [flags]
```

Options

```
      --config string                Path to a kubeadm configuration file.
      --feature-gates string         A set of key=value pairs that describe feature gates for various features. No feature gates are available in this release.
  -h, --help                         help for coredns
      --image-repository string      Choose a container registry to pull control plane images from (default "k8s.gcr.io")
      --kubeconfig string            The kubeconfig file to use when talking to the cluster. If the flag is not set, a set of standard locations can be searched for an existing kubeconfig file. (default "/etc/kubernetes/admin.conf")
      --kubernetes-version string    Choose a specific Kubernetes version for the control plane. (default "stable-1")
      --service-cidr string          Use alternative range of IP address for service VIPs. (default "10.96.0.0/12")
      --service-dns-domain string    Use alternative domain for services, e.g. "myorg.internal". (default "cluster.local")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

To use kube-dns instead of CoreDNS you have to pass a configuration file:

```
# for installing a DNS addon only
kubeadm init phase addon coredns --config=someconfig.yaml
# for creating a complete control plane node
kubeadm init --config=someconfig.yaml
# for listing or pulling images
kubeadm config images list/pull --config=someconfig.yaml
# for upgrades
kubeadm upgrade apply --config=someconfig.yaml
```

The file has to contain a `DNS` field in`ClusterConfiguration` and also a type for the addon - `kube-dns` (default value is `CoreDNS`).

```
apiVersion: kubeadm.k8s.io/v1beta2
kind: ClusterConfiguration
```

```
dns:
  type: "kube-dns"
```

For more details on each field in the `v1beta2` configuration you can navigate to our API reference pages.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to connect a node to the cluster
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`
- kubeadm alpha to try experimental functionality

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 11, 2019 at 4:56 AM PST by kubeadm: update the reference documentation for 1.15 (#14596) (Page History)

Edit This Page

kubeadm join phase

- - kubeadm join phase
  - kubeadm join phase preflight
  - kubeadm join phase control-plane-prepare
  - kubeadm join phase kubelet-start
  - kubeadm join phase control-plane-join
  - What's next

In v1.14.0, kubeadm introduces the `kubeadm join phase` command with the aim of making kubeadm more modular. This modularity enables you to invoke atomic sub-steps of the join process. Hence, you can let kubeadm do some parts and fill in yourself where you need customizations.

`kubeadm join phase` is consistent with the kubeadm join workflow, and behind the scene both use the same code.

kubeadm join phase

- phase

Use this command to invoke single phase of the join workflow

Synopsis

Use this command to invoke single phase of the join workflow

Options

-h, --help	
	help for phase

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

kubeadm join phase preflight

Using this phase you can execute preflight checks on a joining node.

- preflight

Run join pre-flight checks

Synopsis

Run pre-flight checks for kubeadm join.

```
kubeadm join phase preflight [api-server-endpoint] [flags]
```

Examples

```
  # Run join pre-flight checks using a config file.
  kubeadm join phase preflight --config kubeadm-config.yml
```

Options

```
      --apiserver-advertise-address string            If the
node should host a new control plane instance, the IP address
the API Server will advertise it's listening on. If not set the
default network interface will be used.
      --apiserver-bind-port int32                     If the
node should host a new control plane instance, the port for the
API Server to bind to. (default 6443)
      --certificate-key string                        Use this
```

```
key to decrypt the certificate secrets uploaded by init.
      --config string                                 Path to
kubeadm config file.
      --control-plane                                 Create a
new control plane instance on this node
      --cri-socket string                             Path to
the CRI socket to connect. If empty kubeadm will try to auto-
detect this value; use this option only if you have more than
one CRI installed or if you have non-standard CRI socket.
      --discovery-file string                         For file-
based discovery, a file or URL from which to load cluster
information.
      --discovery-token string                        For token-
based discovery, the token used to validate cluster information
fetched from the API server.
      --discovery-token-ca-cert-hash strings          For token-
based discovery, validate that the root CA public key matches
this hash (format: "<type>:<value>").
      --discovery-token-unsafe-skip-ca-verification   For token-
based discovery, allow joining without --discovery-token-ca-cert-
hash pinning.
      --experimental-control-plane                    Create a
new control plane instance on this node
  -h, --help                                          help for
preflight
      --ignore-preflight-errors strings               A list of
checks whose errors will be shown as warnings. Example:
'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all
checks.
      --node-name string                              Specify
the node name.
      --tls-bootstrap-token string                    Specify
the token used to temporarily authenticate with the Kubernetes
Control Plane while joining the node.
      --token string                                  Use this
token for both discovery-token and tls-bootstrap-token when
those values are not provided.
```

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

kubeadm join phase control-plane-prepare

Using this phase you can prepare a node for serving a control-plane.

- control-plane-prepare
- all
- download-certs
- certs
- kubeconfig

- control-plane

Prepare the machine for serving a control plane

Synopsis

Prepare the machine for serving a control plane

```
kubeadm join phase control-plane-prepare [flags]
```

Examples

```
  # Prepares the machine for serving a control plane
  kubeadm join phase control-plane-prepare all
```

Options

-h, --help	
	help for control-plane-prepare

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

Prepare the machine for serving a control plane

Synopsis

Prepare the machine for serving a control plane

```
kubeadm join phase control-plane-prepare all [api-server-
endpoint] [flags]
```

Options

```
      --apiserver-advertise-address string            If the
node should host a new control plane instance, the IP address
the API Server will advertise it's listening on. If not set the
default network interface will be used.
      --apiserver-bind-port int32                     If the
node should host a new control plane instance, the port for the
API Server to bind to. (default 6443)
      --certificate-key string                        Use this
key to decrypt the certificate secrets uploaded by init.
      --config string                                 Path to
kubeadm config file.
      --control-plane                                 Create a
new control plane instance on this node
      --discovery-file string                         For file-
based discovery, a file or URL from which to load cluster
information.
```

```
      --discovery-token string                        For token-
based discovery, the token used to validate cluster information
fetched from the API server.
      --discovery-token-ca-cert-hash strings          For token-
based discovery, validate that the root CA public key matches
this hash (format: "<type>:<value>").
      --discovery-token-unsafe-skip-ca-verification   For token-
based discovery, allow joining without --discovery-token-ca-cert-
hash pinning.
      --experimental-control-plane                    Create a
new control plane instance on this node
  -h, --help                                          help for
all
      --node-name string                              Specify
the node name.
      --tls-bootstrap-token string                    Specify
the token used to temporarily authenticate with the Kubernetes
Control Plane while joining the node.
      --token string                                  Use this
token for both discovery-token and tls-bootstrap-token when
those values are not provided.
```

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

[EXPERIMENTAL] Download certificates shared among control-plane nodes from the kubeadm-certs Secret

Synopsis

[EXPERIMENTAL] Download certificates shared among control-plane nodes from the kubeadm-certs Secret

```
kubeadm join phase control-plane-prepare download-certs [api-
server-endpoint] [flags]
```

Options

```
      --certificate-key string                        Use this
key to decrypt the certificate secrets uploaded by init.
      --config string                                 Path to
kubeadm config file.
      --control-plane                                 Create a
new control plane instance on this node
      --discovery-file string                         For file-
based discovery, a file or URL from which to load cluster
information.
      --discovery-token string                        For token-
based discovery, the token used to validate cluster information
fetched from the API server.
      --discovery-token-ca-cert-hash strings          For token-
```

```
based discovery, validate that the root CA public key matches
this hash (format: "<type>:<value>").
      --discovery-token-unsafe-skip-ca-verification   For token-
based discovery, allow joining without --discovery-token-ca-cert-
hash pinning.
      --experimental-control-plane                    Create a
new control plane instance on this node
  -h, --help                                          help for
download-certs
      --tls-bootstrap-token string                    Specify
the token used to temporarily authenticate with the Kubernetes
Control Plane while joining the node.
      --token string                                  Use this
token for both discovery-token and tls-bootstrap-token when
those values are not provided.
```

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

Generate the certificates for the new control plane components

Synopsis

Generate the certificates for the new control plane components

```
kubeadm join phase control-plane-prepare certs [api-server-
endpoint] [flags]
```

Options

```
      --apiserver-advertise-address string            If the
node should host a new control plane instance, the IP address
the API Server will advertise it's listening on. If not set the
default network interface will be used.
      --config string                                 Path to
kubeadm config file.
      --control-plane                                 Create a
new control plane instance on this node
      --discovery-file string                         For file-
based discovery, a file or URL from which to load cluster
information.
      --discovery-token string                        For token-
based discovery, the token used to validate cluster information
fetched from the API server.
      --discovery-token-ca-cert-hash strings          For token-
based discovery, validate that the root CA public key matches
this hash (format: "<type>:<value>").
      --discovery-token-unsafe-skip-ca-verification   For token-
based discovery, allow joining without --discovery-token-ca-cert-
hash pinning.
      --experimental-control-plane                    Create a
```

```
new control plane instance on this node
  -h, --help                                          help for
certs
      --node-name string                              Specify
the node name.
      --tls-bootstrap-token string                    Specify
the token used to temporarily authenticate with the Kubernetes
Control Plane while joining the node.
      --token string                                  Use this
token for both discovery-token and tls-bootstrap-token when
those values are not provided.
```

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

Generate the kubeconfig for the new control plane components

Synopsis

Generate the kubeconfig for the new control plane components

```
kubeadm join phase control-plane-prepare kubeconfig [api-server-
endpoint] [flags]
```

Options

```
      --certificate-key string                        Use this
key to decrypt the certificate secrets uploaded by init.
      --config string                                 Path to
kubeadm config file.
      --control-plane                                 Create a
new control plane instance on this node
      --discovery-file string                         For file-
based discovery, a file or URL from which to load cluster
information.
      --discovery-token string                        For token-
based discovery, the token used to validate cluster information
fetched from the API server.
      --discovery-token-ca-cert-hash strings          For token-
based discovery, validate that the root CA public key matches
this hash (format: "<type>:<value>").
      --discovery-token-unsafe-skip-ca-verification   For token-
based discovery, allow joining without --discovery-token-ca-cert-
hash pinning.
      --experimental-control-plane                    Create a
new control plane instance on this node
  -h, --help                                          help for
kubeconfig
      --tls-bootstrap-token string                    Specify
the token used to temporarily authenticate with the Kubernetes
Control Plane while joining the node.
```

```
      --token string                                  Use this
token for both discovery-token and tls-bootstrap-token when
those values are not provided.
```

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

Generate the manifests for the new control plane components

Synopsis

Generate the manifests for the new control plane components

```
kubeadm join phase control-plane-prepare control-plane [flags]
```

Options

```
      --apiserver-advertise-address string   If the node should
host a new control plane instance, the IP address the API Server
will advertise it's listening on. If not set the default network
interface will be used.
      --apiserver-bind-port int32            If the node should
host a new control plane instance, the port for the API Server
to bind to. (default 6443)
      --config string                        Path to kubeadm
config file.
      --control-plane                        Create a new
control plane instance on this node
      --experimental-control-plane           Create a new
control plane instance on this node
  -h, --help                                 help for control-
plane
```

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

kubeadm join phase kubelet-start

Using this phase you can write the kubelet settings, certificates and (re)start the kubelet.

- kubelet-start

Write kubelet settings, certificates and (re)start the kubelet

Synopsis

Write a file with KubeletConfiguration and an environment file with node specific kubelet settings, and then (re)start kubelet.

```
kubeadm join phase kubelet-start [api-server-endpoint] [flags]
```

Options

--config string	
	Path to kubeadm config file.
--cri-socket string	
	Path to the CRI socket to connect. If empty kubeadm will try to auto-detect this value; use this option only if you have more than one CRI installed or if you have non-standard CRI socket.
--discovery-file string	
	For file-based discovery, a file or URL from which to load cluster information.
--discovery-token string	
	For token-based discovery, the token used to validate cluster information fetched from the API server.
--discovery-token-ca-cert-hash stringSlice	
	For token-based discovery, validate that the root CA public key matches this hash (format: "<type>:<value>").
--discovery-token-unsafe-skip-ca-verification	
	For token-based discovery, allow joining without --discovery-token-ca-cert-hash pinning.
-h, --help	
	help for kubelet-start
--node-name string	
	Specify the node name.
--tls-bootstrap-token string	
	Specify the token used to temporarily authenticate with the Kubernetes Control Plane while joining the node.
--token string	
	Use this token for both discovery-token and tls-bootstrap-token when those values are not provided.

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

kubeadm join phase control-plane-join

Using this phase you can join a node as a control-plane instance.

- [control-plane-join](#)

- all
- etcd
- update-status
- mark-control-plane

Join a machine as a control plane instance

Synopsis

Join a machine as a control plane instance

```
kubeadm join phase control-plane-join [flags]
```

Examples

```
  # Joins a machine as a control plane instance
  kubeadm join phase control-plane-join all
```

Options

-h, --help	
	help for control-plane-join

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

Join a machine as a control plane instance

Synopsis

Join a machine as a control plane instance

```
kubeadm join phase control-plane-join all [flags]
```

Options

```
      --apiserver-advertise-address string   If the node should
host a new control plane instance, the IP address the API Server
will advertise it's listening on. If not set the default network
interface will be used.
      --config string                        Path to kubeadm
config file.
      --control-plane                        Create a new
control plane instance on this node
      --experimental-control-plane           Create a new
control plane instance on this node
  -h, --help                                 help for all
      --node-name string                     Specify the node
name.
```

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

Add a new local etcd member

Synopsis

Add a new local etcd member

```
kubeadm join phase control-plane-join etcd [flags]
```

Options

```
      --apiserver-advertise-address string   If the node should
host a new control plane instance, the IP address the API Server
will advertise it's listening on. If not set the default network
interface will be used.
      --config string                        Path to kubeadm
config file.
      --control-plane                        Create a new
control plane instance on this node
      --experimental-control-plane           Create a new
control plane instance on this node
  -h, --help                                 help for etcd
      --node-name string                     Specify the node
name.
```

Options inherited from parent commands

--rootfs string	
	[EXPERIMENTAL] The path to the 'real' host root filesystem.

Register the new control-plane node into the ClusterStatus maintained in the kubeadm-config ConfigMap

Synopsis

Register the new control-plane node into the ClusterStatus maintained in the kubeadm-config ConfigMap

```
kubeadm join phase control-plane-join update-status [flags]
```

Options

```
      --apiserver-advertise-address string   If the node should
host a new control plane instance, the IP address the API Server
will advertise it's listening on. If not set the default network
interface will be used.
      --config string                        Path to kubeadm
```

```
config file.
      --control-plane                             Create a new
control plane instance on this node
      --experimental-control-plane                Create a new
control plane instance on this node
  -h, --help                                      help for update-
status
      --node-name string                          Specify the node
name.
```

Options inherited from parent commands

--rootfs string
[EXPERIMENTAL] The path to the 'real' host root filesystem.

Mark a node as a control-plane

Synopsis

Mark a node as a control-plane

```
kubeadm join phase control-plane-join mark-control-plane [flags]
```

Options

```
      --config string                     Path to kubeadm config file.
      --control-plane                     Create a new control plane
instance on this node
      --experimental-control-plane        Create a new control plane
instance on this node
  -h, --help                              help for mark-control-plane
      --node-name string                  Specify the node name.
```

Options inherited from parent commands

--rootfs string
[EXPERIMENTAL] The path to the 'real' host root filesystem.

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to connect a node to the cluster
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`
- kubeadm alpha to try experimental functionality

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on March 25, 2019 at 5:06 PM PST by Official 1.14 Release Docs (#13174) (Page History)

Edit This Page

kubeadm reset phase

- - kubeadm reset phase
  - kubeadm reset phase preflight
  - kubeadm reset phase update-cluster-status
  - kubeadm reset phase remove-etcd-member
  - kubeadm reset phase cleanup-node
  - What's next

In v1.15.0, kubeadm introduces the `kubeadm reset phase` command with the aim of making kubeadm more modular. This modularity enables you to invoke atomic sub-steps of the reset process. Hence, you can let kubeadm do some parts and fill in yourself where you need customizations.

`kubeadm reset phase` is consistent with the kubeadm reset workflow, and behind the scene both use the same code.

kubeadm reset phase

- phase

Use this command to invoke single phase of the reset workflow

Synopsis

Use this command to invoke single phase of the reset workflow

Options

```
  -h, --help   help for phase
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

kubeadm reset phase preflight

Using this phase you can execute preflight checks on a node that is being reset.

- preflight

Run reset pre-flight checks

Synopsis

Run pre-flight checks for kubeadm reset.

```
kubeadm reset phase preflight [flags]
```

Options

```
  -f, --force                           Reset the node without
prompting for confirmation.
  -h, --help                            help for preflight
      --ignore-preflight-errors strings   A list of checks whose
errors will be shown as warnings. Example:
'IsPrivilegedUser,Swap'. Value 'all' ignores errors from all
checks.
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm reset phase update-cluster-status

Using this phase you can remove this control-plane node from the ClusterStatus object.

- update-cluster-status

Remove this node from the ClusterStatus object.

Synopsis

Remove this node from the ClusterStatus object if the node is a control plane node.

```
kubeadm reset phase update-cluster-status [flags]
```

Options

```
  -h, --help   help for update-cluster-status
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm reset phase remove-etcd-member

Using this phase you can remove this control-plane node's etcd member from the etcd cluster.

- remove-etcd-member

Remove a local etcd member.

Synopsis

Remove a local etcd member for a control plane node.

```
kubeadm reset phase remove-etcd-member [flags]
```

Options

```
  -h, --help                help for remove-etcd-member
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

kubeadm reset phase cleanup-node

Using this phase you can perform cleanup on this node.

- cleanup-node

Run cleanup node.

Synopsis

Run cleanup node.

```
kubeadm reset phase cleanup-node [flags]
```

Options

```
      --cert-dir string     The path to the directory where the
certificates are stored. If specified, clean this directory.
(default "/etc/kubernetes/pki")
      --cri-socket string   Path to the CRI socket to connect.
If empty kubeadm will try to auto-detect this value; use this
option only if you have more than one CRI installed or if you
have non-standard CRI socket.
  -h, --help                help for cleanup-node
```

Options inherited from parent commands

```
      --rootfs string    [EXPERIMENTAL] The path to the 'real' host root filesystem.
```

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to connect a node to the cluster
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`
- kubeadm alpha to try experimental functionality

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 11, 2019 at 4:56 AM PST by kubeadm: update the reference documentation for 1.15 (#14596) (Page History)

Edit This Page

kubeadm upgrade phase

-
  - kubeadm upgrade node phase
  - What's next

In v1.15.0, kubeadm introduced preliminary support for `kubeadm upgrade node` phases. Phases for other `kubeadm upgrade` sub-commands such as `apply`, could be added in the following releases.

kubeadm upgrade node phase

Using this phase you can choose to execute the separate steps of the upgrade of secondary control-plane or worker nodes. Please note that `kubeadm upgrade apply` still has to be called on a primary control-plane node.

- phase
- control-plane
- kubelet-config

Use this command to invoke single phase of the node workflow

Synopsis

Use this command to invoke single phase of the node workflow

Options

```
  -h, --help   help for phase
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Upgrade the control plane instance deployed on this node, if any

Synopsis

Upgrade the control plane instance deployed on this node, if any

```
kubeadm upgrade node phase control-plane [flags]
```

Options

```
      --dry-run             Do not change any state, just output
the actions that would be performed.
  -h, --help                help for control-plane
      --kubeconfig string   The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
```

Options inherited from parent commands

```
      --rootfs string   [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

Upgrade the kubelet configuration for this node

Synopsis

Download the kubelet configuration from a ConfigMap of the form "kubelet-config-1.X" in the cluster, where X is the minor version of the kubelet. kubeadm uses the KuberneteVersion field in the kubeadm-config ConfigMap to determine what the desired kubelet version is, but the user can override this by using the -kubelet-version parameter.

```
kubeadm upgrade node phase kubelet-config [flags]
```

Options

```
      --dry-run                      Do not change any state, just
output the actions that would be performed.
  -h, --help                         help for kubelet-config
      --kubeconfig string            The kubeconfig file to use when
talking to the cluster. If the flag is not set, a set of
standard locations can be searched for an existing kubeconfig
file. (default "/etc/kubernetes/admin.conf")
      --kubelet-version string       The *desired* version for the
kubelet config after the upgrade. If not specified, the
KubernetesVersion from the kubeadm-config ConfigMap will be used
```

Options inherited from parent commands

```
      --rootfs string    [EXPERIMENTAL] The path to the 'real'
host root filesystem.
```

What's next

- kubeadm init to bootstrap a Kubernetes control-plane node
- kubeadm join to connect a node to the cluster
- kubeadm reset to revert any changes made to this host by `kubeadm init` or `kubeadm join`
- kubeadm upgrade to upgrade a kubeadm node
- kubeadm alpha to try experimental functionality

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 11, 2019 at 4:56 AM PST by kubeadm: update the reference documentation for 1.15 (#14596) (Page History)

Edit This Page

Implementation details

FEATURE STATE: `Kubernetes v1.10` stable

This feature is *stable*, meaning:

- The version name is vX where X is an integer.
- Stable versions of features will appear in released software for many subsequent versions.

`kubeadm init` and `kubeadm join` together provides a nice user experience for creating a best-practice but bare Kubernetes cluster from scratch. However, it might not be obvious *how* kubeadm does that.

This document provides additional details on what happen under the hood, with the aim of sharing knowledge on Kubernetes cluster best practices.

- [Core design principles]
- [Constants and well-known values and paths]
- [kubeadm init workflow internal design]
- [kubeadm join phases internal design]
- [TLS Bootstrap]

Core design principles

The cluster that `kubeadm init` and `kubeadm join` set up should be:

- **Secure**: It should adopt latest best-practices like:
  - enforcing RBAC
  - using the Node Authorizer
  - using secure communication between the control plane components
  - using secure communication between the API server and the kubelets
  - lock-down the kubelet API
  - locking down access to the API for system components like the kube-proxy and CoreDNS
  - locking down what a Bootstrap Token can access
  - etc.
- **Easy to use**: The user should not have to run anything more than a couple of commands:
  - `kubeadm init`
  - `export KUBECONFIG=/etc/kubernetes/admin.conf`
  - `kubectl apply -f <network-of-choice.yaml>`
  - `kubeadm join --token <token> <master-ip>:<master-port>`
- **Extendable**:
  - It should for example *not* favor any network provider, instead configuring a network is out-of-scope
  - Should provide the possibility to use a config file for customizing various parameters

Constants and well-known values and paths

In order to reduce complexity and to simplify development of an on-top-of-kubeadm-implemented deployment solution, kubeadm uses a limited set of constants values for well know-known paths and file names.

The Kubernetes directory `/etc/kubernetes` is a constant in the application, since it is clearly the given path in a majority of cases, and the most intuitive location; other constants paths and file names are:

- `/etc/kubernetes/manifests` as the path where kubelet should look for static Pod manifests. Names of static Pod manifests are:
  - `etcd.yaml`
  - `kube-apiserver.yaml`
  - `kube-controller-manager.yaml`
  - `kube-scheduler.yaml`
- `/etc/kubernetes/` as the path where kubeconfig files with identities for control plane components are stored. Names of kubeconfig files are:
  - `kubelet.conf` (`bootstrap-kubelet.conf` during TLS bootstrap)
  - `controller-manager.conf`
  - `scheduler.conf`
  - `admin.conf` for the cluster admin and kubeadm itself
- Names of certificates and key files :
  - `ca.crt`, `ca.key` for the Kubernetes certificate authority
  - `apiserver.crt`, `apiserver.key` for the API server certificate
  - `apiserver-kubelet-client.crt`, `apiserver-kubelet-client.key` for the client certificate used by the API server to connect to the kubelets securely
  - `sa.pub`, `sa.key` for the key used by the controller manager when signing ServiceAccount
  - `front-proxy-ca.crt`, `front-proxy-ca.key` for the front proxy certificate authority
  - `front-proxy-client.crt`, `front-proxy-client.key` for the front proxy client

kubeadm init workflow internal design

The `kubeadm init` internal workflow consists of a sequence of atomic work tasks to perform, as described in `kubeadm init`.

The `kubeadm init phase` command allows users to invoke individually each task, and ultimately offers a reusable and composable API/toolbox that can be used by other Kubernetes bootstrap tools, by any IT automation tool or by advanced user for creating custom clusters.

Preflight checks

Kubeadm executes a set of preflight checks before starting the init, with the aim to verify preconditions and avoid common cluster startup problems. In any case the user can skip specific preflight checks (or eventually all preflight checks) with the `--ignore-preflight-errors` option.

- [warning] If the Kubernetes version to use (specified with the `--kubernetes-version` flag) is at least one minor version higher than the kubeadm CLI version.
- Kubernetes system requirements:
  - if running on linux:
  - [error] if not Kernel 3.10+ or 4+ with specific KernelSpec
  - [error] if required cgroups subsystem aren't in set up
  - if using docker:
  - [warning/error] if Docker service does not exist, if it is disabled, if it is not active.

    - [error] if Docker endpoint does not exist or does not work
    - [warning] if docker version >17.03
    - If using other cri engine:
    - [error] if crictl socket does not answer
- [error] if user is not root
- [error] if the machine hostname is not a valid DNS subdomain
- [warning] if the host name cannot be reached via network lookup
- [error] if kubelet version is lower that the minimum kubelet version supported by kubeadm (current minor -1)
- [error] if kubelet version is at least one minor higher than the required controlplane version (unsupported version skew)
- [warning] if kubelet service does not exist or if it is disabled
- [warning] if firewalld is active
- [error] if API server bindPort or ports 10250/10251/10252 are used
- [Error] if `/etc/kubernetes/manifest` folder already exists and it is not empty
- [Error] if `/proc/sys/net/bridge/bridge-nf-call-iptables` file does not exist/does not contain 1
- [Error] if advertise address is ipv6 and `/proc/sys/net/bridge/bridge-nf-call-ip6tables` does not exist/does not contain 1.
- [Error] if swap is on
- [Error] if `ip`, `iptables`, `mount`, `nsenter` commands are not present in the command path
- [warning] if `ebtables`, `ethtool`, `socat`, `tc`, `touch`, `crictl` commands are not present in the command path
- [warning] if extra arg flags for API server, controller manager, scheduler contains some invalid options
- [warning] if connection to https://API.AdvertiseAddress:API.BindPort goes through proxy
- [warning] if connection to services subnet goes through proxy (only first address checked)
- [warning] if connection to Pods subnet goes through proxy (only first address checked)
- If external etcd is provided:
    - [Error] if etcd version less than 3.0.14
    - [Error] if etcd certificates or keys are specified, but not provided
- If external etcd is NOT provided (and thus local etcd will be installed):
    - [Error] if ports 2379 is used
    - [Error] if Etcd.DataDir folder already exists and it is not empty
- If authorization mode is ABAC:
    - [Error] if abac_policy.json does not exist
- If authorization mode is WebHook
    - [Error] if webhook_authz.conf does not exist

Please note that:

1. Preflight checks can be invoked individually with the `kubeadm init phase preflight` command

Generate the necessary certificates

Kubeadm generates certificate and private key pairs for different purposes:

- A self signed certificate authority for the Kubernetes cluster saved into `ca.crt` file and `ca.key` private key file

- A serving certificate for the API server, generated using `ca.crt` as the CA, and saved into `apiserver.crt` file with its private key `apiserver.key`. This certificate should contain following alternative names:
  - The Kubernetes service's internal clusterIP (the first address in the services CIDR, e.g. `10.96.0.1` if service subnet is `10.96.0.0/12`)
  - Kubernetes DNS names, e.g. `kubernetes.default.svc.cluster.local` if `--service-dns-domain` flag value is `cluster.local`, plus default DNS names `kubernetes.default.svc`, `kubernetes.default`, `kubernetes`
  - The node-name
  - The `--apiserver-advertise-address`
  - Additional alternative names specified by the user
- A client certificate for the API server to connect to the kubelets securely, generated using `ca.crt` as the CA and saved into `apiserver-kubelet-client.crt` file with its private key `apiserver-kubelet-client.key`. This certificate should be in the `system:masters` organization
- A private key for signing ServiceAccount Tokens saved into `sa.key` file along with its public key `sa.pub`
- A certificate authority for the front proxy saved into `front-proxy-ca.crt` file with its key `front-proxy-ca.key`
- A client cert for the front proxy client, generate using `front-proxy-ca.crt` as the CA and saved into `front-proxy-client.crt` file with its private key`front-proxy-client.key`

Certificates are stored by default in `/etc/kubernetes/pki`, but this directory is configurable using the `--cert-dir` flag.

Please note that:

1. If a given certificate and private key pair both exist, and its content is evaluated compliant with the above specs, the existing files will be used and the generation phase for the given certificate skipped. This means the user can, for example, copy an existing CA to `/etc/kubernetes/pki/ca.{crt,key}`, and then kubeadm will use those files for signing the rest of the certs. See also [using custom certificates]
2. Only for the CA, it is possible to provide the `ca.crt` file but not the `ca.key` file, if all other certificates and kubeconfig files already are in place kubeadm recognize this condition and activates the ExternalCA , which also implies the `csrsigner`controller in controller-manager won't be started
3. If kubeadm is running in [ExternalCA mode]; all the certificates must be provided by the user, because kubeadm cannot generate them by itself
4. In case of kubeadm is executed in the `--dry-run` mode, certificates files are written in a temporary folder
5. Certificate generation can be invoked individually with the [`kubeadm init phase certs all`] command

Generate kubeconfig files for control plane components

Kubeadm kubeconfig files with identities for control plane components:

- A kubeconfig file for kubelet to use, `/etc/kubernetes/kubelet.conf`; inside this file is embedded a client certificate with kubelet identity. This client cert should:
  - Be in the `system:nodes` organization, as required by the [Node Authorization] module
  - Have the CN `system:node:<hostname-lowercased>`

- A kubeconfig file for controller-manager, `/etc/kubernetes/controller-manager.conf`; inside this file is embedded a client certificate with controller-manager identity. This client cert should have the CN `system:kube-controller-manager`, as defined by default [RBAC core components roles]
- A kubeconfig file for scheduler, `/etc/kubernetes/scheduler.conf`; inside this file is embedded a client certificate with scheduler identity. This client cert should have the CN `system:kube-scheduler`, as defined by default [RBAC core components roles]

Additionally, a kubeconfig file for kubeadm to use itself and the admin is generated and save into the `/etc/kubernetes/admin.conf` file. The "admin" here is defined the actual person(s) that is administering the cluster and want to have full control (**root**) over the cluster. The embedded client certificate for admin should: - Be in the `system:masters` organization, as defined by default [RBAC user facing role bindings] - Include a CN, but that can be anything. Kubeadm uses the `kubernetes-admin` CN

Please note that:

1. `ca.crt` certificate is embedded in all the kubeconfig files.
2. If a given kubeconfig file exists, and its content is evaluated compliant with the above specs, the existing file will be used and the generation phase for the given kubeconfig skipped
3. If kubeadm is running in [ExternalCA mode], all the required kubeconfig must be provided by the user as well, because kubeadm cannot generate any of them by itself
4. In case of kubeadm is executed in the `--dry-run` mode, kubeconfig files are written in a temporary folder
5. Kubeconfig files generation can be invoked individually with the [`kubeadm init phase kubeconfig all`] command

Generate static Pod manifests for control plane components

Kubeadm writes static Pod manifest files for control plane components to `/etc/kubernetes/manifests`; the kubelet watches this directory for Pods to create on startup.

Static Pod manifest share a set of common properties:

- All static Pods are deployed on `kube-system` namespace
- All static Pods gets `tier:control-plane` and `component:{component-name}` labels
- All static Pods gets `scheduler.alpha.kubernetes.io/critical-pod` annotation (this will be moved over to the proper solution of using Pod Priority and Preemption when ready)
- `hostNetwork: true` is set on all static Pods to allow control plane startup before a network is configured; as a consequence:
  - The `address` that the controller-manager and the scheduler use to refer the API server is `127.0.0.1`
  - If using a local etcd server, `etcd-servers` address will be set to `127.0.0.1:2379`
- Leader election is enabled for both the controller-manager and the scheduler
- Controller-manager and the scheduler will reference kubeconfig files with their respective, unique identities
- All static Pods gets any extra flags specified by the user as described in [passing custom arguments to control plane components]
- All static Pods gets any extra Volumes specified by the user (Host path)

Please note that:

1. All the images, for the `--kubernetes-version`/current architecture, will be pulled from `k8s.gcr.io`; In case an alternative image repository or CI image repository is specified this one will be used; In case a specific container image should be used for all control plane components, this one will be used. see using custom images for more details
2. In case of kubeadm is executed in the `--dry-run` mode, static Pods files are written in a temporary folder
3. Static Pod manifest generation for master components can be invoked individually with the `kubeadm init phase control-plane all` command

API server

The static Pod manifest for the API server is affected by following parameters provided by the users:

- The `apiserver-advertise-address` and `apiserver-bind-port` to bind to; if not provided, those value defaults to the IP address of the default network interface on the machine and port 6443
- The `service-cluster-ip-range` to use for services
- If an external etcd server is specified, the `etcd-servers` address and related TLS settings (`etcd-cafile`, `etcd-certfile`, `etcd-keyfile`); if an external etcd server is not be provided, a local etcd will be used (via host network)
- If a cloud provider is specified, the corresponding `--cloud-provider` is configured, together with the `--cloud-config` path if such file exists (this is experimental, alpha and will be removed in a future version)

Other API server flags that are set unconditionally are:

- `--insecure-port=0` to avoid insecure connections to the api server
- `--enable-bootstrap-token-auth=true` to enable the `BootstrapTokenAuthenticator` authentication module. See TLS Bootstrapping for more details
- `--allow-privileged` to `true` (required e.g. by kube proxy)
- `--requestheader-client-ca-file` to `front-proxy-ca.crt`
- `--enable-admission-plugins` to:
    - `NamespaceLifecycle` e.g. to avoid deletion of system reserved namespaces
    - `LimitRanger` and `ResourceQuota` to enforce limits on namespaces
    - `ServiceAccount` to enforce service account automation
    - `PersistentVolumeLabel` attaches region or zone labels to PersistentVolumes as defined by the cloud provider (This admission controller is deprecated and will be removed in a future version. It is not deployed by kubeadm by default with v1.9 onwards when not explicitly opting into using `gce` or `aws` as cloud providers)
    - `DefaultStorageClass` to enforce default storage class on `PersistentVolumeClaim` objects
    - `DefaultTolerationSeconds`
    - `NodeRestriction` to limit what a kubelet can modify (e.g. only pods on this node)
- `--kubelet-preferred-address-types` to `InternalIP,ExternalIP,Hostname;` this makes `kubectl logs` and other API server-kubelet communication work in environments where the hostnames of the nodes aren't resolvable
- Flags for using certificates generated in previous steps:
    - `--client-ca-file` to `ca.crt`
    - `--tls-cert-file` to `apiserver.crt`

    - `--tls-private-key-file` to `apiserver.key`
    - `--kubelet-client-certificate` to `apiserver-kubelet-client.crt`
    - `--kubelet-client-key` to `apiserver-kubelet-client.key`
    - `--service-account-key-file` to `sa.pub`
    - `--requestheader-client-ca-file` to`front-proxy-ca.crt`
    - `--proxy-client-cert-file` to `front-proxy-client.crt`
    - `--proxy-client-key-file` to `front-proxy-client.key`
- Other flags for securing the front proxy (API Aggregation) communications:
    - `--requestheader-username-headers=X-Remote-User`
    - `--requestheader-group-headers=X-Remote-Group`
    - `--requestheader-extra-headers-prefix=X-Remote-Extra-`
    - `--requestheader-allowed-names=front-proxy-client`

Controller manager

The static Pod manifest for the API server is affected by following parameters provided by the users:

- If kubeadm is invoked specifying a `--pod-network-cidr`, the subnet manager feature required for some CNI network plugins is enabled by setting:
    - `--allocate-node-cidrs=true`
    - `--cluster-cidr` and `--node-cidr-mask-size` flags according to the given CIDR
    - If a cloud provider is specified, the corresponding `--cloud-provider` is specified, together with the `--cloud-config` path if such configuration file exists (this is experimental, alpha and will be removed in a future version)

Other flags that are set unconditionally are:

- `--controllers` enabling all the default controllers plus `BootstrapSigner` and `TokenCleaner` controllers for TLS bootstrap. See TLS Bootstrapping for more details
- `--use-service-account-credentials` to `true`
- Flags for using certificates generated in previous steps:
    - `--root-ca-file` to `ca.crt`
    - `--cluster-signing-cert-file` to `ca.crt`, if External CA mode is disabled, otherwise to `""`
    - `--cluster-signing-key-file` to `ca.key`, if External CA mode is disabled, otherwise to `""`
    - `--service-account-private-key-file` to `sa.key`

Scheduler

The static Pod manifest for the scheduler is not affected by parameters provided by the users.

Generate static Pod manifest for local etcd

If the user specified an external etcd this step will be skipped, otherwise kubeadm generates a static Pod manifest file for creating a local etcd instance running in a Pod with following attributes:

- listen on `localhost:2379` and use `HostNetwork=true`
- make a `hostPath` mount out from the `dataDir` to the host's filesystem

- Any extra flags specified by the user

Please note that:

1. The etcd image will be pulled from `k8s.gcr.io`. In case an alternative image repository is specified this one will be used; In case an alternative image name is specified, this one will be used. see using custom images for more details
2. in case of kubeadm is executed in the `--dry-run` mode, the etcd static Pod manifest is written in a temporary folder
3. Static Pod manifest generation for local etcd can be invoked individually with the `kubeadm init phase etcd local` command

Optional Dynamic Kublet Configuration

To use this functionality call `kubeadm alpha kubelet config enable-dynamic`. It writes the kubelet init configuration into `/var/lib/kubelet/config/init/kubelet` file.

The init configuration is used for starting the kubelet on this specific node, providing an alternative for the kubelet drop-in file; such configuration will be replaced by the kubelet base configuration as described in following steps. See set Kubelet parameters via a config file for additional info.

Please note that:

1. To make dynamic kubelet configuration work, flag `--dynamic-config-dir=/var/lib/kubelet/config/dynamic` should be specified in `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf`
2. The kubelet configuration can be changed by passing a `KubeletConfiguration` object to `kubeadm init` or `kubeadm join` by using a configuration file `--config some-file.yaml`. The `KubeletConfiguration` object can be separated from other objects such as `InitConfiguration` using the `---` separator. For more details have a look at the `kubeadm config print-default` command.

Wait for the control plane to come up

This is a critical moment in time for kubeadm clusters. kubeadm waits until `localhost:6443/healthz` returns `ok`, however in order to detect deadlock conditions, kubeadm fails fast if `localhost:10255/healthz` (kubelet liveness) or `localhost:10255/healthz/syncloop` (kubelet readiness) don't return `ok`, respectively after 40 and 60 second.

kubeadm relies on the kubelet to pull the control plane images and run them properly as static Pods. After the control plane is up, kubeadm completes the tasks described in following paragraphs.

(optional and alpha in v1.9) Write base kubelet configuration

If kubeadm is invoked with `--feature-gates=DynamicKubeletConfig`:

1. Write the kubelet base configuration into the `kubelet-base-config-v1.9` ConfigMap in the `kube-system` namespace
2. Creates RBAC rules for granting read access to that ConfigMap to all bootstrap tokens and all kubelet instances (that is `system:bootstrappers:kubeadm:default-node-token` and `system:nodes` groups)

3. Enable the dynamic kubelet configuration feature for the initial control-plane node by pointing `Node.spec.configSource` to the newly-created ConfigMap

Save the kubeadm ClusterConfiguration in a ConfigMap for later reference

kubeadm saves the configuration passed to `kubeadm init`, either via flags or the config file, in a ConfigMap named `kubeadm-config` under `kube-system` namespace.

This will ensure that kubeadm actions executed in future (e.g `kubeadm upgrade`) will be able to determine the actual/current cluster state and make new decisions based on that data.

Please note that:

1. Before uploading, sensitive information like e.g. the token is stripped from the configuration
2. Upload of master configuration can be invoked individually with the [`kubeadm init phase upload-config`]() command
3. If you initialized your cluster using kubeadm v1.7.x or lower, you must create manually the master configuration ConfigMap before `kubeadm upgrade` to v1.8 . In order to facilitate this task, the [`kubeadm config upload (from-flags|from-file)`]() was implemented

Mark master

As soon as the control plane is available, kubeadm executes following actions:

- Label the master with `node-role.kubernetes.io/master=""`
- Taints the master with `node-role.kubernetes.io/master:NoSchedule`

Please note that:

1. Mark control-plane phase phase can be invoked individually with the [`kubeadm init phase mark-control-plane`]() command

Configure TLS-Bootstrapping for node joining

Kubeadm uses [Authenticating with Bootstrap Tokens]() for joining new nodes to an existing cluster; for more details see also [design proposal]().

`kubeadm init` ensures that everything is properly configured for this process, and this includes following steps as well as setting API server and controller flags as already described in previous paragraphs. Please note that:

1. TLS bootstrapping for nodes can be configured with the [`kubeadm init phase bootstrap-token`]() command, executing all the configuration steps described in following paragraphs; alternatively, each step can be invoked individually

Create a bootstrap token

`kubeadm init` create a first bootstrap token, either generated automatically or provided by the user with the `--token` flag; as documented in bootstrap token specification, token should be saved as secrets with name `bootstrap-token-<token-id>` under `kube-system` namespace. Please note that:

1. The default token created by `kubeadm init` will be used to validate temporary user during TLS bootstrap process; those users will be member of `system:bootstrappers:kubeadm:default-node-token` group
2. The token has a limited validity, default 24 hours (the interval may be changed with the `â€”token-ttl` flag)
3. Additional tokens can be created with the [`kubeadm token`]() command, that provide as well other useful functions for token management

Allow joining nodes to call CSR API

Kubeadm ensures that users in `system:bootstrappers:kubeadm:default-node-token` group are able to access the certificate signing API.

This is implemented by creating a ClusterRoleBinding named `kubeadm:kubelet-bootstrap` between the group above and the default RBAC role `system:node-bootstrapper`.

Setup auto approval for new bootstrap tokens

Kubeadm ensures that the Bootstrap Token will get its CSR request automatically approved by the csrapprover controller.

This is implemented by creating ClusterRoleBinding named `kubeadm:node-autoapprove-bootstrap` between the `system:bootstrappers:kubeadm:default-node-token` group and the default role `system:certificates.k8s.io:certificatesigningrequests:nodeclient`.

The role `system:certificates.k8s.io:certificatesigningrequests:nodeclient` should be created as well, granting POST permission to `/apis/certificates.k8s.io/certificatesigningrequests/nodeclient`.

Setup nodes certificate rotation with auto approval

Kubeadm ensures that certificate rotation is enabled for nodes, and that new certificate request for nodes will get its CSR request automatically approved by the csrapprover controller.

This is implemented by creating ClusterRoleBinding named `kubeadm:node-autoapprove-certificate-rotation` between the `system:nodes` group and the default role `system:certificates.k8s.io:certificatesigningrequests:selfnodeclient`.

Create the public cluster-info ConfigMap

This phase creates the `cluster-info` ConfigMap in the `kube-public` namespace.

Additionally it is created a role and a RoleBinding granting access to the ConfigMap for unauthenticated users (i.e. users in RBAC group `system:unauthenticated`)

Please note that:

1. The access to the `cluster-info` ConfigMap *is not* rate-limited. This may or may not be a problem if you expose your master to the internet; worst-case scenario here is a DoS attack where an attacker uses all the in-flight requests the kube-apiserver can handle to serving the `cluster-info` ConfigMap.

Install addons

Kubeadm installs the internal DNS server and the kube-proxy addon components via the API server. Please note that:

1. This phase can be invoked individually with the [`kubeadm init phase addon all`] command.

proxy

A ServiceAccount for `kube-proxy` is created in the `kube-system` namespace; then kube-proxy is deployed as a DaemonSet:

- The credentials (`ca.crt` and `token`) to the master come from the ServiceAccount
- The location of the master comes from a ConfigMap
- The `kube-proxy` ServiceAccount is bound to the privileges in the `system:node-proxier` ClusterRole

DNS

Note that:

- The CoreDNS service is named `kube-dns`. This is done to prevent any interruption in service when the user is switching the cluster DNS from kube-dns to CoreDNS or vice-versa
- In Kubernetes version 1.10 and earlier, you must enable CoreDNS with `--feature-gates=CoreDNS=true`
- In Kubernetes version 1.11 and 1.12, CoreDNS is the default DNS server and you must invoke kubeadm with `--feature-gates=CoreDNS=false` to install kube-dns instead
- In Kubernetes version 1.13 and later, the `CoreDNS` feature gate is no longer available and kube-dns can be installed using the `--config` method described here

A ServiceAccount for CoreDNS/kube-dns is created in the `kube-system` namespace.

Deploy the `kube-dns` Deployment and Service:

- It's the upstream CoreDNS deployment relatively unmodified
- The `kube-dns` ServiceAccount is bound to the privileges in the `system:kube-dns` ClusterRole

kubeadm join phases internal design

Similarly to `kubeadm init`, also `kubeadm join` internal workflow consists of a sequence of atomic work tasks to perform.

This is split into discovery (having the Node trust the Kubernetes Master) and TLS bootstrap (having the Kubernetes Master trust the Node).

see [Authenticating with Bootstrap Tokens] or the corresponding [design proposal].

Preflight checks

`kubeadm` executes a set of preflight checks before starting the join, with the aim to verify preconditions and avoid common cluster startup problems.

Please note that:

1. `kubeadm join` preflight checks are basically a subset `kubeadm init` preflight checks
2. Starting from 1.9, kubeadm provides better support for CRI-generic functionality; in that case, docker specific controls are skipped or replaced by similar controls for crictl.
3. Starting from 1.9, kubeadm provides support for joining nodes running on Windows; in that case, linux specific controls are skipped.
4. In any case the user can skip specific preflight checks (or eventually all preflight checks) with the `--ignore-preflight-errors` option.

Discovery cluster-info

There are 2 main schemes for discovery. The first is to use a shared token along with the IP address of the API server. The second is to provide a file (that is a subset of the standard kubeconfig file).

Shared token discovery

If `kubeadm join` is invoked with `--discovery-token`, token discovery is used; in this case the node basically retrieves the cluster CA certificates from the `cluster-info` ConfigMap in the `kube-public` namespace.

In order to prevent "man in the middle" attacks, several steps are taken:

- First, the CA certificate is retrieved via insecure connection (this is possible because `kubeadm init` granted access to `cluster-info` users for `system:unauthenticated`)
- Then the CA certificate goes trough following validation steps:
  - Basic validation: using the token ID against a JWT signature
  - Pub key validation: using provided `--discovery-token-ca-cert-hash`. This value is available in the output of `kubeadm init` or can be calculated using standard tools (the hash is calculated over the bytes of the Subject Public Key Info (SPKI) object as in RFC7469). The `--discovery-token-ca-cert-hash flag` may be repeated multiple times to allow more than one public key.
  - As a additional validation, the CA certificate is retrieved via secure connection and then compared with the CA retrieved initially

Please note that:

1. Pub key validation can be skipped passing `--discovery-token-unsafe-skip-ca-verification` flag; This weakens the kubeadm security model since others can potentially impersonate the Kubernetes Master.

File/https discovery

If `kubeadm join` is invoked with `--discovery-file`, file discovery is used; this file can be a local file or downloaded via an HTTPS URL; in case of HTTPS, the host installed CA bundle is used to verify the connection.

With file discovery, the cluster CA certificates is provided into the file itself; in fact, the discovery file is a kubeconfig file with only `server` and `certificate-authority-data` attributes set, as described in [`kubeadm join`]() reference doc; when the connection with the cluster is established, kubeadm try to access the `cluster-info` ConfigMap, and if available, uses it.

TLS Bootstrap

Once the cluster info are known, the file `bootstrap-kubelet.conf` is written, thus allowing kubelet to do TLS Bootstrapping (conversely until v.1.7 TLS bootstrapping were managed by kubeadm).

The TLS bootstrap mechanism uses the shared token to temporarily authenticate with the Kubernetes Master to submit a certificate signing request (CSR) for a locally created key pair.

The request is then automatically approved and the operation completes saving `ca.crt` file and `kubelet.conf` file to be used by kubelet for joining the cluster, while`bootstrap-kubelet.conf` is deleted.

Please note that:

- The temporary authentication is validated against the token saved during the `kubeadm init` process (or with additional tokens created with `kubeadm token`)
- The temporary authentication resolve to a user member of `system:bootstrappers:kubeadm:default-node-token` group which was granted access to CSR api during the `kubeadm init` process
- The automatic CSR approval is managed by the csrapprover controller, according with configuration done the `kubeadm init` process

(optional and alpha in v1.9) Write init kubelet configuration

If kubeadm is invoked with `--feature-gates=DynamicKubeletConfig`:

1. Read the kubelet base configuration from the `kubelet-base-config-v1.9` ConfigMap in the `kube-system` namespace using the Bootstrap Token credentials, and write it to disk as kubelet init configuration file `/var/lib/kubelet/config/init/kubelet`
2. As soon as kubelet starts with the Node's own credential (`/etc/kubernetes/kubelet.conf`), update current node configuration specifying that the source for the node/kubelet configuration is the above ConfigMap.

Please note that:

1. To make dynamic kubelet configuration work, flag `--dynamic-config-dir=/var/lib/kubelet/config/dynamic` should be specified in `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf`

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on May 30, 2019 at 5:10 AM PST by Remove initializers feature gate (#14602) (Page History)

Edit This Page

- - kubelet
    - Synopsis
      - Pod Lifecycle Event Generator (PLEG)
    - Options

kubelet

Synopsis

The kubelet is the primary "node agent" that runs on each node. The kubelet works in terms of a PodSpec. A PodSpec is a YAML or JSON object that describes a pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms (primarily through the apiserver) and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

Other than from a PodSpec from the apiserver, there are three ways that a container manifest can be provided to the Kubelet.

File: Path passed as a flag on the command line. Files under this path will be monitored periodically for updates. The monitoring period is 20s by default and is configurable via a flag.

HTTP endpoint: HTTP endpoint passed as a parameter on the command line. This endpoint is checked every 20 seconds (also configurable with a flag).

HTTP server: The kubelet can also listen for HTTP and respond to a simple API (underspec'd currently) to submit a new manifest.

Pod Lifecycle Event Generator (PLEG)

The Pod Lifecycle Event Generator is a function of the kubelet that creates a list of the states for all containers and pods then compares it to the previous states of the containers and pods in a process called Relisting. This allows the PLEG to know which pods and containers need to be synced. In versions prior to 1.2, this was accomplished by polling and was CPU intensive. By changing to this method, this significantly reduced resource utilization allowing for better container density.

```
kubelet [flags]
```

Options

--address 0.0.0.0	The IP address for the Kubelet to serve on (set to 0.0.0.0 for all IPv4 interfaces and `::` for all IPv6 interfaces) (default 0.0.0.0)
--alsologtostderr	log to standard error as well as files
--anonymous-auth	Enables anonymous requests to the Kubelet server. Requests that are not rejected by another authentication method are treated as anonymous requests. Anonymous requests have a username of system:anonymous, and a group name of system:unauthenticated. (default true)
--application-metrics-count-limit int	Max number of application metrics to store (per container) (default 100)
--authentication-token-webhook	Use the TokenReview API to determine authentication for bearer tokens.
--authentication-token-webhook-cache-ttl duration	The duration to cache responses from the webhook token authenticator. (default 2m0s)
--authorization-mode string	Authorization mode for Kubelet server. Valid options are AlwaysAllow or Webhook. Webhook mode uses the SubjectAccessReview API to determine authorization. (default "AlwaysAllow")
--authorization-webhook-cache-authorized-ttl duration	The duration to cache 'authorized' responses from the webhook authorizer. (default 5m0s)
--authorization-webhook-cache-unauthorized-ttl duration	The duration to cache 'unauthorized' responses from the webhook authorizer. (default 30s)
--azure-container-registry-config string	Path to the file container Azure container registry configuration information.
--boot-id-file string	Comma-separated list of files to check for boot-id. Use the first one that exists. (default "/proc/sys/kernel/random/boot_id")
--bootstrap-checkpoint-path string	Path to the directory where the checkpoints are stored
--bootstrap-kubeconfig string	Path to a kubeconfig file that will be used to get client certificate for kubelet. If the file specified by --kubeconfig does not exist, the bootstrap kubeconfig is used to request a client certificate from the API server. On success, a kubeconfig file referencing the generated client certificate and key is written to the path specified by --kubeconfig. The client certificate and key file will be stored in the directory pointed by --cert-dir.
--cert-dir string	The directory where the TLS certs are located. If --tls-cert-file and --tls-private-key-file are provided, this flag will be ignored. (default "/var/lib/kubelet/pki")

--cgroup-driver string	
	Driver that the kubelet uses to manipulate cgroups on the host.
--cgroup-root string	
	Optional root cgroup to use for pods. This is handled by the container runtime on a best effort basis. Default: '', which means use the container runtime default.
--cgroups-per-qos	
	Enable creation of QoS cgroup hierarchy, if true top level QoS and pod cgroups are created. (default true)
--chaos-chance float	
	If > 0.0, introduce random client errors and latency. Intended for testing.
--client-ca-file string	
	If set, any request presenting a client certificate signed by one of the authorities in the client-ca-file is authenticated with an identity corresponding to the CommonName of the client certificate.
--cloud-config string	
	The path to the cloud provider configuration file.
--cloud-provider string	
	The provider for cloud services. Specify empty string for running with no cloud provider.
--cloud-provider-gce-lb-src-cidrs cidrs	
	CIDRs opened in GCE firewall for LB traffic proxy & health checks (default 130.211.0.0/22,35.191.0.0/16,209.85.152.0/22,209.85.204.0/22)
--cluster-dns stringSlice	
	Comma-separated list of DNS server IP address.
--cluster-domain string	
	Domain for this cluster.
--cni-bin-dir string	
	The full path of the directory in which to search for CNI plugin binaries. Default: /opt/cni/bin
--cni-conf-dir string	
	The full path of the directory in which to search for CNI config files. Default: /etc/cni/net.d
--container-hints string	
	location of the container hints file (default "/etc/cadvisor/container_hints.json")
--container-runtime string	
	The container runtime to use. Possible values: 'docker', 'remote', 'rkt(deprecated)'. (default "docker")
--container-runtime-endpoint string	
	[Experimental] The endpoint of remote runtime service. Currently unix socket is supported on Linux, and tcp is supported on windows.
--containerd string	
	containerd endpoint (default "unix:///var/run/containerd.sock")
--containerized	
	Experimental support for running kubelet in a container.
--contention-profiling	
	Enable lock contention profiling, if profiling is enabled
--cpu-cfs-quota	

	Enable CPU CFS quota enforcement for containers that specify CPU limits (default true)
--cpu-manager-policy string	
	CPU Manager policy to use. Possible values: 'none', 'static'. (default "none")
--cpu-manager-reconcile-period NodeStatusUpdateFrequency	
	CPU Manager reconciliation period. Examples: '10s', or '1m'. If not supplied, defaults to NodeStatusUpdateFrequency (default 10s)
--docker string	
	docker endpoint (default "unix:///var/run/docker.sock")
--docker-disable-shared-pid	
	The Container Runtime Interface (CRI) defaults to using a shared PID namespace for containers in a pod when running with Docker 1.13.1 or higher. Setting this flag reverts to the previous behavior of isolated PID namespaces. This ability will be removed in a future Kubernetes release. (default true)
--docker-endpoint string	
	Use this for the docker endpoint to communicate with (default "unix:///var/run/docker.sock")
--docker-env-metadata-whitelist string	
	a comma-separated list of environment variable keys that needs to be collected for docker containers
--docker-only	
	Only report docker containers in addition to root stats
--docker-root string	
	DEPRECATED: docker root is read from docker info (this is a fallback, default: /var/lib/docker) (default "/var/lib/docker")
--docker-tls	
	use TLS to connect to docker
--docker-tls-ca string	
	path to trusted CA (default "ca.pem")
--docker-tls-cert string	
	path to client certificate (default "cert.pem")
--docker-tls-key string	
	path to private key (default "key.pem")
--dynamic-config-dir string	
	The Kubelet will use this directory for checkpointing downloaded configurations and tracking configuration health. The Kubelet will create this directory if it does not already exist. The path may be absolute or relative; relative paths start at the Kubelet's current working directory. Providing this flag enables dynamic Kubelet configuration. Presently, you must also enable the DynamicKubeletConfig feature gate to pass this flag.
--enable-controller-attach-detach	
	Enables the Attach/Detach controller to manage attachment/detachment of volumes scheduled to this node, and disables kubelet from executing any attach/detach operations (default true)
--enable-debugging-handlers	
	Enables server endpoints for log collection and local running of containers and commands (default true)
--enable-load-reader	

	Whether to enable cpu load reader
--enable-server	
	Enable the Kubelet's server (default true)
--enforce-node-allocatable stringSlice	
	A comma separated list of levels of node allocatable enforcement to be enforced by kubelet. Acceptable options are 'pods', 'system-reserved' & 'kube-reserved'. If the latter two options are specified, '--system-reserved-cgroup' & '--kube-reserved-cgroup' must also be set respectively. See /docs/tasks/administer-cluster/reserve-compute-resources/ for more details. (default [pods])
--event-burst int32	
	Maximum size of a bursty event records, temporarily allows event records to burst to this number, while still not exceeding event-qps. Only used if --event-qps > 0 (default 10)
--event-qps int32	
	If > 0, limit event creations per second to this value. If 0, unlimited. (default 5)
--event-storage-age-limit string	
	Max length of time for which to store events (per type). Value is a comma separated list of key values, where the keys are event types (e.g.: creation, oom) or "default" and the value is a duration. Default is applied to all non-specified event types (default "default=0")
--event-storage-event-limit string	
	Max number of events to store (per type). Value is a comma separated list of key values, where the keys are event types (e.g.: creation, oom) or "default" and the value is an integer. Default is applied to all non-specified event types (default "default=0")
--eviction-hard mapStringString	
	A set of eviction thresholds (e.g. memory.available<1Gi) that if met would trigger a pod eviction. (default imagefs.available<15%,memory.available<100Mi,nodefs.available<10%,nodefs.inodesFree<5%)
--eviction-max-pod-grace-period int32	
	Maximum allowed grace period (in seconds) to use when terminating pods in response to a soft eviction threshold being met.
--eviction-minimum-reclaim mapStringString	
	A set of minimum reclaims (e.g. imagefs.available=2Gi) that describes the minimum amount of resource the kubelet will reclaim when performing a pod eviction if that resource is under pressure.
--eviction-pressure-transition-period duration	
	Duration for which the kubelet has to wait before transitioning out of an eviction pressure condition. (default 5m0s)
--eviction-soft mapStringString	
	A set of eviction thresholds (e.g. memory.available<1.5Gi) that if met over a corresponding grace period would trigger a pod eviction.
--eviction-soft-grace-period mapStringString	
	A set of eviction grace periods (e.g. memory.available=1m30s) that correspond to how long a soft eviction threshold must hold before triggering a pod eviction.
--exit-on-lock-contention	
	Whether kubelet should exit upon lock-file contention.

--experimental-allocatable-ignore-eviction	
	When set to 'true', Hard Eviction Thresholds will be ignored while calculating Node Allocatable. See /docs/tasks/administer-cluster/reserve-compute-resources/ for more details. [default=false]
--experimental-allowed-unsafe-sysctls stringSlice	
	Comma-separated whitelist of unsafe sysctls or unsafe sysctl patterns (ending in *). Use these at your own risk.
--experimental-bootstrap-kubeconfig string	
	deprecated: use --bootstrap-kubeconfig
--experimental-check-node-capabilities-before-mount	
	[Experimental] if set true, the kubelet will check the underlying node for required components (binaries, etc.) before performing the mount
--experimental-kernel-memcg-notification	
	If enabled, the kubelet will integrate with the kernel memcg notification to determine if memory eviction thresholds are crossed rather than polling.
--experimental-mounter-path string	
	[Experimental] Path of mounter binary. Leave empty to use the default mount.
--experimental-qos-reserved mapStringString	
	A set of ResourceName=Percentage (e.g. memory=50%) pairs that describe how pod resource requests are reserved at the QoS level. Currently only memory is supported. [default=none]
--fail-swap-on	
	Makes the Kubelet fail to start if swap is enabled on the node.
--feature-gates mapStringBool	

<table>
<tr><td></td><td>A set of key=value pairs that describe feature gates for alpha/experimental features. Options are:
APIListChunking=true|false (BETA - default=true)
APIResponseCompression=true|false (ALPHA - default=false)
Accelerators=true|false
AdvancedAuditing=true|false (BETA - default=true)
AllAlpha=true|false (ALPHA - default=false)
AllowExtTrafficLocalEndpoints=true|false
AppArmor=true|false (BETA - default=true)
BlockVolume=true|false (ALPHA - default=false)
CPUManager=true|false (BETA - default=true)
CSIPersistentVolume=true|false (ALPHA - default=false)
CustomPodDNS=true|false (ALPHA - default=false)
CustomResourceValidation=true|false (BETA - default=true)
DebugContainers=true|false
DevicePlugins=true|false (ALPHA - default=false)
DynamicKubeletConfig=true|false (ALPHA - default=false)
EnableEquivalenceClassCache=true|false (ALPHA - default=false)
ExpandPersistentVolumes=true|false (ALPHA - default=false)
ExperimentalCriticalPodAnnotation=true|false (ALPHA - default=false)
ExperimentalHostUserNamespaceDefaulting=true|false (BETA - default=false)
HugePages=true|false (ALPHA - default=false)
Initializers=true|false (ALPHA - default=false)
KubeletConfigFile=true|false (ALPHA - default=false)
LocalStorageCapacityIsolation=true|false (ALPHA - default=false)
LocalStorageCapacityIsolationFSQuotaMonitoring=true|false (ALPHA - default=false)
MountContainers=true|false (ALPHA - default=false)
MountPropagation=true|false (ALPHA - default=false)
PVCProtection=true|false (ALPHA - default=false)
PersistentLocalVolumes=true|false (ALPHA - default=false)
PodPriority=true|false (ALPHA - default=false)
ReadOnlyAPIDataVolumes=true|false
ResourceLimitsPriorityFunction=true|false (ALPHA - default=false)
RotateKubeletClientCertificate=true|false (BETA - default=true)
RotateKubeletServerCertificate=true|false (ALPHA - default=false)
ServiceNodeExclusion=true|false (ALPHA - default=false)
ServiceProxyAllowExternalIPs=true|false
StreamingProxyRedirects=true|false (BETA - default=true)
SupportIPVSProxyMode=true|false (ALPHA - default=false)
SupportNodePidsLimit=true|false (BETA - default=true)
TaintBasedEvictions=true|false (BETA - default=true)
TaintNodesByCondition=true|false (BETA - default=true)
VolumeScheduling=true|false (ALPHA - default=false)
VolumeSubpath=true|false</td></tr>
<tr><td colspan="2">--file-check-frequency duration</td></tr>
<tr><td></td><td>Duration between checking config files for new data (default 20s)</td></tr>
</table>

--global-housekeeping-interval duration	
	Interval between global housekeepings (default 1m0s)
--google-json-key string	
	The Google Cloud Platform Service Account JSON Key to use for authentication.
--hairpin-mode string	
	How should the kubelet setup hairpin NAT. This allows endpoints of a Service to loadbalance back to themselves if they should try to access their own Service. Valid values are "promiscuous-bridge", "hairpin-veth" and "none". (default "promiscuous-bridge")
--healthz-bind-address 0.0.0.0	
	The IP address for the healthz server to serve on (set to 0.0.0.0 for all IPv4 interfaces and \`::\` for all IPv6 interfaces) (default 127.0.0.1)
--healthz-port int32	
	The port of the localhost healthz endpoint (set to 0 to disable) (default 10248)
--host-ipc-sources stringSlice	
	Comma-separated list of sources from which the Kubelet allows pods to use the host ipc namespace. (default [*])
--host-network-sources stringSlice	
	Comma-separated list of sources from which the Kubelet allows pods to use of host network. (default [*])
--host-pid-sources stringSlice	
	Comma-separated list of sources from which the Kubelet allows pods to use the host pid namespace. (default [*])
--hostname-override string	
	If non-empty, will use this string as identification instead of the actual hostname.
--housekeeping-interval duration	
	Interval between container housekeepings (default 10s)
--http-check-frequency duration	
	Duration between checking http for new data (default 20s)
--image-gc-high-threshold int32	
	The percent of disk usage after which image garbage collection is always run. (default 85)
--image-gc-low-threshold int32	
	The percent of disk usage before which image garbage collection is never run. Lowest disk usage to garbage collect to. (default 80)
--image-pull-progress-deadline duration	
	If no pulling progress is made before this deadline, the image pulling will be cancelled. (default 1m0s)
--image-service-endpoint string	
	[Experimental] The endpoint of remote image service. If not specified, it will be the same with container-runtime-endpoint by default. Currently unix socket is supported on Linux, and tcp is supported on windows.
--init-config-dir string	

	The Kubelet will look in this directory for the init configuration. The path may be absolute or relative; relative paths start at the Kubelet's current working directory. Omit this argument to use the built-in default configuration values. Presently, you must also enable the KubeletConfigFile feature gate to pass this flag.
--iptables-drop-bit int32	
	The bit of the fwmark space to mark packets for dropping. Must be within the range [0, 31]. (default 15)
--iptables-masquerade-bit int32	
	The bit of the fwmark space to mark packets for SNAT. Must be within the range [0, 31]. Please match this parameter with corresponding parameter in kube-proxy. (default 14)
--kube-api-burst int32	
	Burst to use while talking with kubernetes apiserver (default 10)
--kube-api-content-type string	
	Content type of requests sent to apiserver. (default "application/vnd.kubernetes.protobuf")
--kube-api-qps int32	
	QPS to use while talking with kubernetes apiserver (default 5)
--kube-reserved mapStringString	
	A set of ResourceName=ResourceQuantity (e.g. cpu=200m,memory=500Mi,ephemeral-storage=1Gi,pid=1000) pairs that describe resources reserved for kubernetes system components. Currently cpu, memory, pid, and local ephemeral storage for root file system are supported. See http://kubernetes.io/docs/user-guide/compute-resources for more detail. [default=none]
--kube-reserved-cgroup string	
	Absolute name of the top level cgroup that is used to manage kubernetes components for which compute resources were reserved via '--kube-reserved' flag. Ex. '/kube-reserved'. [default='']
--kubeconfig string	
	Path to a kubeconfig file, specifying how to connect to the API server. Providing --kubeconfig enables API server mode, omitting --kubeconfig enables standalone mode.
--kubelet-cgroups string	
	Optional absolute name of cgroups to create and run the Kubelet in.
--lock-file string	
	The path to file for kubelet to use as a lock file.
--log-backtrace-at traceLocation	
	when logging hits line file:N, emit a stack trace (default :0)
--log-cadvisor-usage	
	Whether to log the usage of the cAdvisor container
--log-dir string	
	If non-empty, write log files in this directory
--log-flush-frequency duration	
	Maximum number of seconds between log flushes (default 5s)
--logtostderr	
	log to standard error instead of files (default true)
--machine-id-file string	

	Comma-separated list of files to check for machine-id. Use the first one that exists. (default "/etc/machine-id,/var/lib/dbus/machine-id")
--make-iptables-util-chains	
	If true, kubelet will ensure iptables utility rules are present on host. (default true)
--manifest-url string	
	URL for accessing the container manifest
--manifest-url-header --manifest-url-header 'a:hello,b:again,c:world' --manifest-url-header 'b:beautiful'	
	Comma-separated list of HTTP headers to use when accessing the manifest URL. Multiple headers with the same name will be added in the same order provided. This flag can be repeatedly invoked. For example: --manifest-url-header 'a:hello,b:again,c:world' --manifest-url-header 'b:beautiful'
--max-open-files int	
	Number of files that can be opened by Kubelet process. (default 1000000)
--max-pods int32	
	Number of Pods that can run on this Kubelet. (default 110)
--minimum-image-ttl-duration duration	
	Minimum age for an unused image before it is garbage collected.
--network-plugin string	
	The name of the network plugin to be invoked for various events in kubelet/pod lifecycle
--network-plugin-mtu int32	
	The MTU to be passed to the network plugin, to override the default. Set to 0 to use the default 1460 MTU.
--node-ip string	
	IP address of the node. If set, kubelet will use this IP address for the node
--node-labels mapStringString	
	Labels to add when registering the node in the cluster.
--node-status-update-frequency duration	
	Specifies how often kubelet posts node status to master. Note: be cautious when changing the constant, it must work with nodeMonitorGracePeriod in nodecontroller. (default 10s)
--oom-score-adj int32	
	The oom-score-adj value for kubelet process. Values must be within the range [-1000, 1000] (default -999)
--pod-cidr string	
	The CIDR to use for pod IP addresses, only used in standalone mode.
--pod-infra-container-image string	
	The image whose network/ipc namespaces containers in each pod will use. (default "k8s.gcr.io/pause:3.1")
--pod-manifest-path string	
	Path to the directory containing pod manifest files to run, or the path to a single pod manifest file. Files starting with dots will be ignored.
--pods-per-core int32	

	Number of Pods per core that can run on this Kubelet. The total number of Pods on this Kubelet cannot exceed max-pods, so max-pods will be used if this calculation results in a larger number of Pods allowed on the Kubelet. A value of 0 disables this limit.
--port int32	
	The port for the Kubelet to serve on. (default 10250)
--protect-kernel-defaults	
	Default kubelet behaviour for kernel tuning. If set, kubelet errors if any of kernel tunables is different than kubelet defaults.
--provider-id string	
	Unique identifier for identifying the node in a machine database, i.e cloudprovider
--read-only-port int32	
	The read-only port for the Kubelet to serve on with no authentication/authorization (set to 0 to disable) (default 10255)
--really-crash-for-testing	
	If true, when panics occur crash. Intended for testing.
--register-node	
	Register the node with the apiserver. If --kubeconfig is not provided, this flag is irrelevant, as the Kubelet won't have an apiserver to register with. Default=true. (default true)
--register-with-taints []api.Taint	
	Register the node with the given list of taints (comma separated "=:"). No-op if register-node is false.
--registry-burst int32	
	Maximum size of bursty pulls, temporarily allows pulls to burst to this number, while still not exceeding registry-qps. Only used if --registry-qps > 0 (default 10)
--registry-qps int32	
	If > 0, limit registry pull QPS to this value.
--resolv-conf string	
	Resolver configuration file used as the basis for the container DNS resolution configuration. (default "/etc/resolv.conf")
--root-dir string	
	Directory path for managing kubelet files (volume mounts,etc). (default "/var/lib/kubelet")
--rotate-certificates	
	Auto rotate the kubelet client certificates by requesting new certificates from the kube-apiserver when the certificate expiration approaches.
--rotate-server-certificates	
	Auto-request and rotate the kubelet serving certificates by requesting new certificates from the kube-apiserver when the certificate expiration approaches. Requires the RotateKubeletServerCertificate feature gate to be enabled, and approval of the submitted CertificateSigningRequest objects.
--runonce	
	If true, exit after spawning pods from local manifests or remote urls. Exclusive with --enable-server
--runtime-cgroups string	
	Optional absolute name of cgroups to create and run the runtime in.

--runtime-request-timeout duration	
	Timeout of all runtime requests except long running request - pull, logs, exec and attach. When timeout exceeded, kubelet will cancel the request, throw out an error and retry later. (default 2m0s)
--seccomp-profile-root string	
	Directory path for seccomp profiles. (default "/var/lib/kubelet/seccomp")
--serialize-image-pulls	
	Pull images one at a time. We recommend *not* changing the default value on nodes that run docker daemon with version < 1.9 or an Aufs storage backend. Issue #10959 has more details. (default true)
--stderrthreshold severity	
	logs at or above this threshold go to stderr (default 2)
--storage-driver-buffer-duration duration	
	Writes in the storage driver will be buffered for this duration, and committed to the non memory backends as a single transaction (default 1m0s)
--storage-driver-db string	
	database name (default "cadvisor")
--storage-driver-host string	
	database host:port (default "localhost:8086")
--storage-driver-password string	
	database password (default "root")
--storage-driver-secure	
	use secure connection with database
--storage-driver-table string	
	table name (default "stats")
--storage-driver-user string	
	database username (default "root")
--streaming-connection-idle-timeout duration	
	Maximum time a streaming connection can be idle before the connection is automatically closed. 0 indicates no timeout. Example: '5m' (default 4h0m0s)
--sync-frequency duration	
	Max period between synchronizing running containers and config (default 1m0s)
--system-cgroups /	
	Optional absolute name of cgroups in which to place all non-kernel processes that are not already inside a cgroup under /. Empty for no container. Rolling back the flag requires a reboot.
--system-reserved mapStringString	
	A set of ResourceName=ResourceQuantity (e.g. cpu=200m,memory=500Mi,ephemeral-storage=1Gi,pid=1000) pairs that describe resources reserved for non-kubernetes components. Currently only cpu, memory, and pid are supported. See http://kubernetes.io/docs/user-guide/compute-resources for more detail. [default=none]
--system-reserved-cgroup string	
	Absolute name of the top level cgroup that is used to manage non-kubernetes components for which compute resources were reserved via '--system-reserved' flag. Ex. '/system-reserved'. [default='']

--tls-cert-file string	
	File containing x509 Certificate used for serving HTTPS (with intermediate certs, if any, concatenated after server cert). If --tls-cert-file and --tls-private-key-file are not provided, a self-signed certificate and key are generated for the public address and saved to the directory passed to --cert-dir.
--tls-cipher-suites stringSlice	
	Comma-separated list of cipher suites for the server. If omitted, the default Go cipher suites will be used. Possible values: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_RC4_128_SHA,TLS_RSA_WITH_3DES_EDE_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_RC4_128_SHA
--tls-private-key-file string	
	File containing x509 private key matching --tls-cert-file.
-v, --v Level	
	log level for V logs
--version version[=true]	
	Print version information and quit
--vmodule moduleSpec	
	comma-separated list of pattern=N settings for file-filtered logging
--volume-plugin-dir string	
	The full path of the directory in which to search for additional third party volume plugins (default "/usr/libexec/kubernetes/kubelet-plugins/volume/exec/")
--volume-stats-agg-period duration	
	Specifies interval for kubelet to calculate and cache the volume disk usage for all pods and volumes.
-h, --help	
	help for kubelet

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

[Analytics]()

[Create an Issue]() [Edit This Page]()
Page last modified on July 06, 2019 at 5:28 PM PST by [Remove --allow-privileged as it is deprecated in latest 1.15.0 (#15285)]() ([Page History]())

[Edit This Page]()

Feature Gates

This page contains an overview of the various feature gates an administrator can specify on different Kubernetes components.

- [Overview]()
- [Using a Feature]()

Overview

Feature gates are a set of key=value pairs that describe alpha or experimental features. An administrator can use the `--feature-gates` command line flag on each component to turn a feature on or off. Each component supports a set of feature gates unique to that component. Use `-h` flag to see a full set of feature gates for all components. To set feature gates for a component, such as kubelet, use the `--feature-gates` flag assigned to a list of feature pairs:

```
--feature-gates="...,DynamicKubeletConfig=true"
```

The following table is a summary of the feature gates that you can set on different Kubernetes components.

- The "Since" column contains the Kubernetes release when a feature is introduced or its release stage is changed.
- The "Until" column, if not empty, contains the last Kubernetes release in which you can still use a feature gate.

Feature	Default	Stage	Since	Until
`Accelerators`	`false`	Alpha	1.6	1.10
`AdvancedAuditing`	`false`	Alpha	1.7	1.7
`AdvancedAuditing`	`true`	Beta	1.8	1.11
`AdvancedAuditing`	`true`	GA	1.12	-
`AffinityInAnnotations`	`false`	Alpha	1.6	1.7
`AllowExtTrafficLocalEndpoints`	`false`	Beta	1.4	1.6
`AllowExtTrafficLocalEndpoints`	`true`	GA	1.7	-
`APIListChunking`	`false`	Alpha	1.8	1.8
`APIListChunking`	`true`	Beta	1.9	
`APIResponseCompression`	`false`	Alpha	1.7	
`AppArmor`	`true`	Beta	1.4	
`AttachVolumeLimit`	`true`	Alpha	1.11	1.11
`AttachVolumeLimit`	`true`	Beta	1.12	
`BlockVolume`	`false`	Alpha	1.9	

Feature	Default	Stage	Since	Until
BlockVolume	true	Beta	1.13	-
BoundServiceAccountTokenVolume	false	Alpha	1.13	
CPUManager	false	Alpha	1.8	1.9
CPUManager	true	Beta	1.10	
CRIContainerLogRotation	false	Alpha	1.10	1.10
CRIContainerLogRotation	true	Beta	1.11	
CSIBlockVolume	false	Alpha	1.11	1.13
CSIBlockVolume	true	Beta	1.14	
CSIDriverRegistry	false	Alpha	1.12	1.13
CSIDriverRegistry	true	Beta	1.14	
CSIInlineVolume	false	Alpha	1.15	-
CSIMigration	false	Alpha	1.14	
CSIMigrationAWS	false	Alpha	1.14	
CSIMigrationAzureDisk	false	Alpha	1.15	
CSIMigrationAzureFile	false	Alpha	1.15	
CSIMigrationGCE	false	Alpha	1.14	
CSIMigrationOpenStack	false	Alpha	1.14	
CSINodeInfo	false	Alpha	1.12	1.13
CSINodeInfo	true	Beta	1.14	
CSIPersistentVolume	false	Alpha	1.9	1.9
CSIPersistentVolume	true	Beta	1.10	1.12
CSIPersistentVolume	true	GA	1.13	-
CustomCPUCFSQuotaPeriod	false	Alpha	1.12	
CustomPodDNS	false	Alpha	1.9	1.9
CustomPodDNS	true	Beta	1.10	1.13
CustomPodDNS	true	GA	1.14	-
CustomResourcePublishOpenAPI	false	Alpha	1.14	1.14
CustomResourcePublishOpenAPI	true	Beta	1.15	
CustomResourceSubresources	false	Alpha	1.10	1.11
CustomResourceSubresources	true	Beta	1.11	-
CustomResourceValidation	false	Alpha	1.8	1.8
CustomResourceValidation	true	Beta	1.9	
CustomResourceWebhookConversion	false	Alpha	1.13	1.14
CustomResourceWebhookConversion	true	Beta	1.15	
DebugContainers	false	Alpha	1.10	
DevicePlugins	false	Alpha	1.8	1.9
DevicePlugins	true	Beta	1.10	
DryRun	true	Beta	1.13	
DynamicAuditing	false	Alpha	1.13	
DynamicKubeletConfig	false	Alpha	1.4	1.10
DynamicKubeletConfig	true	Beta	1.11	
DynamicProvisioningScheduling	false	Alpha	1.11	1.11
DynamicVolumeProvisioning	true	Alpha	1.3	1.7
DynamicVolumeProvisioning	true	GA	1.8	
EnableEquivalenceClassCache	false	Alpha	1.8	

Feature	Default	Stage	Since	Until
ExpandCSIVolumes	false	Alpha	1.14	
ExpandInUsePersistentVolumes	false	Alpha	1.11	1.14
ExpandInUsePersistentVolumes	true	Beta	1.15	
ExpandPersistentVolumes	false	Alpha	1.8	1.10
ExpandPersistentVolumes	true	Beta	1.11	
ExperimentalCriticalPodAnnotation	false	Alpha	1.5	
ExperimentalHostUserNamespaceDefaulting	false	Beta	1.5	
GCERegionalPersistentDisk	true	Beta	1.10	1.12
GCERegionalPersistentDisk	true	GA	1.13	-
HugePages	false	Alpha	1.8	1.9
HugePages	true	Beta	1.10	1.13
HugePages	true	GA	1.14	
HyperVContainer	false	Alpha	1.10	
Initializers	false	Alpha	1.7	1.13
Initializers	-	Deprecated	1.14	
KubeletConfigFile	false	Alpha	1.8	1.9
KubeletPluginsWatcher	false	Alpha	1.11	1.11
KubeletPluginsWatcher	true	Beta	1.12	1.12
KubeletPluginsWatcher	true	GA	1.13	-
KubeletPodResources	false	Alpha	1.13	1.14
KubeletPodResources	true	Beta	1.15	
LocalStorageCapacityIsolation	false	Alpha	1.7	1.9
LocalStorageCapacityIsolation	true	Beta	1.10	
LocalStorageCapacityIsolationFSQuotaMonitoring	false	Alpha	1.15	
MountContainers	false	Alpha	1.9	
MountPropagation	false	Alpha	1.8	1.9
MountPropagation	true	Beta	1.10	1.11
MountPropagation	true	GA	1.12	
NodeLease	false	Alpha	1.12	1.13
NodeLease	true	Beta	1.14	
NonPreemptingPriority	false	Alpha	1.15	
PersistentLocalVolumes	false	Alpha	1.7	1.9
PersistentLocalVolumes	true	Beta	1.10	1.13
PersistentLocalVolumes	true	GA	1.14	
PodPriority	false	Alpha	1.8	1.10
PodPriority	true	Beta	1.11	1.13
PodPriority	true	GA	1.14	
PodReadinessGates	false	Alpha	1.11	1.11
PodReadinessGates	true	Beta	1.12	1.13
PodReadinessGates	true	GA	1.14	-
PodShareProcessNamespace	false	Alpha	1.10	
PodShareProcessNamespace	true	Beta	1.12	
ProcMountType	false	Alpha	1.12	
PVCProtection	false	Alpha	1.9	1.9
RemainingItemCount	false	Alpha	1.15	

Feature	Default	Stage	Since	Until
`ResourceLimitsPriorityFunction`	`false`	Alpha	1.9	
`RequestManagement`	`false`	Alpha	1.15	
`ResourceQuotaScopeSelectors`	`false`	Alpha	1.11	1.11
`ResourceQuotaScopeSelectors`	`true`	Beta	1.12	
`RotateKubeletClientCertificate`	`true`	Beta	1.8	
`RotateKubeletServerCertificate`	`false`	Alpha	1.7	1.11
`RotateKubeletServerCertificate`	`true`	Beta	1.12	
`RunAsGroup`	`true`	Beta	1.14	
`RuntimeClass`	`true`	Beta	1.14	
`SCTPSupport`	`false`	Alpha	1.12	
`ServerSideApply`	`false`	Alpha	1.14	
`ServiceLoadBalancerFinalizer`	`false`	Alpha	1.15	
`ServiceNodeExclusion`	`false`	Alpha	1.8	
`StorageObjectInUseProtection`	`true`	Beta	1.10	1.10
`StorageObjectInUseProtection`	`true`	GA	1.11	
`StorageVersionHash`	`false`	Alpha	1.14	1.14
`StorageVersionHash`	`true`	Beta	1.15	
`StreamingProxyRedirects`	`true`	Beta	1.5	
`SupportIPVSProxyMode`	`false`	Alpha	1.8	1.8
`SupportIPVSProxyMode`	`false`	Beta	1.9	1.9
`SupportIPVSProxyMode`	`true`	Beta	1.10	1.10
`SupportIPVSProxyMode`	`true`	GA	1.11	
`SupportNodePidsLimit`	`false`	Alpha	1.14	1.14
`SupportNodePidsLimit`	`true`	Beta	1.15	
`SupportPodPidsLimit`	`false`	Alpha	1.10	1.13
`SupportPodPidsLimit`	`true`	Beta	1.14	
`Sysctls`	`true`	Beta	1.11	
`TaintBasedEvictions`	`false`	Alpha	1.6	1.12
`TaintBasedEvictions`	`true`	Beta	1.13	
`TaintNodesByCondition`	`false`	Alpha	1.8	1.11
`TaintNodesByCondition`	`true`	Beta	1.12	
`TokenRequest`	`false`	Alpha	1.10	1.11
`TokenRequest`	`true`	Beta	1.12	
`TokenRequestProjection`	`false`	Alpha	1.11	1.11
`TokenRequestProjection`	`true`	Beta	1.12	
`TTLAfterFinished`	`false`	Alpha	1.12	
`VolumePVCDataSource`	`false`	Alpha	1.15	
`VolumeScheduling`	`false`	Alpha	1.9	1.9
`VolumeScheduling`	`true`	Beta	1.10	1.12
`VolumeScheduling`	`true`	GA	1.13	
`VolumeSubpathEnvExpansion`	`false`	Alpha	1.14	1.14
`VolumeSubpathEnvExpansion`	`true`	Beta	1.15	
`VolumeSnapshotDataSource`	`false`	Alpha	1.12	-
`ScheduleDaemonSetPods`	`false`	Alpha	1.11	1.11
`ScheduleDaemonSetPods`	`true`	Beta	1.12	

Feature	Default	Stage	Since	Until
`WatchBookmark`	`false`	Alpha	1.15	
`WindowsGMSA`	`false`	Alpha	1.14	

Using a Feature

Feature Stages

A feature can be in *Alpha*, *Beta* or *GA* stage. An *Alpha* feature means:

- Disabled by default.
- Might be buggy. Enabling the feature may expose bugs.
- Support for feature may be dropped at any time without notice.
- The API may change in incompatible ways in a later software release without notice.
- Recommended for use only in short-lived testing clusters, due to increased risk of bugs and lack of long-term support.

A *Beta* feature means:

- Enabled by default.
- The feature is well tested. Enabling the feature is considered safe.
- Support for the overall feature will not be dropped, though details may change.
- The schema and/or semantics of objects may change in incompatible ways in a subsequent beta or stable release. When this happens, we will provide instructions for migrating to the next version. This may require deleting, editing, and re-creating API objects. The editing process may require some thought. This may require downtime for applications that rely on the feature.
- Recommended for only non-business-critical uses because of potential for incompatible changes in subsequent releases. If you have multiple clusters that can be upgraded independently, you may be able to relax this restriction.

  Note: Please do try *Beta* features and give feedback on them! After they exit beta, it may not be practical for us to make more changes.

A *GA* feature is also referred to as a *stable* feature. It means:

- The corresponding feature gate is no longer needed.
- Stable versions of features will appear in released software for many subsequent versions.

Feature Gates

Each feature gate is designed for enabling/disabling a specific feature:

- `Accelerators`: Enable Nvidia GPU support when using Docker
- `AdvancedAuditing`: Enable [advanced auditing]
- `AffinityInAnnotations`(*deprecated*): Enable setting [Pod affinity or anti-affinitys].
- `AllowExtTrafficLocalEndpoints`: Enable a service to route external requests to node local endpoints.
- `APIListChunking`: Enable the API clients to retrieve (`LIST` or `GET`) resources from API server in chunks.
- `APIResponseCompression`: Compress the API responses for `LIST` or `GET` requests.
- `AppArmor`: Enable AppArmor based mandatory access control on Linux nodes when using Docker. See [AppArmor Tutorial] for more details.

- `AttachVolumeLimit`: Enable volume plugins to report limits on number of volumes that can be attached to a node. See dynamic volume limits for more details.
- `BlockVolume`: Enable the definition and consumption of raw block devices in Pods. See Raw Block Volume Support for more details.
- `BoundServiceAccountTokenVolume`: Migrate ServiceAccount volumes to use a projected volume consisting of a ServiceAccountTokenVolumeProjection. Check Service Account Token Volumes for more details.
- `CPUManager`: Enable container level CPU affinity support, see CPU Management Policies.
- `CRIContainerLogRotation`: Enable container log rotation for cri container runtime.
- `CSIBlockVolume`: Enable external CSI volume drivers to support block storage. See the `csi` raw block volume support documentation for more details.
- `CSIDriverRegistry`: Enable all logic related to the CSIDriver API object in csi.storage.k8s.io.
- `CSIInlineVolume`: Enable CSI Inline volumes support for pods.
- `CSIMigration`: Enables shims and translation logic to route volume operations from in-tree plugins to corresponding pre-installed CSI plugins
- `CSIMigrationAWS`: Enables shims and translation logic to route volume operations from the AWS-EBS in-tree plugin to EBS CSI plugin
- `CSIMigrationAzureDisk`: Enables shims and translation logic to route volume operations from the Azure-Disk in-tree plugin to Azure Disk CSI plugin
- `CSIMigrationAzureFile`: Enables shims and translation logic to route volume operations from the Azure-File in-tree plugin to Azure File CSI plugin
- `CSIMigrationGCE`: Enables shims and translation logic to route volume operations from the GCE-PD in-tree plugin to PD CSI plugin
- `CSIMigrationOpenStack`: Enables shims and translation logic to route volume operations from the Cinder in-tree plugin to Cinder CSI plugin
- `CSINodeInfo`: Enable all logic related to the CSINodeInfo API object in csi.storage.k8s.io.
- `CSIPersistentVolume`: Enable discovering and mounting volumes provisioned through a CSI (Container Storage Interface) compatible volume plugin. Check the `csi` volume type documentation for more details.
- `CustomCPUCFSQuotaPeriod`: Enable nodes to change CPUCFSQuotaPeriod.
- `CustomPodDNS`: Enable customizing the DNS settings for a Pod using its `dnsConfig` property. Check Pod's DNS Config for more details.
- `CustomResourcePublishOpenAPI`: Enables publishing of CRD OpenAPI specs.
- `CustomResourceSubresources`: Enable `/status` and `/scale` subresources on resources created from CustomResourceDefinition.
- `CustomResourceValidation`: Enable schema based validation on resources created from CustomResourceDefinition.
- `CustomResourceWebhookConversion`: Enable webhook-based conversion on resources created from CustomResourceDefinition.
- `DebugContainers`: Enable running a "debugging" container in a Pod's namespace to troubleshoot a running Pod.
- `DevicePlugins`: Enable the device-plugins based resource provisioning on nodes.
- `DryRun`: Enable server-side dry run requests.
- `DynamicAuditing`: Enable dynamic auditing
- `DynamicKubeletConfig`: Enable the dynamic configuration of kubelet. See Reconfigure kubelet.
- `DynamicProvisioningScheduling`: Extend the default scheduler to be aware of volume topology and handle PV provisioning. This feature is superceded by the `VolumeScheduling` feature completely in v1.12.
- `DynamicVolumeProvisioning`(*deprecated*): Enable the dynamic provisioning of persistent volumes to Pods.

- `EnableEquivalenceClassCache`: Enable the scheduler to cache equivalence of nodes when scheduling Pods.
- `ExpandInUsePersistentVolumes`: Enable expanding in-use PVCs. See Resizing an in-use PersistentVolumeClaim.
- `ExpandPersistentVolumes`: Enable the expanding of persistent volumes. See Expanding Persistent Volumes Claims.
- `ExperimentalCriticalPodAnnotation`: Enable annotating specific pods as *critical* so that their scheduling is guaranteed.
- `ExperimentalHostUserNamespaceDefaultingGate`: Enabling the defaulting user namespace to host. This is for containers that are using other host namespaces, host mounts, or containers that are privileged or using specific non-namespaced capabilities (e.g. `MKNODE`, `SYS_MODULE` etc.). This should only be enabled if user namespace remapping is enabled in the Docker daemon.
- `GCERegionalPersistentDisk`: Enable the regional PD feature on GCE.
- `HugePages`: Enable the allocation and consumption of pre-allocated huge pages.
- `HyperVContainer`: Enable Hyper-V isolation for Windows containers.
- `KubeletConfigFile`: Enable loading kubelet configuration from a file specified using a config file. See setting kubelet parameters via a config file for more details.
- `KubeletPluginsWatcher`: Enable probe-based plugin watcher utility to enable kubelet to discover plugins such as CSI volume drivers.
- `KubeletPodResources`: Enable the kubelet's pod resources grpc endpoint. See Support Device Monitoring for more details.
- `LocalStorageCapacityIsolation`: Enable the consumption of local ephemeral storage and also the `sizeLimit` property of an emptyDir volume.
- `LocalStorageCapacityIsolationFSQuotaMonitoring`: When `LocalStorageCapacityIsolation` is enabled for local ephemeral storage and the backing filesystem for emptyDir volumes supports project quotas and they are enabled, use project quotas to monitor emptyDir volume storage consumption rather than filesystem walk for better performance and accuracy.
- `MountContainers`: Enable using utility containers on host as the volume mounter.
- `MountPropagation`: Enable sharing volume mounted by one container to other containers or pods. For more details, please see mount propagation.
- `NodeLease`: Enable the new Lease API to report node heartbeats, which could be used as a node health signal.
- `NonPreemptingPriority`: Enable NonPreempting option for PriorityClass and Pod.
- `PersistentLocalVolumes`: Enable the usage of `local` volume type in Pods. Pod affinity has to be specified if requesting a `local` volume.
- `PodPriority`: Enable the descheduling and preemption of Pods based on their priorities.
- `PodReadinessGates`: Enable the setting of `PodReadinessGate` field for extending Pod readiness evaluation. For more details, please see Pod readiness gate.
- `ProcMountType`: Enables control over ProcMountType for containers.
- `PVCProtection`: Enable the prevention of a PersistentVolumeClaim (PVC) from being deleted when it is still used by any Pod. More details can be found here.
- `ResourceLimitsPriorityFunction`: Enable a scheduler priority function that assigns a lowest possible score of 1 to a node that satisfies at least one of the input Pod's cpu and memory limits. The intent is to break ties between nodes with same scores.
- `RequestManagement`: Enable managing request concurrency with prioritization and fairness at each server.
- `ResourceQuotaScopeSelectors`: Enable resource quota scope selectors.
- `RotateKubeletClientCertificate`: Enable the rotation of the client TLS certificate on the kubelet. See kubelet configuration for more details.
- `RotateKubeletServerCertificate`: Enable the rotation of the server TLS certificate on the kubelet. See kubelet configuration for more details.

- `RunAsGroup`: Enable control over the primary group ID set on the init processes of containers.
- `RuntimeClass`: Enable the RuntimeClass feature for selecting container runtime configurations.
- `ScheduleDaemonSetPods`: Enable DaemonSet Pods to be scheduled by the default scheduler instead of the DaemonSet controller.
- `SCTPSupport`: Enables the usage of SCTP as `protocol` value in `Service`, `Endpoint`, `NetworkPolicy` and `Pod` definitions
- `ServerSideApply`: Enables the Sever Side Apply (SSA) path at the API Server.
- `ServiceLoadBalancerFinalizer`: Enable finalizer protection for Service load balancers.
- `ServiceNodeExclusion`: Enable the exclusion of nodes from load balancers created by a cloud provider. A node is eligible for exclusion if annotated with "`alpha.service-controller.kubernetes.io/exclude-balancer`" key.
- `StorageObjectInUseProtection`: Postpone the deletion of PersistentVolume or PersistentVolumeClaim objects if they are still being used.
- `StorageVersionHash`: Allow apiservers to expose the storage version hash in the discovery.
- `StreamingProxyRedirects`: Instructs the API server to intercept (and follow) redirects from the backend (kubelet) for streaming requests. Examples of streaming requests include the `exec`, `attach` and `port-forward` requests.
- `SupportIPVSProxyMode`: Enable providing in-cluster service load balancing using IPVS. See service proxies for more details.
- `SupportPodPidsLimit`: Enable the support to limiting PIDs in Pods.
- `Sysctls`: Enable support for namespaced kernel parameters (sysctls) that can be set for each pod. See sysctls for more details.
- `TaintBasedEvictions`: Enable evicting pods from nodes based on taints on nodes and tolerations on Pods. See taints and tolerations for more details.
- `TaintNodesByCondition`: Enable automatic tainting nodes based on node conditions.
- `TokenRequest`: Enable the `TokenRequest` endpoint on service account resources.
- `TokenRequestProjection`: Enable the injection of service account tokens into a Pod through the `projected` volume.
- `TTLAfterFinished`: Allow a TTL controller to clean up resources after they finish execution.
- `VolumePVCDataSource`: Enable support for specifying an existing PVC as a DataSource.
- `VolumeScheduling`: Enable volume topology aware scheduling and make the PersistentVolumeClaim (PVC) binding aware of scheduling decisions. It also enables the usage of `local` volume type when used together with the `PersistentLocalVolumes` feature gate.
- `VolumeSnapshotDataSource`: Enable volume snapshot data source support.
- `VolumeSubpathEnvExpansion`: Enable `subPathExpr` field for expanding environment variables into a `subPath`.
- `WatchBookmark`: Enable support for watch bookmark events.
- `WindowsGMSA`: Enables passing of GMSA credential specs from pods to container runtimes.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 24, 2019 at 9:27 AM PST by Update feature-gates.md with v1.15 release (#15071) (Page History)

Edit This Page

Kubelet authentication/authorization

- - Overview
  - Kubelet authentication
  - Kubelet 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:

HTTP verb	request verb
POST	create
GET, HEAD	get
PUT	update
PATCH	patch
DELETE	delete

The resource and subresource is determined from the incoming request's path:

Kubelet API	resource	subresource
/stats/*	nodes	stats
/metrics/*	nodes	metrics
/logs/*	nodes	log
/spec/*	nodes	spec
all others	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

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on July 18, 2018 at 6:07 PM PST by Update authorization links (#9465) (Page History)

Edit This Page

TLS bootstrapping

In a Kubernetes cluster, the components on the worker nodes - kubelet and kube-proxy - need to communicate with Kubernetes master 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 to simplify the process. 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
- Configuration
- Certificate Authority
- kube-apiserver configuration

- kube-controller-manager configuration
- kubelet configuration
- Other authenticating components
- kubectl approval
- Limits

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 master 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
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 master nodes.

For the purposes of this document, we will assume these have been distributed to master 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 - **beta**
2. Token authentication file

Bootstrap tokens are a simpler and more easily managed method to authenticate kubelets, and do not require any additional flags when starting kube-apiserver. Using bootstrap tokens is currently **beta** as of Kubernetes version 1.12.

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 an 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 ' '
```

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:bootstrappers"
```

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 just these) permissions, `system:node-bootstrapper`.

To do this, you just 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 master 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"
--cluster-signing-key-file="/etc/path/to/kubernetes/ca/ca.key"
```

for example:

```
--cluster-signing-cert-file="/var/lib/kubernetes/ca.pem" --
cluster-signing-key-file="/var/lib/kubernetes/ca-key.pem"
```

The validity duration of signed certificates can be configured with flag:

```
--experimental-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`:

```yaml
# 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:nod
eclient
  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`:

```yaml
# 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:sel
fnodeclient
  apiGroup: rbac.authorization.k8s.io
```

Note: Kubernetes Below 1.8: If you are running an earlier version of Kubernetes, notably a version below 1.8, then the cluster roles referenced above do not ship by default. You will have to create them yourself *in addition to* the `ClusterRoleBindings` listed.

To create the `ClusterRoles`:

```yaml
# A ClusterRole which instructs the CSR approver to approve a
user requesting
# node client credentials.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: system:certificates.k8s.io:certificatesigningrequests:nod
eclient
rules:
- apiGroups: ["certificates.k8s.io"]
  resources: ["certificatesigningrequests/nodeclient"]
  verbs: ["create"]
```

```
---
# A ClusterRole which instructs the CSR approver to approve a
node renewing its
# own client credentials.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: system:certificates.k8s.io:certificatesigningrequests:sel
fnodeclient
rules:
- apiGroups: ["certificates.k8s.io"]
  resources: ["certificatesigningrequests/selfnodeclient"]
  verbs: ["create"]
```

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 builtin 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 master 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 just 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-cluster bootstrap --server='https://my.server.example.com:6443' --certificate-authority=/var/lib/kubernetes/ca.pem
kubectl config --kubeconfig=/var/lib/kubelet/bootstrap-kubeconfig set-credentials kubelet-bootstrap --token=07401b.f395accd246ae52d
kubectl config --kubeconfig=/var/lib/kubelet/bootstrap-kubeconfig set-context bootstrap --user=kubelet-bootstrap --cluster=bootstrap
kubectl config --kubeconfig=/var/lib/kubelet/bootstrap-kubeconfig use-context bootstrap
```

To indicate to the kubelet to use the bootstrap `kubeconfig`, use the following kubelet flag:

```
--bootstrap-kubeconfig="/var/lib/kubelet/bootstrap-kubeconfig" --kubeconfig="/var/lib/kubelet/kubeconfig"
```

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 **beta** features for enabling rotation of its client and/or serving certficates. These can be enabled through the respective `RotateKubeletClientCertificate` and `RotateKubeletServerCertificate` feature flags on the kubelet and are enabled by default.

`RotateKubeletClientCertificate` causes the kubelet to rotate its client certificates by creating new CSRs as its existing credentials expire. To enable this feature pass the following flag to the kubelet:

```
--rotate-certificates
```

`RotateKubeletServerCertificate` causes the kubelet **both** to request a serving certificate after bootstrapping its client credentials **and** to rotate that certificate. To enable this feature pass the following flag to the kubelet:

```
--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.

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 control plane 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 builtin to 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>`.

Limits

Although Kubernetes supports running control plane master components like kube-apiserver and kube-controller-manager in containers, and even as `Pods` in a kubelet, as of this writing, you cannot both TLS Bootstrap a kubelet and run master plane components on it.

The reason for this limitation is that the kubelet attempts to bootstrap communication with kube-apiserver *before* starting any pods, even static ones define on disk and referenced via the kubelet option `--pod-manifest-path=<PATH>`. Trying to do both TLS Bootstrapping and master components in kubelet leads to a race condition: kubelet needs to communicate to kube-apiserver to request certificates, yet requires those certificates to be available to start kube-apiserver.

An issue is open referencing this here.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on May 08, 2019 at 10:40 PM PST by Fix orders of yaml of reference/kube* (#14238) (Page History)

Edit This Page

- - cloud-controller-manager
    - Synopsis
    - Options

cloud-controller-manager

Synopsis

The Cloud controller manager is a daemon that embeds the cloud specific control loops shipped with Kubernetes.

```
cloud-controller-manager [flags]
```

Options

--allocate-node-cidrs	
	Should CIDRs for Pods be allocated and set on the cloud provider.
--alsologtostderr	
	log to standard error as well as files
--authentication-kubeconfig string	
	kubeconfig file pointing at the 'core' kubernetes server with enough rights to create tokenaccessreviews.authentication.k8s.io. This is optional. If empty, all token requests are considered to be anonymous and no client CA is looked up in the cluster.
--authentication-skip-lookup	
	If false, the authentication-kubeconfig will be used to lookup missing authentication configuration from the cluster.
--authentication-token-webhook-cache-ttl durationÂ Â Â Â Â Default: 10s	
	The duration to cache responses from the webhook token authenticator.
--authentication-tolerate-lookup-failure	
	If true, failures to look up missing authentication configuration from the cluster are not considered fatal. Note that this can result in authentication that treats all requests as anonymous.
--authorization-always-allow-paths stringSliceÂ Â Â Â Â Default: [/healthz]	
	A list of HTTP paths to skip during authorization, i.e. these are authorized without contacting the 'core' kubernetes server.
--authorization-kubeconfig string	
	kubeconfig file pointing at the 'core' kubernetes server with enough rights to create subjectaccessreviews.authorization.k8s.io. This is optional. If empty, all requests not skipped by authorization are forbidden.
--authorization-webhook-cache-authorized-ttl durationÂ Â Â Â Â Default: 10s	
	The duration to cache 'authorized' responses from the webhook authorizer.
--authorization-webhook-cache-unauthorized-ttl durationÂ Â Â Â Â Default: 10s	
	The duration to cache 'unauthorized' responses from the webhook authorizer.
--azure-container-registry-config string	
	Path to the file containing Azure container registry configuration information.
--bind-address ipÂ Â Â Â Â Default: 0.0.0.0	
	The IP address on which to listen for the --secure-port port. The associated interface(s) must be reachable by the rest of the cluster, and by CLI/web clients. If blank, all interfaces will be used (0.0.0.0 for all IPv4 interfaces and :: for all IPv6 interfaces).
--cert-dir string	
	The directory where the TLS certs are located. If --tls-cert-file and --tls-private-key-file are provided, this flag will be ignored.
--cidr-allocator-type stringÂ Â Â Â Â Default: "RangeAllocator"	
	Type of CIDR allocator to use
--client-ca-file string	

	If set, any request presenting a client certificate signed by one of the authorities in the client-ca-file is authenticated with an identity corresponding to the CommonName of the client certificate.
--cloud-config string	
	The path to the cloud provider configuration file. Empty string for no configuration file.
--cloud-provider string	
	The provider for cloud services. Empty string for no provider.
--cloud-provider-gce-lb-src-cidrs cidrsÂ Â Â Â Â Default: 130.211.0.0/22,209.85.152.0/22,209.85.204.0/22,35.191.0.0/16	
	CIDRs opened in GCE firewall for LB traffic proxy & health checks
--cluster-cidr string	
	CIDR Range for Pods in cluster. Requires --allocate-node-cidrs to be true
--cluster-name stringÂ Â Â Â Â Default: "kubernetes"	
	The instance prefix for the cluster.
--concurrent-service-syncs int32Â Â Â Â Â Default: 1	
	The number of services that are allowed to sync concurrently. Larger number = more responsive service management, but more CPU (and network) load
--configure-cloud-routesÂ Â Â Â Â Default: true	
	Should CIDRs allocated by allocate-node-cidrs be configured on the cloud provider.
--contention-profiling	
	Enable lock contention profiling, if profiling is enabled
--controller-start-interval duration	
	Interval between starting controller managers.
--controllers stringSliceÂ Â Â Â Â Default: [*]	
	A list of controllers to enable. '*' enables all on-by-default controllers, 'foo' enables the controller named 'foo', '-foo' disables the controller named 'foo'. All controllers: cloud-node, cloud-node-lifecycle, route, service Disabled-by-default controllers:
--external-cloud-volume-plugin string	
	The plugin to use when cloud provider is set to external. Can be empty, should only be set when cloud-provider is external. Currently used to allow node and volume controllers to work for in tree cloud providers.
--feature-gates mapStringBool	

	A set of key=value pairs that describe feature gates for alpha/experimental features. Options are: APIListChunking=true|false (BETA - default=true) APIResponseCompression=true|false (ALPHA - default=false) AllAlpha=true|false (ALPHA - default=false) AppArmor=true|false (BETA - default=true) AttachVolumeLimit=true|false (BETA - default=true) BalanceAttachedNodeVolumes=true|false (ALPHA - default=false) BlockVolume=true|false (BETA - default=true) BoundServiceAccountTokenVolume=true|false (ALPHA - default=false) CPUManager=true|false (BETA - default=true) CRIContainerLogRotation=true|false (BETA - default=true) CSIBlockVolume=true|false (BETA - default=true) CSIDriverRegistry=true|false (BETA - default=true) CSIInlineVolume=true|false (ALPHA - default=false) CSIMigration=true|false (ALPHA - default=false) CSIMigrationAWS=true|false (ALPHA - default=false) CSIMigrationGCE=true|false (ALPHA - default=false) CSIMigrationOpenStack=true|false (ALPHA - default=false) CSINodeInfo=true|false (BETA - default=true) CustomCPUCFSQuotaPeriod=true|false (ALPHA - default=false) CustomResourcePublishOpenAPI=true|false (ALPHA - default=false) CustomResourceSubresources=true|false (BETA - default=true) CustomResourceValidation=true|false (BETA - default=true) CustomResourceWebhookConversion=true|false (ALPHA - default=false) DebugContainers=true|false (ALPHA - default=false) DevicePlugins=true|false (BETA - default=true) DryRun=true|false (BETA - default=true) DynamicAuditing=true|false (ALPHA - default=false) DynamicKubeletConfig=true|false (BETA - default=true) ExpandCSIVolumes=true|false (ALPHA - default=false) ExpandInUsePersistentVolumes=true|false (ALPHA - default=false) ExpandPersistentVolumes=true|false (BETA - default=true) ExperimentalCriticalPodAnnotation=true|false (ALPHA - default=false) ExperimentalHostUserNamespaceDefaulting=true|false (BETA - default=false) HyperVContainer=true|false (ALPHA - default=false) KubeletPodResources=true|false (ALPHA - default=false) LocalStorageCapacityIsolation=true|false (BETA - default=true) MountContainers=true|false (ALPHA - default=false) NodeLease=true|false (BETA - default=true) PodShareProcessNamespace=true|false (BETA - default=true) ProcMountType=true|false (ALPHA - default=false) QOSReserved=true|false (ALPHA - default=false) ResourceLimitsPriorityFunction=true|false (ALPHA - default=false) ResourceQuotaScopeSelectors=true|false (BETA - default=true) RotateKubeletClientCertificate=true|false (BETA - default=true) RotateKubeletServerCertificate=true|false (BETA - default=true) RunAsGroup=true|false (BETA - default=true) RuntimeClass=true|false (BETA - default=true) SCTPSupport=true|false (ALPHA - default=false)

-h, --help	
	help for cloud-controller-manager
--http2-max-streams-per-connection int	
	The limit that the server gives to clients for the maximum number of streams in an HTTP/2 connection. Zero means to use golang's default.
--kube-api-burst int32Â Â Â Â Â Default: 30	
	Burst to use while talking with kubernetes apiserver.
--kube-api-content-type stringÂ Â Â Â Â Default: "application/vnd.kubernetes.protobuf"	
	Content type of requests sent to apiserver.
--kube-api-qps float32Â Â Â Â Â Default: 20	
	QPS to use while talking with kubernetes apiserver.
--kubeconfig string	
	Path to kubeconfig file with authorization and master location information.
--leader-electÂ Â Â Â Â Default: true	
	Start a leader election client and gain leadership before executing the main loop. Enable this when running replicated components for high availability.
--leader-elect-lease-duration durationÂ Â Â Â Â Default: 15s	
	The duration that non-leader candidates will wait after observing a leadership renewal until attempting to acquire leadership of a led but unrenewed leader slot. This is effectively the maximum duration that a leader can be stopped before it is replaced by another candidate. This is only applicable if leader election is enabled.
--leader-elect-renew-deadline durationÂ Â Â Â Â Default: 10s	
	The interval between attempts by the acting master to renew a leadership slot before it stops leading. This must be less than or equal to the lease duration. This is only applicable if leader election is enabled.
--leader-elect-resource-lock endpointsÂ Â Â Â Â Default: "endpoints"	
	The type of resource object that is used for locking during leader election. Supported options are endpoints (default) and `configmaps`.
--leader-elect-retry-period durationÂ Â Â Â Â Default: 2s	
	The duration the clients should wait between attempting acquisition and renewal of a leadership. This is only applicable if leader election is enabled.
--log-backtrace-at traceLocationÂ Â Â Â Â Default: :0	
	when logging hits line file:N, emit a stack trace
--log-dir string	
	If non-empty, write log files in this directory
--log-file string	
	If non-empty, use this log file
--log-flush-frequency durationÂ Â Â Â Â Default: 5s	
	Maximum number of seconds between log flushes
--logtostderrÂ Â Â Â Â Default: true	
	log to standard error instead of files
--master string	
	The address of the Kubernetes API server (overrides any value in kubeconfig).
--min-resync-period durationÂ Â Â Â Â Default: 12h0m0s	

	The resync period in reflectors will be random between MinResyncPeriod and 2*MinResyncPeriod.
--node-monitor-period durationÂ Â Â Â Â Default: 5s	
	The period for syncing NodeStatus in NodeController.
--node-status-update-frequency durationÂ Â Â Â Â Default: 5m0s	
	Specifies how often the controller updates nodes' status.
--profiling	
	Enable profiling via web interface host:port/debug/pprof/
--requestheader-allowed-names stringSlice	
	List of client certificate common names to allow to provide usernames in headers specified by --requestheader-username-headers. If empty, any client certificate validated by the authorities in --requestheader-client-ca-file is allowed.
--requestheader-client-ca-file string	
	Root certificate bundle to use to verify client certificates on incoming requests before trusting usernames in headers specified by --requestheader-username-headers. WARNING: generally do not depend on authorization being already done for incoming requests.
--requestheader-extra-headers-prefix stringSliceÂ Â Â Â Â Default: [x-remote-extra-]	
	List of request header prefixes to inspect. X-Remote-Extra- is suggested.
--requestheader-group-headers stringSliceÂ Â Â Â Â Default: [x-remote-group]	
	List of request headers to inspect for groups. X-Remote-Group is suggested.
--requestheader-username-headers stringSliceÂ Â Â Â Â Default: [x-remote-user]	
	List of request headers to inspect for usernames. X-Remote-User is common.
--route-reconciliation-period durationÂ Â Â Â Â Default: 10s	
	The period for reconciling routes created for Nodes by cloud provider.
--secure-port intÂ Â Â Â Â Default: 10258	
	The port on which to serve HTTPS with authentication and authorization.If 0, don't serve HTTPS at all.
--skip-headers	
	If true, avoid header prefixes in the log messages
--stderrthreshold severityÂ Â Â Â Â Default: 2	
	logs at or above this threshold go to stderr
--tls-cert-file string	
	File containing the default x509 Certificate for HTTPS. (CA cert, if any, concatenated after server cert). If HTTPS serving is enabled, and --tls-cert-file and --tls-private-key-file are not provided, a self-signed certificate and key are generated for the public address and saved to the directory specified by --cert-dir.
--tls-cipher-suites stringSlice	

	Comma-separated list of cipher suites for the server. If omitted, the default Go cipher suites will be use. Possible values: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_RC4_128_SHA,TLS_RSA_WITH_3DES_EDE_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_RC4_128_SHA
--tls-min-version string	
	Minimum TLS version supported. Possible values: VersionTLS10, VersionTLS11, VersionTLS12
--tls-private-key-file string	
	File containing the default x509 private key matching --tls-cert-file.
--tls-sni-cert-key namedCertKeyÂ Â Â Â Â Default: []	
	A pair of x509 certificate and private key file paths, optionally suffixed with a list of domain patterns which are fully qualified domain names, possibly with prefixed wildcard segments. If no domain patterns are provided, the names of the certificate are extracted. Non-wildcard matches trump over wildcard matches, explicit domain patterns trump over extracted names. For multiple key/certificate pairs, use the --tls-sni-cert-key multiple times. Examples: "example.crt,example.key" or "foo.crt,foo.key:*.foo.com,foo.com".
--use-service-account-credentials	
	If true, use individual service account credentials for each controller.
-v, --v Level	
	number for the log level verbosity
--version version[=true]	
	Print version information and quit
--vmodule moduleSpec	
	comma-separated list of pattern=N settings for file-filtered logging

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

- - kube-apiserver
    - Synopsis
    - Options

kube-apiserver

Synopsis

The Kubernetes API server validates and configures data for the api objects which include pods, services, replicationcontrollers, and others. The API Server services REST operations and provides the frontend to the cluster's shared state through which all other components interact.

```
kube-apiserver [flags]
```

Options

--admission-control-config-file string	
	File with admission control configuration.
--advertise-address ip	
	The IP address on which to advertise the apiserver to members of the cluster. This address must be reachable by the rest of the cluster. If blank, the --bind-address will be used. If --bind-address is unspecified, the host's default interface will be used.
--allow-privileged	
	If true, allow privileged containers. [default=false]
--alsologtostderr	
	log to standard error as well as files
--anonymous-authÂ Â Â Â Â Default: true	
	Enables anonymous requests to the secure port of the API server. Requests that are not rejected by another authentication method are treated as anonymous requests. Anonymous requests have a username of system:anonymous, and a group name of system:unauthenticated.
--api-audiences stringSlice	
	Identifiers of the API. The service account token authenticator will validate that tokens used against the API are bound to at least one of these audiences. If the --service-account-issuer flag is configured and this flag is not, this field defaults to a single element list containing the issuer URL .
--apiserver-count intÂ Â Â Â Â Default: 1	
	The number of apiservers running in the cluster, must be a positive number. (In use when --endpoint-reconciler-type=master-count is enabled.)
--audit-dynamic-configuration	
	Enables dynamic audit configuration. This feature also requires the DynamicAuditing feature flag
--audit-log-batch-buffer-size intÂ Â Â Â Â Default: 10000	

	The size of the buffer to store events before batching and writing. Only used in batch mode.
--audit-log-batch-max-size intÂ Â Â Â Â Default: 1	
	The maximum size of a batch. Only used in batch mode.
--audit-log-batch-max-wait duration	
	The amount of time to wait before force writing the batch that hadn't reached the max size. Only used in batch mode.
--audit-log-batch-throttle-burst int	
	Maximum number of requests sent at the same moment if ThrottleQPS was not utilized before. Only used in batch mode.
--audit-log-batch-throttle-enable	
	Whether batching throttling is enabled. Only used in batch mode.
--audit-log-batch-throttle-qps float32	
	Maximum average number of batches per second. Only used in batch mode.
--audit-log-format stringÂ Â Â Â Â Default: "json"	
	Format of saved audits. "legacy" indicates 1-line text format for each event. "json" indicates structured json format. Known formats are legacy,json.
--audit-log-maxage int	
	The maximum number of days to retain old audit log files based on the timestamp encoded in their filename.
--audit-log-maxbackup int	
	The maximum number of old audit log files to retain.
--audit-log-maxsize int	
	The maximum size in megabytes of the audit log file before it gets rotated.
--audit-log-mode stringÂ Â Â Â Â Default: "blocking"	
	Strategy for sending audit events. Blocking indicates sending events should block server responses. Batch causes the backend to buffer and write events asynchronously. Known modes are batch,blocking,blocking-strict.
--audit-log-path string	
	If set, all requests coming to the apiserver will be logged to this file. '-' means standard out.
--audit-log-truncate-enabled	
	Whether event and batch truncating is enabled.
--audit-log-truncate-max-batch-size intÂ Â Â Â Â Default: 10485760	
	Maximum size of the batch sent to the underlying backend. Actual serialized size can be several hundreds of bytes greater. If a batch exceeds this limit, it is split into several batches of smaller size.
--audit-log-truncate-max-event-size intÂ Â Â Â Â Default: 102400	
	Maximum size of the audit event sent to the underlying backend. If the size of an event is greater than this number, first request and response are removed, and if this doesn't reduce the size enough, event is discarded.
--audit-log-version stringÂ Â Â Â Â Default: "audit.k8s.io/v1"	
	API group and version used for serializing audit events written to log.
--audit-policy-file string	
	Path to the file that defines the audit policy configuration.
--audit-webhook-batch-buffer-size intÂ Â Â Â Â Default: 10000	

	The size of the buffer to store events before batching and writing. Only used in batch mode.
--audit-webhook-batch-max-size intÂ Â Â Â Â Default: 400	
	The maximum size of a batch. Only used in batch mode.
--audit-webhook-batch-max-wait durationÂ Â Â Â Â Default: 30s	
	The amount of time to wait before force writing the batch that hadn't reached the max size. Only used in batch mode.
--audit-webhook-batch-throttle-burst intÂ Â Â Â Â Default: 15	
	Maximum number of requests sent at the same moment if ThrottleQPS was not utilized before. Only used in batch mode.
--audit-webhook-batch-throttle-enableÂ Â Â Â Â Default: true	
	Whether batching throttling is enabled. Only used in batch mode.
--audit-webhook-batch-throttle-qps float32Â Â Â Â Â Default: 10	
	Maximum average number of batches per second. Only used in batch mode.
--audit-webhook-config-file string	
	Path to a kubeconfig formatted file that defines the audit webhook configuration.
--audit-webhook-initial-backoff durationÂ Â Â Â Â Default: 10s	
	The amount of time to wait before retrying the first failed request.
--audit-webhook-mode stringÂ Â Â Â Â Default: "batch"	
	Strategy for sending audit events. Blocking indicates sending events should block server responses. Batch causes the backend to buffer and write events asynchronously. Known modes are batch,blocking,blocking-strict.
--audit-webhook-truncate-enabled	
	Whether event and batch truncating is enabled.
--audit-webhook-truncate-max-batch-size intÂ Â Â Â Â Default: 10485760	
	Maximum size of the batch sent to the underlying backend. Actual serialized size can be several hundreds of bytes greater. If a batch exceeds this limit, it is split into several batches of smaller size.
--audit-webhook-truncate-max-event-size intÂ Â Â Â Â Default: 102400	
	Maximum size of the audit event sent to the underlying backend. If the size of an event is greater than this number, first request and response are removed, and if this doesn't reduce the size enough, event is discarded.
--audit-webhook-version stringÂ Â Â Â Â Default: "audit.k8s.io/v1"	
	API group and version used for serializing audit events written to webhook.
--authentication-token-webhook-cache-ttl durationÂ Â Â Â Â Default: 2m0s	
	The duration to cache responses from the webhook token authenticator.
--authentication-token-webhook-config-file string	
	File with webhook configuration for token authentication in kubeconfig format. The API server will query the remote service to determine authentication for bearer tokens.
--authorization-mode stringSliceÂ Â Â Â Â Default: [AlwaysAllow]	
	Ordered list of plug-ins to do authorization on secure port. Comma-delimited list of: AlwaysAllow,AlwaysDeny,ABAC,Webhook,RBAC,Node.
--authorization-policy-file string	
	File with authorization policy in json line by line format, used with --authorization-mode=ABAC, on the secure port.

--authorization-webhook-cache-authorized-ttl durationÂ Â Â Â Â Default: 5m0s	
	The duration to cache 'authorized' responses from the webhook authorizer.
--authorization-webhook-cache-unauthorized-ttl durationÂ Â Â Â Â Default: 30s	
	The duration to cache 'unauthorized' responses from the webhook authorizer.
--authorization-webhook-config-file string	
	File with webhook configuration in kubeconfig format, used with --authorization-mode=Webhook. The API server will query the remote service to determine access on the API server's secure port.
--azure-container-registry-config string	
	Path to the file containing Azure container registry configuration information.
--basic-auth-file string	
	If set, the file that will be used to admit requests to the secure port of the API server via http basic authentication.
--bind-address ipÂ Â Â Â Â Default: 0.0.0.0	
	The IP address on which to listen for the --secure-port port. The associated interface(s) must be reachable by the rest of the cluster, and by CLI/web clients. If blank, all interfaces will be used (0.0.0.0 for all IPv4 interfaces and :: for all IPv6 interfaces).
--cert-dir stringÂ Â Â Â Â Default: "/var/run/kubernetes"	
	The directory where the TLS certs are located. If --tls-cert-file and --tls-private-key-file are provided, this flag will be ignored.
--client-ca-file string	
	If set, any request presenting a client certificate signed by one of the authorities in the client-ca-file is authenticated with an identity corresponding to the CommonName of the client certificate.
--cloud-config string	
	The path to the cloud provider configuration file. Empty string for no configuration file.
--cloud-provider string	
	The provider for cloud services. Empty string for no provider.
--cloud-provider-gce-lb-src-cidrs cidrsÂ Â Â Â Â Default: 130.211.0.0/22,209.85.152.0/22,209.85.204.0/22,35.191.0.0/16	
	CIDRs opened in GCE firewall for LB traffic proxy & health checks
--contention-profiling	
	Enable lock contention profiling, if profiling is enabled
--cors-allowed-origins stringSlice	
	List of allowed origins for CORS, comma separated. An allowed origin can be a regular expression to support subdomain matching. If this list is empty CORS will not be enabled.
--default-not-ready-toleration-seconds intÂ Â Â Â Â Default: 300	
	Indicates the tolerationSeconds of the toleration for notReady:NoExecute that is added by default to every pod that does not already have such a toleration.
--default-unreachable-toleration-seconds intÂ Â Â Â Â Default: 300	
	Indicates the tolerationSeconds of the toleration for unreachable:NoExecute that is added by default to every pod that does not already have such a toleration.
--default-watch-cache-size intÂ Â Â Â Â Default: 100	

	Default watch cache size. If zero, watch cache will be disabled for resources that do not have a default watch size set.
--delete-collection-workers intÂ Â Â Â Â Default: 1	
	Number of workers spawned for DeleteCollection call. These are used to speed up namespace cleanup.
--disable-admission-plugins stringSlice	
	admission plugins that should be disabled although they are in the default enabled plugins list (NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, Priority, DefaultTolerationSeconds, DefaultStorageClass, PersistentVolumeClaimResize, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota). Comma-delimited list of admission plugins: AlwaysAdmit, AlwaysDeny, AlwaysPullImages, DefaultStorageClass, DefaultTolerationSeconds, DenyEscalatingExec, DenyExecOnPrivileged, EventRateLimit, ExtendedResourceToleration, ImagePolicyWebhook, LimitPodHardAntiAffinityTopology, LimitRanger, MutatingAdmissionWebhook, NamespaceAutoProvision, NamespaceExists, NamespaceLifecycle, NodeRestriction, OwnerReferencesPermissionEnforcement, PersistentVolumeClaimResize, PersistentVolumeLabel, PodNodeSelector, PodPreset, PodSecurityPolicy, PodTolerationRestriction, Priority, ResourceQuota, SecurityContextDeny, ServiceAccount, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionWebhook. The order of plugins in this flag does not matter.
--enable-admission-plugins stringSlice	
	admission plugins that should be enabled in addition to default enabled ones (NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, Priority, DefaultTolerationSeconds, DefaultStorageClass, PersistentVolumeClaimResize, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota). Comma-delimited list of admission plugins: AlwaysAdmit, AlwaysDeny, AlwaysPullImages, DefaultStorageClass, DefaultTolerationSeconds, DenyEscalatingExec, DenyExecOnPrivileged, EventRateLimit, ExtendedResourceToleration, ImagePolicyWebhook, LimitPodHardAntiAffinityTopology, LimitRanger, MutatingAdmissionWebhook, NamespaceAutoProvision, NamespaceExists, NamespaceLifecycle, NodeRestriction, OwnerReferencesPermissionEnforcement, PersistentVolumeClaimResize, PersistentVolumeLabel, PodNodeSelector, PodPreset, PodSecurityPolicy, PodTolerationRestriction, Priority, ResourceQuota, SecurityContextDeny, ServiceAccount, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionWebhook. The order of plugins in this flag does not matter.
--enable-aggregator-routing	
	Turns on aggregator routing requests to endpoints IP rather than cluster IP.
--enable-bootstrap-token-auth	
	Enable to allow secrets of type 'bootstrap.kubernetes.io/token' in the 'kube-system' namespace to be used for TLS bootstrapping authentication.
--enable-garbage-collectorÂ Â Â Â Â Default: true	
	Enables the generic garbage collector. MUST be synced with the corresponding flag of the kube-controller-manager.
--enable-logs-handlerÂ Â Â Â Â Default: true	
	If true, install a /logs handler for the apiserver logs.

--encryption-provider-config string
The file containing configuration for encryption providers to be used for storing secrets in etcd
--endpoint-reconciler-type stringÂ Â Â Â Â Default: "lease"
Use an endpoint reconciler (master-count, lease, none)
--etcd-cafile string
SSL Certificate Authority file used to secure etcd communication.
--etcd-certfile string
SSL certification file used to secure etcd communication.
--etcd-compaction-interval durationÂ Â Â Â Â Default: 5m0s
The interval of compaction requests. If 0, the compaction request from apiserver is disabled.
--etcd-count-metric-poll-period durationÂ Â Â Â Â Default: 1m0s
Frequency of polling etcd for number of resources per type. 0 disables the metric collection.
--etcd-keyfile string
SSL key file used to secure etcd communication.
--etcd-prefix stringÂ Â Â Â Â Default: "/registry"
The prefix to prepend to all resource paths in etcd.
--etcd-servers stringSlice
List of etcd servers to connect with (scheme://ip:port), comma separated.
--etcd-servers-overrides stringSlice
Per-resource etcd servers overrides, comma separated. The individual override format: group/resource#servers, where servers are URLs, semicolon separated.
--event-ttl durationÂ Â Â Â Â Default: 1h0m0s
Amount of time to retain events.
--external-hostname string
The hostname to use when generating externalized URLs for this master (e.g. Swagger API Docs).
--feature-gates mapStringBool

	A set of key=value pairs that describe feature gates for alpha/experimental features. Options are: APIListChunking=true|false (BETA - default=true) APIResponseCompression=true|false (ALPHA - default=false) AllAlpha=true|false (ALPHA - default=false) AppArmor=true|false (BETA - default=true) AttachVolumeLimit=true|false (BETA - default=true) BalanceAttachedNodeVolumes=true|false (ALPHA - default=false) BlockVolume=true|false (BETA - default=true) BoundServiceAccountTokenVolume=true|false (ALPHA - default=false) CPUManager=true|false (BETA - default=true) CRIContainerLogRotation=true|false (BETA - default=true) CSIBlockVolume=true|false (ALPHA - default=false) CSIDriverRegistry=true|false (ALPHA - default=false) CSINodeInfo=true|false (ALPHA - default=false) CustomCPUCFSQuotaPeriod=true|false (ALPHA - default=false) CustomPodDNS=true|false (BETA - default=true) CustomResourceSubresources=true|false (BETA - default=true) CustomResourceValidation=true|false (BETA - default=true) CustomResourceWebhookConversion=true|false (ALPHA - default=false) DebugContainers=true|false (ALPHA - default=false) DevicePlugins=true|false (BETA - default=true) DryRun=true|false (BETA - default=true) DynamicAuditing=true|false (ALPHA - default=false) DynamicKubeletConfig=true|false (BETA - default=true) EnableEquivalenceClassCache=true|false (ALPHA - default=false) ExpandInUsePersistentVolumes=true|false (ALPHA - default=false) ExpandPersistentVolumes=true|false (BETA - default=true) ExperimentalCriticalPodAnnotation=true|false (ALPHA - default=false) ExperimentalHostUserNamespaceDefaulting=true|false (BETA - default=false) HugePages=true|false (BETA - default=true) HyperVContainer=true|false (ALPHA - default=false) Initializers=true|false (ALPHA - default=false) KubeletPodResources=true|false (ALPHA - default=false) LocalStorageCapacityIsolation=true|false (BETA - default=true) LocalStorageCapacityIsolationFSQuotaMonitoring=true|false (ALPHA - default=false) MountContainers=true|false (ALPHA - default=false) NodeLease=true|false (ALPHA - default=false) PersistentLocalVolumes=true|false (BETA - default=true) PodPriority=true|false (BETA - default=true) PodReadinessGates=true|false (BETA - default=true) PodShareProcessNamespace=true|false (BETA - default=true) ProcMountType=true|false (ALPHA - default=false) QOSReserved=true|false (ALPHA - default=false) ResourceLimitsPriorityFunction=true|false (ALPHA - default=false) ResourceQuotaScopeSelectors=true|false (BETA - default=true) RotateKubeletClientCertificate=true|false (BETA - default=true) RotateKubeletServerCertificate=true|false (BETA - default=true) RunAsGroup=true|false (ALPHA - default=false) RuntimeClass=true|false (ALPHA - default=false)

-h, --help	
	help for kube-apiserver
--http2-max-streams-per-connection int	
	The limit that the server gives to clients for the maximum number of streams in an HTTP/2 connection. Zero means to use golang's default.
--kubelet-certificate-authority string	
	Path to a cert file for the certificate authority.
--kubelet-client-certificate string	
	Path to a client cert file for TLS.
--kubelet-client-key string	
	Path to a client key file for TLS.
--kubelet-httpsÂ Â Â Â Â Default: true	
	Use https for kubelet connections.
--kubelet-preferred-address-types stringSliceÂ Â Â Â Â Default: [Hostname,InternalDNS,InternalIP,ExternalDNS,ExternalIP]	
	List of the preferred NodeAddressTypes to use for kubelet connections.
--kubelet-read-only-port uintÂ Â Â Â Â Default: 10255	
	DEPRECATED: kubelet port.
--kubelet-timeout durationÂ Â Â Â Â Default: 5s	
	Timeout for kubelet operations.
--kubernetes-service-node-port int	
	If non-zero, the Kubernetes master service (which apiserver creates/maintains) will be of type NodePort, using this as the value of the port. If zero, the Kubernetes master service will be of type ClusterIP.
--log-backtrace-at traceLocationÂ Â Â Â Â Default: :0	
	when logging hits line file:N, emit a stack trace
--log-dir string	
	If non-empty, write log files in this directory
--log-file string	
	If non-empty, use this log file
--log-flush-frequency durationÂ Â Â Â Â Default: 5s	
	Maximum number of seconds between log flushes
--logtostderrÂ Â Â Â Â Default: true	
	log to standard error instead of files
--master-service-namespace stringÂ Â Â Â Â Default: "default"	
	DEPRECATED: the namespace from which the kubernetes master services should be injected into pods.
--max-connection-bytes-per-sec int	
	If non-zero, throttle each user connection to this number of bytes/sec. Currently only applies to long-running requests.
--max-mutating-requests-inflight intÂ Â Â Â Â Default: 200	
	The maximum number of mutating requests in flight at a given time. When the server exceeds this, it rejects requests. Zero for no limit.
--max-requests-inflight intÂ Â Â Â Â Default: 400	

	The maximum number of non-mutating requests in flight at a given time. When the server exceeds this, it rejects requests. Zero for no limit.
--min-request-timeout intÂ Â Â Â Â Default: 1800	
	An optional field indicating the minimum number of seconds a handler must keep a request open before timing it out. Currently only honored by the watch request handler, which picks a randomized value above this number as the connection timeout, to spread out load.
--oidc-ca-file string	
	If set, the OpenID server's certificate will be verified by one of the authorities in the oidc-ca-file, otherwise the host's root CA set will be used.
--oidc-client-id string	
	The client ID for the OpenID Connect client, must be set if oidc-issuer-url is set.
--oidc-groups-claim string	
	If provided, the name of a custom OpenID Connect claim for specifying user groups. The claim value is expected to be a string or array of strings. This flag is experimental, please see the authentication documentation for further details.
--oidc-groups-prefix string	
	If provided, all groups will be prefixed with this value to prevent conflicts with other authentication strategies.
--oidc-issuer-url string	
	The URL of the OpenID issuer, only HTTPS scheme will be accepted. If set, it will be used to verify the OIDC JSON Web Token (JWT).
--oidc-required-claim mapStringString	
	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.
--oidc-signing-algs stringSliceÂ Â Â Â Â Default: [RS256]	
	Comma-separated list of allowed JOSE asymmetric signing algorithms. JWTs with a 'alg' header value not in this list will be rejected. Values are defined by RFC 7518 https://tools.ietf.org/html/rfc7518#section-3.1.
--oidc-username-claim stringÂ Â Â Â Â Default: "sub"	
	The OpenID claim to use as the user name. Note that claims other than the default ('sub') is not guaranteed to be unique and immutable. This flag is experimental, please see the authentication documentation for further details.
--oidc-username-prefix string	
	If provided, all usernames will be prefixed with this value. If not provided, username claims other than 'email' are prefixed by the issuer URL to avoid clashes. To skip any prefixing, provide the value '-'.
--profilingÂ Â Â Â Â Default: true	
	Enable profiling via web interface host:port/debug/pprof/
--proxy-client-cert-file string	

	Client certificate used to prove the identity of the aggregator or kube-apiserver when it must call out during a request. This includes proxying requests to a user api-server and calling out to webhook admission plugins. It is expected that this cert includes a signature from the CA in the --requestheader-client-ca-file flag. That CA is published in the 'extension-apiserver-authentication' configmap in the kube-system namespace. Components receiving calls from kube-aggregator should use that CA to perform their half of the mutual TLS verification.
--proxy-client-key-file string	
	Private key for the client certificate used to prove the identity of the aggregator or kube-apiserver when it must call out during a request. This includes proxying requests to a user api-server and calling out to webhook admission plugins.
--request-timeout durationÂ Â Â Â Â Default: 1m0s	
	An optional field indicating the duration a handler must keep a request open before timing it out. This is the default request timeout for requests but may be overridden by flags such as --min-request-timeout for specific types of requests.
--requestheader-allowed-names stringSlice	
	List of client certificate common names to allow to provide usernames in headers specified by --requestheader-username-headers. If empty, any client certificate validated by the authorities in --requestheader-client-ca-file is allowed.
--requestheader-client-ca-file string	
	Root certificate bundle to use to verify client certificates on incoming requests before trusting usernames in headers specified by --requestheader-username-headers. WARNING: generally do not depend on authorization being already done for incoming requests.
--requestheader-extra-headers-prefix stringSlice	
	List of request header prefixes to inspect. X-Remote-Extra- is suggested.
--requestheader-group-headers stringSlice	
	List of request headers to inspect for groups. X-Remote-Group is suggested.
--requestheader-username-headers stringSlice	
	List of request headers to inspect for usernames. X-Remote-User is common.
--runtime-config mapStringString	
	A set of key=value pairs that describe runtime configuration that may be passed to apiserver. <group>/<version> (or <version> for the core group) key can be used to turn on/off specific api versions. api/all is special key to control all api versions, be careful setting it false, unless you know what you do. api/legacy is deprecated, we will remove it in the future, so stop using it.
--secure-port intÂ Â Â Â Â Default: 6443	
	The port on which to serve HTTPS with authentication and authorization.It cannot be switched off with 0.
--service-account-issuer string	
	Identifier of the service account token issuer. The issuer will assert this identifier in "iss" claim of issued tokens. This value is a string or URI.
--service-account-key-file stringArray	
	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. Must be specified when --service-account-signing-key is provided
--service-account-lookupÂ Â Â Â Â Default: true	

	If true, validate ServiceAccount tokens exist in etcd as part of authentication.
--service-account-max-token-expiration duration	
	The maximum validity duration of a token created by the service account token issuer. If an otherwise valid TokenRequest with a validity duration larger than this value is requested, a token will be issued with a validity duration of this value.
--service-account-signing-key-file string	
	Path to the file that contains the current private key of the service account token issuer. The issuer will sign issued ID tokens with this private key. (Requires the 'TokenRequest' feature gate.)
--service-cluster-ip-range ipNetÂ Â Â Â Â Default: 10.0.0.0/24	
	A CIDR notation IP range from which to assign service cluster IPs. This must not overlap with any IP ranges assigned to nodes for pods.
--service-node-port-range portRangeÂ Â Â Â Â Default: 30000-32767	
	A port range to reserve for services with NodePort visibility. Example: '30000-32767'. Inclusive at both ends of the range.
--skip-headers	
	If true, avoid header prefixes in the log messages
--stderrthreshold severityÂ Â Â Â Â Default: 2	
	logs at or above this threshold go to stderr
--storage-backend string	
	The storage backend for persistence. Options: 'etcd3' (default).
--storage-media-type stringÂ Â Â Â Â Default: "application/vnd.kubernetes.protobuf"	
	The media type to use to store objects in storage. Some resources or storage backends may only support a specific media type and will ignore this setting.
--target-ram-mb int	
	Memory limit for apiserver in MB (used to configure sizes of caches, etc.)
--tls-cert-file string	
	File containing the default x509 Certificate for HTTPS. (CA cert, if any, concatenated after server cert). If HTTPS serving is enabled, and --tls-cert-file and --tls-private-key-file are not provided, a self-signed certificate and key are generated for the public address and saved to the directory specified by --cert-dir.
--tls-cipher-suites stringSlice	

	Comma-separated list of cipher suites for the server. If omitted, the default Go cipher suites will be used. Possible values: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_RC4_128_SHA,TLS_RSA_WITH_3DES_EDE_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_RC4_128_SHA
--tls-min-version string	
	Minimum TLS version supported. Possible values: VersionTLS10, VersionTLS11, VersionTLS12
--tls-private-key-file string	
	File containing the default x509 private key matching --tls-cert-file.
--tls-sni-cert-key namedCertKeyÂ Â Â Â Â Default: []	
	A pair of x509 certificate and private key file paths, optionally suffixed with a list of domain patterns which are fully qualified domain names, possibly with prefixed wildcard segments. If no domain patterns are provided, the names of the certificate are extracted. Non-wildcard matches trump over wildcard matches, explicit domain patterns trump over extracted names. For multiple key/certificate pairs, use the --tls-sni-cert-key multiple times. Examples: "example.crt,example.key" or "foo.crt,foo.key:*.foo.com,foo.com".
--token-auth-file string	
	If set, the file that will be used to secure the secure port of the API server via token authentication.
-v, --v Level	
	number for the log level verbosity
--version version[=true]	
	Print version information and quit
--vmodule moduleSpec	
	comma-separated list of pattern=N settings for file-filtered logging
--watch-cacheÂ Â Â Â Â Default: true	
	Enable watch caching in the apiserver
--watch-cache-sizes stringSlice	
	Watch cache size settings for some resources (pods, nodes, etc.), comma separated. The individual setting format: resource[.group]#size, where resource is lowercase plural (no version), group is omitted for resources of apiVersion v1 (the legacy core API) and included for others, and size is a number. It takes effect when watch-cache is enabled. Some resources (replicationcontrollers, endpoints, nodes, pods, services, apiservices.apiregistration.k8s.io) have system defaults set by heuristics, others default to default-watch-cache-size

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on July 09, 2019 at 8:48 AM PST by Grammar Correction (#15354) (Page History)

Edit This Page

- - kube-controller-manager
    - Synopsis
    - Options

kube-controller-manager

Synopsis

The Kubernetes controller manager is a daemon that embeds the core control loops shipped with Kubernetes. In applications of robotics and automation, a control loop is a non-terminating loop that regulates the state of the system. In Kubernetes, a controller is a control loop that watches the shared state of the cluster through the apiserver and makes changes attempting to move the current state towards the desired state. Examples of controllers that ship with Kubernetes today are the replication controller, endpoints controller, namespace controller, and serviceaccounts controller.

```
kube-controller-manager [flags]
```

Options

--allocate-node-cidrs	
	Should CIDRs for Pods be allocated and set on the cloud provider.
--alsologtostderr	
	log to standard error as well as files
--attach-detach-reconcile-sync-period durationÂ Â Â Â Â Default: 1m0s	
	The reconciler sync wait time between volume attach detach. This duration must be larger than one second, and increasing this value from the default may allow for volumes to be mismatched with pods.
--authentication-kubeconfig string	

	kubeconfig file pointing at the 'core' kubernetes server with enough rights to create tokenaccessreviews.authentication.k8s.io. This is optional. If empty, all token requests are considered to be anonymous and no client CA is looked up in the cluster.
--authentication-skip-lookup	
	If false, the authentication-kubeconfig will be used to lookup missing authentication configuration from the cluster.
--authentication-token-webhook-cache-ttl durationÂ Â Â Â Â Default: 10s	
	The duration to cache responses from the webhook token authenticator.
--authentication-tolerate-lookup-failure	
	If true, failures to look up missing authentication configuration from the cluster are not considered fatal. Note that this can result in authentication that treats all requests as anonymous.
--authorization-always-allow-paths stringSliceÂ Â Â Â Â Default: [/healthz]	
	A list of HTTP paths to skip during authorization, i.e. these are authorized without contacting the 'core' kubernetes server.
--authorization-kubeconfig string	
	kubeconfig file pointing at the 'core' kubernetes server with enough rights to create subjectaccessreviews.authorization.k8s.io. This is optional. If empty, all requests not skipped by authorization are forbidden.
--authorization-webhook-cache-authorized-ttl durationÂ Â Â Â Â Default: 10s	
	The duration to cache 'authorized' responses from the webhook authorizer.
--authorization-webhook-cache-unauthorized-ttl durationÂ Â Â Â Â Default: 10s	
	The duration to cache 'unauthorized' responses from the webhook authorizer.
--azure-container-registry-config string	
	Path to the file containing Azure container registry configuration information.
--bind-address ipÂ Â Â Â Â Default: 0.0.0.0	
	The IP address on which to listen for the --secure-port port. The associated interface(s) must be reachable by the rest of the cluster, and by CLI/web clients. If blank, all interfaces will be used (0.0.0.0 for all IPv4 interfaces and :: for all IPv6 interfaces).
--cert-dir string	
	The directory where the TLS certs are located. If --tls-cert-file and --tls-private-key-file are provided, this flag will be ignored.
--cidr-allocator-type stringÂ Â Â Â Â Default: "RangeAllocator"	
	Type of CIDR allocator to use
--client-ca-file string	
	If set, any request presenting a client certificate signed by one of the authorities in the client-ca-file is authenticated with an identity corresponding to the CommonName of the client certificate.
--cloud-config string	
	The path to the cloud provider configuration file. Empty string for no configuration file.
--cloud-provider string	
	The provider for cloud services. Empty string for no provider.
--cluster-cidr string	
	CIDR Range for Pods in cluster. Requires --allocate-node-cidrs to be true

--cluster-name stringÂ Â Â Â Â Default: "kubernetes"	
	The instance prefix for the cluster.
--cluster-signing-cert-file stringÂ Â Â Â Â Default: "/etc/kubernetes/ca/ca.pem"	
	Filename containing a PEM-encoded X509 CA certificate used to issue cluster-scoped certificates
--cluster-signing-key-file stringÂ Â Â Â Â Default: "/etc/kubernetes/ca/ca.key"	
	Filename containing a PEM-encoded RSA or ECDSA private key used to sign cluster-scoped certificates
--concurrent-deployment-syncs int32Â Â Â Â Â Default: 5	
	The number of deployment objects that are allowed to sync concurrently. Larger number = more responsive deployments, but more CPU (and network) load
--concurrent-endpoint-syncs int32Â Â Â Â Â Default: 5	
	The number of endpoint syncing operations that will be done concurrently. Larger number = faster endpoint updating, but more CPU (and network) load
--concurrent-gc-syncs int32Â Â Â Â Â Default: 20	
	The number of garbage collector workers that are allowed to sync concurrently.
--concurrent-namespace-syncs int32Â Â Â Â Â Default: 10	
	The number of namespace objects that are allowed to sync concurrently. Larger number = more responsive namespace termination, but more CPU (and network) load
--concurrent-replicaset-syncs int32Â Â Â Â Â Default: 5	
	The number of replica sets that are allowed to sync concurrently. Larger number = more responsive replica management, but more CPU (and network) load
--concurrent-resource-quota-syncs int32Â Â Â Â Â Default: 5	
	The number of resource quotas that are allowed to sync concurrently. Larger number = more responsive quota management, but more CPU (and network) load
--concurrent-service-syncs int32Â Â Â Â Â Default: 1	
	The number of services that are allowed to sync concurrently. Larger number = more responsive service management, but more CPU (and network) load
--concurrent-serviceaccount-token-syncs int32Â Â Â Â Â Default: 5	
	The number of service account token objects that are allowed to sync concurrently. Larger number = more responsive token generation, but more CPU (and network) load
--concurrent-ttl-after-finished-syncs int32Â Â Â Â Â Default: 5	
	The number of TTL-after-finished controller workers that are allowed to sync concurrently.
--concurrent_rc_syncs int32Â Â Â Â Â Default: 5	
	The number of replication controllers that are allowed to sync concurrently. Larger number = more responsive replica management, but more CPU (and network) load
--configure-cloud-routesÂ Â Â Â Â Default: true	
	Should CIDRs allocated by allocate-node-cidrs be configured on the cloud provider.
--contention-profiling	
	Enable lock contention profiling, if profiling is enabled
--controller-start-interval duration	
	Interval between starting controller managers.
--controllers stringSliceÂ Â Â Â Â Default: [*]	

	A list of controllers to enable. '*' enables all on-by-default controllers, 'foo' enables the controller named 'foo', '-foo' disables the controller named 'foo'. All controllers: attachdetach, bootstrapsigner, cloud-node-lifecycle, clusterrole-aggregation, cronjob, csrapproving, csrcleaner, csrsigning, daemonset, deployment, disruption, endpoint, garbagecollector, horizontalpodautoscaling, job, namespace, nodeipam, nodelifecycle, persistentvolume-binder, persistentvolume-expander, podgc, pv-protection, pvc-protection, replicaset, replicationcontroller, resourcequota, root-ca-cert-publisher, route, service, serviceaccount, serviceaccount-token, statefulset, tokencleaner, ttl, ttl-after-finished Disabled-by-default controllers: bootstrapsigner, tokencleaner
--deployment-controller-sync-period durationÂ Â Â Â Â Default: 30s	
	Period for syncing the deployments.
--disable-attach-detach-reconcile-sync	
	Disable volume attach detach reconciler sync. Disabling this may cause volumes to be mismatched with pods. Use wisely.
--enable-dynamic-provisioningÂ Â Â Â Â Default: true	
	Enable dynamic provisioning for environments that support it.
--enable-garbage-collectorÂ Â Â Â Â Default: true	
	Enables the generic garbage collector. MUST be synced with the corresponding flag of the kube-apiserver.
--enable-hostpath-provisioner	
	Enable HostPath PV provisioning when running without a cloud provider. This allows testing and development of provisioning features. HostPath provisioning is not supported in any way, won't work in a multi-node cluster, and should not be used for anything other than testing or development.
--enable-taint-managerÂ Â Â Â Â Default: true	
	WARNING: Beta feature. If set to true enables NoExecute Taints and will evict all not-tolerating Pod running on Nodes tainted with this kind of Taints.
--experimental-cluster-signing-duration durationÂ Â Â Â Â Default: 8760h0m0s	
	The length of duration signed certificates will be given.
--external-cloud-volume-plugin string	
	The plugin to use when cloud provider is set to external. Can be empty, should only be set when cloud-provider is external. Currently used to allow node and volume controllers to work for in tree cloud providers.
--feature-gates mapStringBool	

	A set of key=value pairs that describe feature gates for alpha/experimental features. Options are: APIListChunking=true|false (BETA - default=true) APIResponseCompression=true|false (ALPHA - default=false) AllAlpha=true|false (ALPHA - default=false) AppArmor=true|false (BETA - default=true) AttachVolumeLimit=true|false (BETA - default=true) BalanceAttachedNodeVolumes=true|false (ALPHA - default=false) BlockVolume=true|false (BETA - default=true) BoundServiceAccountTokenVolume=true|false (ALPHA - default=false) CPUManager=true|false (BETA - default=true) CRIContainerLogRotation=true|false (BETA - default=true) CSIBlockVolume=true|false (ALPHA - default=false) CSIDriverRegistry=true|false (ALPHA - default=false) CSINodeInfo=true|false (ALPHA - default=false) CustomCPUCFSQuotaPeriod=true|false (ALPHA - default=false) CustomPodDNS=true|false (BETA - default=true) CustomResourceSubresources=true|false (BETA - default=true) CustomResourceValidation=true|false (BETA - default=true) CustomResourceWebhookConversion=true|false (ALPHA - default=false) DebugContainers=true|false (ALPHA - default=false) DevicePlugins=true|false (BETA - default=true) DryRun=true|false (BETA - default=true) DynamicAuditing=true|false (ALPHA - default=false) DynamicKubeletConfig=true|false (BETA - default=true) EnableEquivalenceClassCache=true|false (ALPHA - default=false) ExpandInUsePersistentVolumes=true|false (ALPHA - default=false) ExpandPersistentVolumes=true|false (BETA - default=true) ExperimentalCriticalPodAnnotation=true|false (ALPHA - default=false) ExperimentalHostUserNamespaceDefaulting=true|false (BETA - default=false) HugePages=true|false (BETA - default=true) HyperVContainer=true|false (ALPHA - default=false) Initializers=true|false (ALPHA - default=false) KubeletPodResources=true|false (ALPHA - default=false) LocalStorageCapacityIsolation=true|false (BETA - default=true) LocalStorageCapacityIsolationFSQuotaMonitoring=true|false (ALPHA - default=false) MountContainers=true|false (ALPHA - default=false) NodeLease=true|false (ALPHA - default=false) PersistentLocalVolumes=true|false (BETA - default=true) PodPriority=true|false (BETA - default=true) PodReadinessGates=true|false (BETA - default=true) PodShareProcessNamespace=true|false (BETA - default=true) ProcMountType=true|false (ALPHA - default=false) QOSReserved=true|false (ALPHA - default=false) ResourceLimitsPriorityFunction=true|false (ALPHA - default=false) ResourceQuotaScopeSelectors=true|false (BETA - default=true) RotateKubeletClientCertificate=true|false (BETA - default=true) RotateKubeletServerCertificate=true|false (BETA - default=true) RunAsGroup=true|false (ALPHA - default=false) RuntimeClass=true|false (ALPHA - default=false)

--flex-volume-plugin-dir stringÂ Â Â Â Â Default: "/usr/libexec/kubernetes/kubelet-plugins/volume/exec/"
Full path of the directory in which the flex volume plugin should search for additional third party volume plugins.
-h, --help
help for kube-controller-manager
--horizontal-pod-autoscaler-cpu-initialization-period durationÂ Â Â Â Â Default: 5m0s
The period after pod start when CPU samples might be skipped.
--horizontal-pod-autoscaler-downscale-stabilization durationÂ Â Â Â Â Default: 5m0s
The period for which autoscaler will look backwards and not scale down below any recommendation it made during that period.
--horizontal-pod-autoscaler-initial-readiness-delay durationÂ Â Â Â Â Default: 30s
The period after pod start during which readiness changes will be treated as initial readiness.
--horizontal-pod-autoscaler-sync-period durationÂ Â Â Â Â Default: 15s
The period for syncing the number of pods in horizontal pod autoscaler.
--horizontal-pod-autoscaler-tolerance floatÂ Â Â Â Â Default: 0.1
The minimum change (from 1.0) in the desired-to-actual metrics ratio for the horizontal pod autoscaler to consider scaling.
--http2-max-streams-per-connection int
The limit that the server gives to clients for the maximum number of streams in an HTTP/2 connection. Zero means to use golang's default.
--kube-api-burst int32Â Â Â Â Â Default: 30
Burst to use while talking with kubernetes apiserver.
--kube-api-content-type stringÂ Â Â Â Â Default: "application/vnd.kubernetes.protobuf"
Content type of requests sent to apiserver.
--kube-api-qps float32Â Â Â Â Â Default: 20
QPS to use while talking with kubernetes apiserver.
--kubeconfig string
Path to kubeconfig file with authorization and master location information.
--large-cluster-size-threshold int32Â Â Â Â Â Default: 50
Number of nodes from which NodeController treats the cluster as large for the eviction logic purposes. --secondary-node-eviction-rate is implicitly overridden to 0 for clusters this size or smaller.
--leader-electÂ Â Â Â Â Default: true
Start a leader election client and gain leadership before executing the main loop. Enable this when running replicated components for high availability.
--leader-elect-lease-duration durationÂ Â Â Â Â Default: 15s
The duration that non-leader candidates will wait after observing a leadership renewal until attempting to acquire leadership of a led but unrenewed leader slot. This is effectively the maximum duration that a leader can be stopped before it is replaced by another candidate. This is only applicable if leader election is enabled.
--leader-elect-renew-deadline durationÂ Â Â Â Â Default: 10s

	The interval between attempts by the acting master to renew a leadership slot before it stops leading. This must be less than or equal to the lease duration. This is only applicable if leader election is enabled.
--leader-elect-resource-lock endpointsÂ Â Â Â Â Default: "endpoints"	
	The type of resource object that is used for locking during leader election. Supported options are endpoints (default) and `configmaps`.
--leader-elect-retry-period durationÂ Â Â Â Â Default: 2s	
	The duration the clients should wait between attempting acquisition and renewal of a leadership. This is only applicable if leader election is enabled.
--log-backtrace-at traceLocationÂ Â Â Â Â Default: :0	
	when logging hits line file:N, emit a stack trace
--log-dir string	
	If non-empty, write log files in this directory
--log-file string	
	If non-empty, use this log file
--log-flush-frequency durationÂ Â Â Â Â Default: 5s	
	Maximum number of seconds between log flushes
--logtostderrÂ Â Â Â Â Default: true	
	log to standard error instead of files
--master string	
	The address of the Kubernetes API server (overrides any value in kubeconfig).
--min-resync-period durationÂ Â Â Â Â Default: 12h0m0s	
	The resync period in reflectors will be random between MinResyncPeriod and 2*MinResyncPeriod.
--namespace-sync-period durationÂ Â Â Â Â Default: 5m0s	
	The period for syncing namespace life-cycle updates
--node-cidr-mask-size int32Â Â Â Â Â Default: 24	
	Mask size for node cidr in cluster.
--node-eviction-rate float32Â Â Â Â Â Default: 0.1	
	Number of nodes per second on which pods are deleted in case of node failure when a zone is healthy (see --unhealthy-zone-threshold for definition of healthy/unhealthy). Zone refers to entire cluster in non-multizone clusters.
--node-monitor-grace-period durationÂ Â Â Â Â Default: 40s	
	Amount of time which we allow running Node to be unresponsive before marking it unhealthy. Must be N times more than kubelet's nodeStatusUpdateFrequency, where N means number of retries allowed for kubelet to post node status.
--node-monitor-period durationÂ Â Â Â Â Default: 5s	
	The period for syncing NodeStatus in NodeController.
--node-startup-grace-period durationÂ Â Â Â Â Default: 1m0s	
	Amount of time which we allow starting Node to be unresponsive before marking it unhealthy.
--pod-eviction-timeout durationÂ Â Â Â Â Default: 5m0s	
	The grace period for deleting pods on failed nodes.
--profiling	
	Enable profiling via web interface host:port/debug/pprof/

--pv-recycler-increment-timeout-nfs int32Â Â Â Â Â Default: 30	
	the increment of time added per Gi to ActiveDeadlineSeconds for an NFS scrubber pod
--pv-recycler-minimum-timeout-hostpath int32Â Â Â Â Â Default: 60	
	The minimum ActiveDeadlineSeconds to use for a HostPath Recycler pod. This is for development and testing only and will not work in a multi-node cluster.
--pv-recycler-minimum-timeout-nfs int32Â Â Â Â Â Default: 300	
	The minimum ActiveDeadlineSeconds to use for an NFS Recycler pod
--pv-recycler-pod-template-filepath-hostpath string	
	The file path to a pod definition used as a template for HostPath persistent volume recycling. This is for development and testing only and will not work in a multi-node cluster.
--pv-recycler-pod-template-filepath-nfs string	
	The file path to a pod definition used as a template for NFS persistent volume recycling
--pv-recycler-timeout-increment-hostpath int32Â Â Â Â Â Default: 30	
	the increment of time added per Gi to ActiveDeadlineSeconds for a HostPath scrubber pod. This is for development and testing only and will not work in a multi-node cluster.
--pvclaimbinder-sync-period durationÂ Â Â Â Â Default: 15s	
	The period for syncing persistent volumes and persistent volume claims
--requestheader-allowed-names stringSlice	
	List of client certificate common names to allow to provide usernames in headers specified by --requestheader-username-headers. If empty, any client certificate validated by the authorities in --requestheader-client-ca-file is allowed.
--requestheader-client-ca-file string	
	Root certificate bundle to use to verify client certificates on incoming requests before trusting usernames in headers specified by --requestheader-username-headers. WARNING: generally do not depend on authorization being already done for incoming requests.
--requestheader-extra-headers-prefix stringSliceÂ Â Â Â Â Default: [x-remote-extra-]	
	List of request header prefixes to inspect. X-Remote-Extra- is suggested.
--requestheader-group-headers stringSliceÂ Â Â Â Â Default: [x-remote-group]	
	List of request headers to inspect for groups. X-Remote-Group is suggested.
--requestheader-username-headers stringSliceÂ Â Â Â Â Default: [x-remote-user]	
	List of request headers to inspect for usernames. X-Remote-User is common.
--resource-quota-sync-period durationÂ Â Â Â Â Default: 5m0s	
	The period for syncing quota usage status in the system
--root-ca-file string	
	If set, this root certificate authority will be included in service account's token secret. This must be a valid PEM-encoded CA bundle.
--route-reconciliation-period durationÂ Â Â Â Â Default: 10s	
	The period for reconciling routes created for Nodes by cloud provider.
--secondary-node-eviction-rate float32Â Â Â Â Â Default: 0.01	
	Number of nodes per second on which pods are deleted in case of node failure when a zone is unhealthy (see --unhealthy-zone-threshold for definition of healthy/unhealthy). Zone refers to entire cluster in non-multizone clusters. This value is implicitly overridden to 0 if the cluster size is smaller than --large-cluster-size-threshold.
--secure-port intÂ Â Â Â Â Default: 10257	

	The port on which to serve HTTPS with authentication and authorization.If 0, don't serve HTTPS at all.
--service-account-private-key-file string	
	Filename containing a PEM-encoded private RSA or ECDSA key used to sign service account tokens.
--service-cluster-ip-range string	
	CIDR Range for Services in cluster. Requires --allocate-node-cidrs to be true
--skip-headers	
	If true, avoid header prefixes in the log messages
--stderrthreshold severityÂ Â Â Â Â Default: 2	
	logs at or above this threshold go to stderr
--terminated-pod-gc-threshold int32Â Â Â Â Â Default: 12500	
	Number of terminated pods that can exist before the terminated pod garbage collector starts deleting terminated pods. If <= 0, the terminated pod garbage collector is disabled.
--tls-cert-file string	
	File containing the default x509 Certificate for HTTPS. (CA cert, if any, concatenated after server cert). If HTTPS serving is enabled, and --tls-cert-file and --tls-private-key-file are not provided, a self-signed certificate and key are generated for the public address and saved to the directory specified by --cert-dir.
--tls-cipher-suites stringSlice	
	Comma-separated list of cipher suites for the server. If omitted, the default Go cipher suites will be use. Possible values: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_RC4_128_SHA,TLS_RSA_WITH_3DES_EDE_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_RC4_128_SHA
--tls-min-version string	
	Minimum TLS version supported. Possible values: VersionTLS10, VersionTLS11, VersionTLS12
--tls-private-key-file string	
	File containing the default x509 private key matching --tls-cert-file.
--tls-sni-cert-key namedCertKeyÂ Â Â Â Â Default: []	

	A pair of x509 certificate and private key file paths, optionally suffixed with a list of domain patterns which are fully qualified domain names, possibly with prefixed wildcard segments. If no domain patterns are provided, the names of the certificate are extracted. Non-wildcard matches trump over wildcard matches, explicit domain patterns trump over extracted names. For multiple key/certificate pairs, use the --tls-sni-cert-key multiple times. Examples: "example.crt,example.key" or "foo.crt,foo.key:*.foo.com,foo.com".
--unhealthy-zone-threshold float32Â Â Â Â Â Default: 0.55	
	Fraction of Nodes in a zone which needs to be not Ready (minimum 3) for zone to be treated as unhealthy.
--use-service-account-credentials	
	If true, use individual service account credentials for each controller.
-v, --v Level	
	number for the log level verbosity
--version version[=true]	
	Print version information and quit
--vmodule moduleSpec	
	comma-separated list of pattern=N settings for file-filtered logging

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on May 31, 2019 at 11:28 AM PST by Add support for quotas for ephemeral storage monitoring. (#14268) (Page History)

Edit This Page

-
  - kube-proxy
    - Synopsis
    - Options

kube-proxy

Synopsis

The Kubernetes network proxy runs on each node. This reflects services as defined in the Kubernetes API on each node and can do simple TCP, UDP, and SCTP stream forwarding or round robin TCP, UDP, and SCTP forwarding across a set of backends. Service cluster IPs and ports are currently found through Docker-links-compatible environment variables specifying ports opened by the service proxy. There is an optional addon that provides cluster DNS for these cluster IPs. The user must create a service with the apiserver API to configure the proxy.

```
kube-proxy [flags]
```

Options

--azure-container-registry-config string	
	Path to the file containing Azure container registry configuration information.
--bind-address 0.0.0.0Â Â Â Â Â Default: 0.0.0.0	
	The IP address for the proxy server to serve on (set to 0.0.0.0 for all IPv4 interfaces and `::` for all IPv6 interfaces)
--cleanup	
	If true cleanup iptables and ipvs rules and exit.
--cleanup-ipvsÂ Â Â Â Â Default: true	
	If true make kube-proxy cleanup ipvs rules before running. Default is true
--cluster-cidr string	
	The CIDR range of pods in the cluster. When configured, traffic sent to a Service cluster IP from outside this range will be masqueraded and traffic sent from pods to an external LoadBalancer IP will be directed to the respective cluster IP instead
--config string	
	The path to the configuration file.
--config-sync-period durationÂ Â Â Â Â Default: 15m0s	
	How often configuration from the apiserver is refreshed. Must be greater than 0.
--conntrack-max-per-core int32Â Â Â Â Â Default: 32768	
	Maximum number of NAT connections to track per CPU core (0 to leave the limit as-is and ignore conntrack-min).
--conntrack-min int32Â Â Â Â Â Default: 131072	
	Minimum number of conntrack entries to allocate, regardless of conntrack-max-per-core (set conntrack-max-per-core=0 to leave the limit as-is).
--conntrack-tcp-timeout-close-wait durationÂ Â Â Â Â Default: 1h0m0s	
	NAT timeout for TCP connections in the CLOSE_WAIT state
--conntrack-tcp-timeout-established durationÂ Â Â Â Â Default: 24h0m0s	
	Idle timeout for established TCP connections (0 to leave as-is)
--feature-gates mapStringBool	

	A set of key=value pairs that describe feature gates for alpha/experimental features. Options are: APIListChunking=true|false (BETA - default=true) APIResponseCompression=true|false (ALPHA - default=false) AllAlpha=true|false (ALPHA - default=false) AppArmor=true|false (BETA - default=true) AttachVolumeLimit=true|false (BETA - default=true) BalanceAttachedNodeVolumes=true|false (ALPHA - default=false) BlockVolume=true|false (BETA - default=true) BoundServiceAccountTokenVolume=true|false (ALPHA - default=false) CPUManager=true|false (BETA - default=true) CRIContainerLogRotation=true|false (BETA - default=true) CSIBlockVolume=true|false (ALPHA - default=false) CSIDriverRegistry=true|false (ALPHA - default=false) CSINodeInfo=true|false (ALPHA - default=false) CustomCPUCFSQuotaPeriod=true|false (ALPHA - default=false) CustomPodDNS=true|false (BETA - default=true) CustomResourceSubresources=true|false (BETA - default=true) CustomResourceValidation=true|false (BETA - default=true) CustomResourceWebhookConversion=true|false (ALPHA - default=false) DebugContainers=true|false (ALPHA - default=false) DevicePlugins=true|false (BETA - default=true) DryRun=true|false (BETA - default=true) DynamicAuditing=true|false (ALPHA - default=false) DynamicKubeletConfig=true|false (BETA - default=true) EnableEquivalenceClassCache=true|false (ALPHA - default=false) ExpandInUsePersistentVolumes=true|false (ALPHA - default=false) ExpandPersistentVolumes=true|false (BETA - default=true) ExperimentalCriticalPodAnnotation=true|false (ALPHA - default=false) ExperimentalHostUserNamespaceDefaulting=true|false (BETA - default=false) HugePages=true|false (BETA - default=true) HyperVContainer=true|false (ALPHA - default=false) Initializers=true|false (ALPHA - default=false) KubeletPodResources=true|false (ALPHA - default=false) LocalStorageCapacityIsolation=true|false (BETA - default=true) LocalStorageCapacityIsolationFSQuotaMonitoring=true|false (ALPHA - default=false) MountContainers=true|false (ALPHA - default=false) NodeLease=true|false (ALPHA - default=false) PersistentLocalVolumes=true|false (BETA - default=true) PodPriority=true|false (BETA - default=true) PodReadinessGates=true|false (BETA - default=true) PodShareProcessNamespace=true|false (BETA - default=true) ProcMountType=true|false (ALPHA - default=false) QOSReserved=true|false (ALPHA - default=false) ResourceLimitsPriorityFunction=true|false (ALPHA - default=false) ResourceQuotaScopeSelectors=true|false (BETA - default=true) RotateKubeletClientCertificate=true|false (BETA - default=true) RotateKubeletServerCertificate=true|false (BETA - default=true) RunAsGroup=true|false (ALPHA - default=false) RuntimeClass=true|false (ALPHA - default=false)

--healthz-bind-address 0.0.0.0Â Â Â Â Â Default: 0.0.0.0:10256	
	The IP address for the health check server to serve on (set to 0.0.0.0 for all IPv4 interfaces and `::` for all IPv6 interfaces)
--healthz-port int32Â Â Â Â Â Default: 10256	
	The port to bind the health check server. Use 0 to disable.
-h, --help	
	help for kube-proxy
--hostname-override string	
	If non-empty, will use this string as identification instead of the actual hostname.
--iptables-masquerade-bit int32Â Â Â Â Â Default: 14	
	If using the pure iptables proxy, the bit of the fwmark space to mark packets requiring SNAT with. Must be within the range [0, 31].
--iptables-min-sync-period duration	
	The minimum interval of how often the iptables rules can be refreshed as endpoints and services change (e.g. '5s', '1m', '2h22m').
--iptables-sync-period durationÂ Â Â Â Â Default: 30s	
	The maximum interval of how often iptables rules are refreshed (e.g. '5s', '1m', '2h22m'). Must be greater than 0.
--ipvs-exclude-cidrs stringSlice	
	A comma-separated list of CIDR's which the ipvs proxier should not touch when cleaning up IPVS rules.
--ipvs-min-sync-period duration	
	The minimum interval of how often the ipvs rules can be refreshed as endpoints and services change (e.g. '5s', '1m', '2h22m').
--ipvs-scheduler string	
	The ipvs scheduler type when proxy mode is ipvs
--ipvs-sync-period durationÂ Â Â Â Â Default: 30s	
	The maximum interval of how often ipvs rules are refreshed (e.g. '5s', '1m', '2h22m'). Must be greater than 0.
--kube-api-burst int32Â Â Â Â Â Default: 10	
	Burst to use while talking with kubernetes apiserver
--kube-api-content-type stringÂ Â Â Â Â Default: "application/vnd.kubernetes.protobuf"	
	Content type of requests sent to apiserver.
--kube-api-qps float32Â Â Â Â Â Default: 5	
	QPS to use while talking with kubernetes apiserver
--kubeconfig string	
	Path to kubeconfig file with authorization information (the master location is set by the master flag).
--log-flush-frequency durationÂ Â Â Â Â Default: 5s	
	Maximum number of seconds between log flushes
--masquerade-all	
	If using the pure iptables proxy, SNAT all traffic sent via Service cluster IPs (this not commonly needed)
--master string	

	The address of the Kubernetes API server (overrides any value in kubeconfig)
--metrics-bind-address 0.0.0.0Â Â Â Â Â Default: 127.0.0.1:10249	
	The IP address for the metrics server to serve on (set to 0.0.0.0 for all IPv4 interfaces and `::` for all IPv6 interfaces)
--metrics-port int32Â Â Â Â Â Default: 10249	
	The port to bind the metrics server. Use 0 to disable.
--nodeport-addresses stringSlice	
	A string slice of values which specify the addresses to use for NodePorts. Values may be valid IP blocks (e.g. 1.2.3.0/24, 1.2.3.4/32). The default empty string slice ([]) means to use all local addresses.
--oom-score-adj int32Â Â Â Â Â Default: -999	
	The oom-score-adj value for kube-proxy process. Values must be within the range [-1000, 1000]
--profiling	
	If true enables profiling via web interface on /debug/pprof handler.
--proxy-mode ProxyMode	
	Which proxy mode to use: 'userspace' (older) or 'iptables' (faster) or 'ipvs' (experimental). If blank, use the best-available proxy (currently iptables). If the iptables proxy is selected, regardless of how, but the system's kernel or iptables versions are insufficient, this always falls back to the userspace proxy.
--proxy-port-range port-range	
	Range of host ports (beginPort-endPort, single port or beginPort+offset, inclusive) that may be consumed in order to proxy service traffic. If (unspecified, 0, or 0-0) then ports will be randomly chosen.
--udp-timeout durationÂ Â Â Â Â Default: 250ms	
	How long an idle UDP connection will be kept open (e.g. '250ms', '2s'). Must be greater than 0. Only applicable for proxy-mode=userspace
--version version[=true]	
	Print version information and quit
--write-config-to string	
	If set, write the default configuration values to this file and exit.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page

Page last modified on May 31, 2019 at 11:28 AM PST by Add support for quotas for ephemeral storage monitoring. (#14268) (Page History)

Edit This Page

- - kube-scheduler
    - Synopsis
    - Options

kube-scheduler

Synopsis

The Kubernetes scheduler is a policy-rich, topology-aware, workload-specific function that significantly impacts availability, performance, and capacity. The scheduler needs to take into account individual and collective resource requirements, quality of service requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, deadlines, and so on. Workload-specific requirements will be exposed through the API as necessary.

```
kube-scheduler [flags]
```

Options

--address stringÂ Â Â Â Â Default: "0.0.0.0"	
	DEPRECATED: the IP address on which to listen for the --port port (set to 0.0.0.0 for all IPv4 interfaces and :: for all IPv6 interfaces). See --bind-address instead.
--algorithm-provider string	
	DEPRECATED: the scheduling algorithm provider to use, one of: ClusterAutoscalerProvider | DefaultProvider
--alsologtostderr	
	log to standard error as well as files
--authentication-kubeconfig string	
	kubeconfig file pointing at the 'core' kubernetes server with enough rights to create tokenaccessreviews.authentication.k8s.io. This is optional. If empty, all token requests are considered to be anonymous and no client CA is looked up in the cluster.
--authentication-skip-lookup	
	If false, the authentication-kubeconfig will be used to lookup missing authentication configuration from the cluster.
--authentication-token-webhook-cache-ttl durationÂ Â Â Â Â Default: 10s	
	The duration to cache responses from the webhook token authenticator.
--authentication-tolerate-lookup-failureÂ Â Â Â Â Default: true	
	If true, failures to look up missing authentication configuration from the cluster are not considered fatal. Note that this can result in authentication that treats all requests as anonymous.
--authorization-always-allow-paths stringSliceÂ Â Â Â Â Default: [/healthz]	
	A list of HTTP paths to skip during authorization, i.e. these are authorized without contacting the 'core' kubernetes server.

--authorization-kubeconfig string
kubeconfig file pointing at the 'core' kubernetes server with enough rights to create subjectaccessreviews.authorization.k8s.io. This is optional. If empty, all requests not skipped by authorization are forbidden.
--authorization-webhook-cache-authorized-ttl durationÂ Â Â Â Â Default: 10s
The duration to cache 'authorized' responses from the webhook authorizer.
--authorization-webhook-cache-unauthorized-ttl durationÂ Â Â Â Â Default: 10s
The duration to cache 'unauthorized' responses from the webhook authorizer.
--azure-container-registry-config string
Path to the file containing Azure container registry configuration information.
--bind-address ipÂ Â Â Â Â Default: 0.0.0.0
The IP address on which to listen for the --secure-port port. The associated interface(s) must be reachable by the rest of the cluster, and by CLI/web clients. If blank, all interfaces will be used (0.0.0.0 for all IPv4 interfaces and :: for all IPv6 interfaces).
--cert-dir string
The directory where the TLS certs are located. If --tls-cert-file and --tls-private-key-file are provided, this flag will be ignored.
--client-ca-file string
If set, any request presenting a client certificate signed by one of the authorities in the client-ca-file is authenticated with an identity corresponding to the CommonName of the client certificate.
--config string
The path to the configuration file. Flags override values in this file.
--contention-profiling
DEPRECATED: enable lock contention profiling, if profiling is enabled
--feature-gates mapStringBool

A set of key=value pairs that describe feature gates for alpha/experimental features. Options are:
APIListChunking=true|false (BETA - default=true)
APIResponseCompression=true|false (ALPHA - default=false)
AllAlpha=true|false (ALPHA - default=false)
AppArmor=true|false (BETA - default=true)
AttachVolumeLimit=true|false (BETA - default=true)
BalanceAttachedNodeVolumes=true|false (ALPHA - default=false)
BlockVolume=true|false (BETA - default=true)
BoundServiceAccountTokenVolume=true|false (ALPHA - default=false)
CPUManager=true|false (BETA - default=true)
CRIContainerLogRotation=true|false (BETA - default=true)
CSIBlockVolume=true|false (ALPHA - default=false)
CSIDriverRegistry=true|false (ALPHA - default=false)
CSINodeInfo=true|false (ALPHA - default=false)
CustomCPUCFSQuotaPeriod=true|false (ALPHA - default=false)
CustomPodDNS=true|false (BETA - default=true)
CustomResourceSubresources=true|false (BETA - default=true)
CustomResourceValidation=true|false (BETA - default=true)
CustomResourceWebhookConversion=true|false (ALPHA - default=false)
DebugContainers=true|false (ALPHA - default=false)
DevicePlugins=true|false (BETA - default=true)
DryRun=true|false (BETA - default=true)
DynamicAuditing=true|false (ALPHA - default=false)
DynamicKubeletConfig=true|false (BETA - default=true)
EnableEquivalenceClassCache=true|false (ALPHA - default=false)
ExpandInUsePersistentVolumes=true|false (ALPHA - default=false)
ExpandPersistentVolumes=true|false (BETA - default=true)
ExperimentalCriticalPodAnnotation=true|false (ALPHA - default=false)
ExperimentalHostUserNamespaceDefaulting=true|false (BETA - default=false)
HugePages=true|false (BETA - default=true)
HyperVContainer=true|false (ALPHA - default=false)
Initializers=true|false (ALPHA - default=false)
KubeletPodResources=true|false (ALPHA - default=false)
LocalStorageCapacityIsolation=true|false (BETA - default=true)
LocalStorageCapacityIsolationFSQuotaMonitoring=true|false (ALPHA - default=false)
MountContainers=true|false (ALPHA - default=false)
NodeLease=true|false (ALPHA - default=false)
PersistentLocalVolumes=true|false (BETA - default=true)
PodPriority=true|false (BETA - default=true)
PodReadinessGates=true|false (BETA - default=true)
PodShareProcessNamespace=true|false (BETA - default=true)
ProcMountType=true|false (ALPHA - default=false)
QOSReserved=true|false (ALPHA - default=false)
ResourceLimitsPriorityFunction=true|false (ALPHA - default=false)
ResourceQuotaScopeSelectors=true|false (BETA - default=true)
RotateKubeletClientCertificate=true|false (BETA - default=true)
RotateKubeletServerCertificate=true|false (BETA - default=true)
RunAsGroup=true|false (ALPHA - default=false)
RuntimeClass=true|false (ALPHA - default=false)

-h, --help	
	help for kube-scheduler
--http2-max-streams-per-connection int	
	The limit that the server gives to clients for the maximum number of streams in an HTTP/2 connection. Zero means to use golang's default.
--kube-api-burst int32Â Â Â Â Â Default: 100	
	DEPRECATED: burst to use while talking with kubernetes apiserver
--kube-api-content-type stringÂ Â Â Â Â Default: "application/vnd.kubernetes.protobuf"	
	DEPRECATED: content type of requests sent to apiserver.
--kube-api-qps float32Â Â Â Â Â Default: 50	
	DEPRECATED: QPS to use while talking with kubernetes apiserver
--kubeconfig string	
	DEPRECATED: path to kubeconfig file with authorization and master location information.
--leader-electÂ Â Â Â Â Default: true	
	Start a leader election client and gain leadership before executing the main loop. Enable this when running replicated components for high availability.
--leader-elect-lease-duration durationÂ Â Â Â Â Default: 15s	
	The duration that non-leader candidates will wait after observing a leadership renewal until attempting to acquire leadership of a led but unrenewed leader slot. This is effectively the maximum duration that a leader can be stopped before it is replaced by another candidate. This is only applicable if leader election is enabled.
--leader-elect-renew-deadline durationÂ Â Â Â Â Default: 10s	
	The interval between attempts by the acting master to renew a leadership slot before it stops leading. This must be less than or equal to the lease duration. This is only applicable if leader election is enabled.
--leader-elect-resource-lock endpointsÂ Â Â Â Â Default: "endpoints"	
	The type of resource object that is used for locking during leader election. Supported options are endpoints (default) and `configmaps`.
--leader-elect-retry-period durationÂ Â Â Â Â Default: 2s	
	The duration the clients should wait between attempting acquisition and renewal of a leadership. This is only applicable if leader election is enabled.
--lock-object-name stringÂ Â Â Â Â Default: "kube-scheduler"	
	DEPRECATED: define the name of the lock object.
--lock-object-namespace stringÂ Â Â Â Â Default: "kube-system"	
	DEPRECATED: define the namespace of the lock object.
--log-backtrace-at traceLocationÂ Â Â Â Â Default: :0	
	when logging hits line file:N, emit a stack trace
--log-dir string	
	If non-empty, write log files in this directory
--log-file string	
	If non-empty, use this log file
--log-flush-frequency durationÂ Â Â Â Â Default: 5s	
	Maximum number of seconds between log flushes
--logtostderrÂ Â Â Â Â Default: true	

	log to standard error instead of files
--master string	
	The address of the Kubernetes API server (overrides any value in kubeconfig)
--policy-config-file string	
	DEPRECATED: file with scheduler policy configuration. This file is used if policy ConfigMap is not provided or --use-legacy-policy-config=true
--policy-configmap string	
	DEPRECATED: name of the ConfigMap object that contains scheduler's policy configuration. It must exist in the system namespace before scheduler initialization if --use-legacy-policy-config=false. The config must be provided as the value of an element in 'Data' map with the key='policy.cfg'
--policy-configmap-namespace stringÂ Â Â Â Â Default: "kube-system"	
	DEPRECATED: the namespace where policy ConfigMap is located. The kube-system namespace will be used if this is not provided or is empty.
--port intÂ Â Â Â Â Default: 10251	
	DEPRECATED: the port on which to serve HTTP insecurely without authentication and authorization. If 0, don't serve HTTPS at all. See --secure-port instead.
--profiling	
	DEPRECATED: enable profiling via web interface host:port/debug/pprof/
--requestheader-allowed-names stringSlice	
	List of client certificate common names to allow to provide usernames in headers specified by --requestheader-username-headers. If empty, any client certificate validated by the authorities in --requestheader-client-ca-file is allowed.
--requestheader-client-ca-file string	
	Root certificate bundle to use to verify client certificates on incoming requests before trusting usernames in headers specified by --requestheader-username-headers. WARNING: generally do not depend on authorization being already done for incoming requests.
--requestheader-extra-headers-prefix stringSliceÂ Â Â Â Â Default: [x-remote-extra-]	
	List of request header prefixes to inspect. X-Remote-Extra- is suggested.
--requestheader-group-headers stringSliceÂ Â Â Â Â Default: [x-remote-group]	
	List of request headers to inspect for groups. X-Remote-Group is suggested.
--requestheader-username-headers stringSliceÂ Â Â Â Â Default: [x-remote-user]	
	List of request headers to inspect for usernames. X-Remote-User is common.
--scheduler-name stringÂ Â Â Â Â Default: "default-scheduler"	
	DEPRECATED: name of the scheduler, used to select which pods will be processed by this scheduler, based on pod's "spec.schedulerName".
--secure-port intÂ Â Â Â Â Default: 10259	
	The port on which to serve HTTPS with authentication and authorization.If 0, don't serve HTTPS at all.
--skip-headers	
	If true, avoid header prefixes in the log messages
--stderrthreshold severityÂ Â Â Â Â Default: 2	
	logs at or above this threshold go to stderr
--tls-cert-file string	

	File containing the default x509 Certificate for HTTPS. (CA cert, if any, concatenated after server cert). If HTTPS serving is enabled, and --tls-cert-file and --tls-private-key-file are not provided, a self-signed certificate and key are generated for the public address and saved to the directory specified by --cert-dir.
--tls-cipher-suites stringSlice	
	Comma-separated list of cipher suites for the server. If omitted, the default Go cipher suites will be use. Possible values: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_RC4_128_SHA,TLS_RSA_WITH_3DES_EDE_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_RC4_128_SHA
--tls-min-version string	
	Minimum TLS version supported. Possible values: VersionTLS10, VersionTLS11, VersionTLS12
--tls-private-key-file string	
	File containing the default x509 private key matching --tls-cert-file.
--tls-sni-cert-key namedCertKeyÂ Â Â Â Â Default: []	
	A pair of x509 certificate and private key file paths, optionally suffixed with a list of domain patterns which are fully qualified domain names, possibly with prefixed wildcard segments. If no domain patterns are provided, the names of the certificate are extracted. Non-wildcard matches trump over wildcard matches, explicit domain patterns trump over extracted names. For multiple key/certificate pairs, use the --tls-sni-cert-key multiple times. Examples: "example.crt,example.key" or "foo.crt,foo.key:*.foo.com,foo.com".
--use-legacy-policy-config	
	DEPRECATED: when set to true, scheduler will ignore policy ConfigMap and uses policy config file
-v, --v Level	
	number for the log level verbosity
--version version[=true]	
	Print version information and quit
--vmodule moduleSpec	
	comma-separated list of pattern=N settings for file-filtered logging
--write-config-to string	
	If set, write the configuration values to this file and exit.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on May 31, 2019 at 11:28 AM PST by Add support for quotas for ephemeral storage monitoring. (#14268) (Page History)

Edit This Page

kubectl Usage Conventions

Recommended usage conventions for `kubectl`.

- Using `kubectl` in Reusable Scripts
- Best Practices

Using `kubectl` in Reusable Scripts

For a stable output in a script:

- Request one of the machine-oriented output forms, such as `-o name`, `-o json`, `-o yaml`, `-o go-template`, or `-o jsonpath`.
- Fully-qualify the version. For example, `jobs.v1.batch/myjob`. This will ensure that kubectl does not use its default version that can change over time.
- Specify the `--generator` flag to pin to a specific behavior when you use generator-based commands such as `kubectl run` or `kubectl expose`.
- Don't rely on context, preferences, or other implicit states.

Best Practices

`kubectl run`

For `kubectl run` to satisfy infrastructure as code:

- Tag the image with a version-specific tag and don't move that tag to a new version. For example, use `:v1234`, `v1.2.3`, `r03062016-1-4`, rather than `:latest` (For more information, see Best Practices for Configuration).
- Capture the parameters in a checked-in script, or at least use `--record` to annotate the created objects with the command line for an image that is lightly parameterized.
- Check in the script for an image that is heavily parameterized.

- Switch to configuration files checked into source control for features that are needed, but not expressible via `kubectl run` flags.
- Pin to a specific generator version, such as `kubectl run --generator=deployment/v1beta1`.

Generators

You can create the following resources using `kubectl run` with the `--generator` flag:

Resource	api group	kubectl command
Pod	v1	`kubectl run --generator=run-pod/v1`
Replication controller (deprecated)	v1	`kubectl run --generator=run/v1`
Deployment (deprecated)	extensions/v1beta1	`kubectl run --generator=deployment/v1beta1`
Deployment (deprecated)	apps/v1beta1	`kubectl run --generator=deployment/apps.v1beta1`
Job (deprecated)	batch/v1	`kubectl run --generator=job/v1`
CronJob (deprecated)	batch/v1beta1	`kubectl run --generator=cronjob/v1beta1`
CronJob (deprecated)	batch/v2alpha1	`kubectl run --generator=cronjob/v2alpha1`

> **Note:** `kubectl run --generator` except for `run-pod/v1` is deprecated in v1.12.

If you do not specify a generator flag, other flags prompt you to use a specific generator. The following table lists the flags that force you to use specific generators, depending on the version of the cluster:

Generated Resource	Cluster v1.4 and later	Cluster v1.3	Cluster v1.2	Cluster v1.1 and ea
Pod	`--restart=Never`	`--restart=Never`	`--generator=run-pod/v1`	`--restart=OnFai` OR `--restart=Never`
Replication Controller	`--generator=run/v1`	`--generator=run/v1`	`--generator=run/v1`	`--restart=Alwa`
Deployment	`--restart=Always`	`--restart=Always`	`--restart=Always`	N/A
Job	`--restart=OnFailure`	`--restart=OnFailure`	`--restart=OnFailure` OR `--restart=Never`	N/A
Cron Job	`--schedule=<cron>`	N/A	N/A	N/A

> **Note:** These flags use a default generator only when you have not specified any flag. This means that when you combine `--generator` with other flags the generator that you specified later does not change. For example, in a cluster v1.4, if you initially specify `--restart=Always`, a Deployment is created; if you later specify `--restart=Always` and `--generator=run/v1`, a Replication

Controller is created. This enables you to pin to a specific behavior with the generator, even when the default generator is changed later.

The flags set the generator in the following order: first the `--schedule` flag, then the `--restart` policy flag, and finally the `--generator` flag.

To check the final resource that was created, use the `--dry-run` flag, which provides the object to be submitted to the cluster.

`kubectl apply`

- You can use `kubectl apply` to create or update resources. For more information about using kubectl apply to update resources, see Kubectl Book.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on July 07, 2019 at 12:50 PM PST by update generator table for kubectl run (#15146) (Page History)

Edit This Page

Overview of kubectl

Kubectl is a command line interface for running commands against Kubernetes clusters. `kubectl` looks for a file named config in the $HOME/.kube directory. You can specify other kubeconfig files by setting the KUBECONFIG environment variable or by setting the `--kubeconfig` flag.

This overview covers `kubectl` syntax, describes the command operations, and provides common examples. For details about each command, including all the supported flags and subcommands, see the kubectl reference documentation. For installation instructions see installing kubectl.

- Syntax
- Operations
- Resource types
- Output options
- Examples: Common operations
- Examples: Creating and using plugins
- What's next

Syntax

Use the following syntax to run `kubectl` commands from your terminal window:

```
kubectl [command] [TYPE] [NAME] [flags]
```

where `command`, `TYPE`, `NAME`, and `flags` are:

- `command`: Specifies the operation that you want to perform on one or more resources, for example `create`, `get`, `describe`, `delete`.
- `TYPE`: Specifies the resource type. Resource types are case-insensitive and you can specify the singular, plural, or abbreviated forms. For example, the following commands produce the same output:

  ```
  kubectl get pod pod1
  kubectl get pods pod1
  kubectl get po pod1
  ```

- `NAME`: Specifies the name of the resource. Names are case-sensitive. If the name is omitted, details for all resources are displayed, for example `kubectl get pods`.

When performing an operation on multiple resources, you can specify each resource by type and name or specify one or more files:

- To specify resources by type and name:
  - To group resources if they are all the same type: `TYPE1 name1 name2 name<#>`.
    Example: `kubectl get pod example-pod1 example-pod2`
  - To specify multiple resource types individually: `TYPE1/name1 TYPE1/name2 TYPE2/name3 TYPE<#>/name<#>`.
    Example: `kubectl get pod/example-pod1 replicationcontroller/example-rc1`
- To specify resources with one or more files: `-f file1 -f file2 -f file<#>`
  - Use YAML rather than JSON since YAML tends to be more user-friendly, especially for configuration files.
    Example: `kubectl get pod -f ./pod.yaml`
- `flags`: Specifies optional flags. For example, you can use the `-s` or `--server` flags to specify the address and port of the Kubernetes API server.

  Caution: Flags that you specify from the command line override default values and any corresponding environment variables.

If you need help, just run `kubectl help` from the terminal window.

Operations

The following table includes short descriptions and the general syntax for all of the `kubectl` operations:

Operation	Syntax	Description
`annotate`	`kubectl annotate (-f FILENAME | TYPE NAME | TYPE/NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--overwrite] [--all] [--resource-version=version] [flags]`	Add or update the annotations of one or more resources.
`api-versions`	`kubectl api-versions [flags]`	List the API versions that are available.
`apply`	`kubectl apply -f FILENAME [flags]`	Apply a configuration change to a resource from a file or stdin.
`attach`	`kubectl attach POD -c CONTAINER [-i] [-t] [flags]`	Attach to a running container either to view the output stream or interact with the container (stdin).
`autoscale`	`kubectl autoscale (-f FILENAME | TYPE NAME | TYPE/NAME) [--min=MINPODS] --max=MAXPODS [--cpu-percent=CPU] [flags]`	Automatically scale the set of pods that are managed by a replication controller.
`cluster-info`	`kubectl cluster-info [flags]`	Display endpoint information about the master and services in the cluster.
`config`	`kubectl config SUBCOMMAND [flags]`	Modifies kubeconfig files. See the individual subcommands for details.
`create`	`kubectl create -f FILENAME [flags]`	Create one or more resources from a file or stdin.
`delete`	`kubectl delete (-f FILENAME | TYPE [NAME | /NAME | -l label | --all]) [flags]`	Delete resources either from a file, stdin, or specifying label selectors, names, resource selectors, or resources.
`describe`	`kubectl describe (-f FILENAME | TYPE [NAME_PREFIX | /NAME | -l label]) [flags]`	Display the detailed state of one or more resources.

Operation	Syntax	Description
`diff`	`kubectl diff -f FILENAME [flags]`	Diff file or stdin against live configuration (**BETA**)
`edit`	`kubectl edit (-f FILENAME | TYPE NAME | TYPE/NAME) [flags]`	Edit and update the definition of one or more resources on the server by using the default editor.
`exec`	`kubectl exec POD [-c CONTAINER] [-i] [-t] [flags] [-- COMMAND [args...]]`	Execute a command against a container in a pod.
`explain`	`kubectl explain [--recursive=false] [flags]`	Get documentation of various resources. For instance pods, nodes, services, etc.
`expose`	`kubectl expose (-f FILENAME | TYPE NAME | TYPE/NAME) [--port=port] [--protocol=TCP|UDP] [--target-port=number-or-name] [--name=name] [--external-ip=external-ip-of-service] [--type=type] [flags]`	Expose a replication controller, service, or pod as a new Kubernetes service.
`get`	`kubectl get (-f FILENAME | TYPE [NAME | /NAME | -l label]) [--watch] [--sort-by=FIELD] [[-o | --output]=OUTPUT_FORMAT] [flags]`	List one or more resources.
`label`	`kubectl label (-f FILENAME | TYPE NAME | TYPE/NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--overwrite] [--all] [--resource-version=version] [flags]`	Add or update the labels of one or more resources.
`logs`	`kubectl logs POD [-c CONTAINER] [--follow] [flags]`	Print the logs for a container in a pod.
`patch`	`kubectl patch (-f FILENAME | TYPE NAME | TYPE/NAME) --patch PATCH [flags]`	Update one or more fields of a resource by using the strategic merge patch process.
`port-forward`	`kubectl port-forward POD [LOCAL_PORT:]REMOTE_PORT [...[LOCAL_PORT_N:]REMOTE_PORT_N] [flags]`	Forward one or more local ports to a pod.
`proxy`	`kubectl proxy [--port=PORT] [--www=static-dir] [--www-prefix=prefix] [--api-prefix=prefix] [flags]`	Run a proxy to the Kubernetes API server.
`replace`	`kubectl replace -f FILENAME`	Replace a resource from a file or stdin.
`rolling-update`	`kubectl rolling-update OLD_CONTROLLER_NAME ([NEW_CONTROLLER_NAME] --image=NEW_CONTAINER_IMAGE | -f NEW_CONTROLLER_SPEC) [flags]`	Perform a rolling update by gradually replacing the specified replication controller and its pods.

Operation	Syntax	Description
`run`	`kubectl run NAME --image=image [--env="key=value"] [--port=port] [--replicas=replicas] [--dry-run=bool] [--overrides=inline-json] [flags]`	Run a specified image on the cluster.
`scale`	`kubectl scale (-f FILENAME | TYPE NAME | TYPE/NAME) --replicas=COUNT [--resource-version=version] [--current-replicas=count] [flags]`	Update the size of the specified replication controller.
`stop`	`kubectl stop`	Deprecated: Instead, see `kubectl delete`.
`version`	`kubectl version [--client] [flags]`	Display the Kubernetes version running on the client and server.

Remember: For more about command operations, see the kubectl reference documentation.

Resource types

The following table includes a list of all the supported resource types and their abbreviated aliases.

(This output can be retrieved from `kubectl api-resources`, and is accurate as of Kubernetes 1.13.3.)

Resource Name	Short Names	API Group	Namespaced	Resource
`componentstatuses`	`cs`		false	Compone
`configmaps`	`cm`		true	ConfigM
`endpoints`	`ep`		true	Endpoint
`limitranges`	`limits`		true	LimitRan
`namespaces`	`ns`		false	Namespa
`nodes`	`no`		false	Node
`persistentvolumeclaims`	`pvc`		true	Persistent
`persistentvolumes`	`pv`		false	Persistent
`pods`	`po`		true	Pod
`podtemplates`			true	PodTemp
`replicationcontrollers`	`rc`		true	Replicatio
`resourcequotas`	`quota`		true	Resource
`secrets`			true	Secret
`serviceaccounts`	`sa`		true	ServiceA
`services`	`svc`		true	Service
`mutatingwebhookconfigurations`		admissionregistration.k8s.io	false	Mutating
`validatingwebhookconfigurations`		admissionregistration.k8s.io	false	Validating
`customresourcedefinitions`	`crd`, `crds`	apiextensions.k8s.io	false	CustomR
`apiservices`		apiregistration.k8s.io	false	APIServi

Resource Name	Short Names	API Group	Namespaced	Resource
`controllerrevisions`		apps	true	Controlle
`daemonsets`	`ds`	apps	true	DaemonS
`deployments`	`deploy`	apps	true	Deploym
`replicasets`	`rs`	apps	true	ReplicaSe
`statefulsets`	`sts`	apps	true	StatefulS
`tokenreviews`		authentication.k8s.io	false	TokenRev
`localsubjectaccessreviews`		authorization.k8s.io	true	LocalSub
`selfsubjectaccessreviews`		authorization.k8s.io	false	SelfSubje
`selfsubjectrulesreviews`		authorization.k8s.io	false	SelfSubje
`subjectaccessreviews`		authorization.k8s.io	false	SubjectA
`horizontalpodautoscalers`	`hpa`	autoscaling	true	Horizonta
`cronjobs`	`cj`	batch	true	CronJob
`jobs`		batch	true	Job
`certificatesigningrequests`	`csr`	certificates.k8s.io	false	Certificat
`leases`		coordination.k8s.io	true	Lease
`events`	`ev`	events.k8s.io	true	Event
`ingresses`	`ing`	extensions	true	Ingress
`networkpolicies`	`netpol`	networking.k8s.io	true	NetworkP
`poddisruptionbudgets`	`pdb`	policy	true	PodDisru
`podsecuritypolicies`	`psp`	policy	false	PodSecur
`clusterrolebindings`		rbac.authorization.k8s.io	false	ClusterRo
`clusterroles`		rbac.authorization.k8s.io	false	ClusterRo
`rolebindings`		rbac.authorization.k8s.io	true	RoleBind
`roles`		rbac.authorization.k8s.io	true	Role
`priorityclasses`	`pc`	scheduling.k8s.io	false	PriorityC
`storageclasses`	`sc`	storage.k8s.io	false	StorageC
`volumeattachments`		storage.k8s.io	false	VolumeA

Output options

Use the following sections for information about how you can format or sort the output of certain commands. For details about which commands support the various output options, see the [kubectl]() reference documentation.

Formatting output

The default output format for all `kubectl` commands is the human readable plain-text format. To output details to your terminal window in a specific format, you can add either the `-o` or `--output` flags to a supported `kubectl` command.

Syntax

```
kubectl [command] [TYPE] [NAME] -o <output_format>
```

Depending on the `kubectl` operation, the following output formats are supported:

Output format	Description
`-o custom-columns=<spec>`	Print a table using a comma separated list of custom columns.
`-o custom-columns-file=<filename>`	Print a table using the custom columns template in the `<filename>` file.
`-o json`	Output a JSON formatted API object.
`-o jsonpath=<template>`	Print the fields defined in a jsonpath expression.
`-o jsonpath-file=<filename>`	Print the fields defined by the jsonpath expression in the `<filename>` file.
`-o name`	Print only the resource name and nothing else.
`-o wide`	Output in the plain-text format with any additional information. For pods, the node name is included.
`-o yaml`	Output a YAML formatted API object.

Example

In this example, the following command outputs the details for a single pod as a YAML formatted object:

```
kubectl get pod web-pod-13je7 -o yaml
```

Remember: See the kubectl reference documentation for details about which output format is supported by each command.

Custom columns

To define custom columns and output only the details that you want into a table, you can use the `custom-columns` option. You can choose to define the custom columns inline or use a template file: `-o custom-columns=<spec>` or `-o custom-columns-file=<filename>`.

Examples

Inline:

```
kubectl get pods <pod-name> -o custom-columns=NAME:.metadata.name
,RSRC:.metadata.resourceVersion
```

Template file:

```
kubectl get pods <pod-name> -o custom-columns-file=template.txt
```

where the `template.txt` file contains:

```
NAME          RSRC
metadata.name metadata.resourceVersion
```

The result of running either command is:

```
NAME           RSRC
submit-queue   610995
```

Server-side columns

`kubectl` supports receiving specific column information from the server about objects. This means that for any given resource, the server will return columns and rows relevant to that resource, for the client to print. This allows for consistent human-readable output across clients used against the same cluster, by having the server encapsulate the details of printing.

This feature is enabled by default in `kubectl` 1.11 and higher. To disable it, add the `--server-print=false` flag to the `kubectl get` command.

Examples

To print information about the status of a pod, use a command like the following:

```
kubectl get pods <pod-name> --server-print=false
```

Output looks like this:

```
NAME       READY     STATUS              RESTARTS    AGE
pod-name   1/1       Running             0           1m
```

Sorting list objects

To output objects to a sorted list in your terminal window, you can add the `--sort-by` flag to a supported `kubectl` command. Sort your objects by specifying any numeric or string field with the `--sort-by` flag. To specify a field, use a jsonpath expression.

Syntax

```
kubectl [command] [TYPE] [NAME] --sort-by=<jsonpath_exp>
```

Example

To print a list of pods sorted by name, you run:

```
kubectl get pods --sort-by=.metadata.name
```

Examples: Common operations

Use the following set of examples to help you familiarize yourself with running the commonly used `kubectl` operations:

`kubectl apply` - Apply or Update a resource from a file or stdin.

```
# Create a service using the definition in example-service.yaml.
kubectl apply -f example-service.yaml

# Create a replication controller using the definition in
example-controller.yaml.
kubectl apply -f example-controller.yaml
```

```
# Create the objects that are defined in any .yaml, .yml,
or .json file within the <directory> directory.
kubectl apply -f <directory>
```

`kubectl get` - List one or more resources.

```
# List all pods in plain-text output format.
kubectl get pods

# List all pods in plain-text output format and include
additional information (such as node name).
kubectl get pods -o wide

# List the replication controller with the specified name in
plain-text output format. Tip: You can shorten and replace the
'replicationcontroller' resource type with the alias 'rc'.
kubectl get replicationcontroller <rc-name>

# List all replication controllers and services together in
plain-text output format.
kubectl get rc,services

# List all daemon sets, including uninitialized ones, in plain-
text output format.
kubectl get ds --include-uninitialized

# List all pods running on node server01
kubectl get pods --field-selector=spec.nodeName=server01
```

`kubectl describe` - Display detailed state of one or more resources, including the uninitialized ones by default.

```
# Display the details of the node with name <node-name>.
kubectl describe nodes <node-name>

# Display the details of the pod with name <pod-name>.
kubectl describe pods/<pod-name>

# Display the details of all the pods that are managed by the
replication controller named <rc-name>.
# Remember: Any pods that are created by the replication
controller get prefixed with the name of the replication
controller.
kubectl describe pods <rc-name>

# Describe all pods, not including uninitialized ones
kubectl describe pods --include-uninitialized=false
```

> **Note:** The `kubectl get` command is usually used for retrieving one or more resources of the same resource type. It features a rich set of flags that allows you to customize the output format using the `-o` or `--output` flag, for example. You can specify the `-w` or `--watch` flag to start watching updates to a particular object. The `kubectl describe` command is more focused on describing the many related

aspects of a specified resource. It may invoke several API calls to the API server to build a view for the user. For example, the `kubectl describe node` command retrieves not only the information about the node, but also a summary of the pods running on it, the events generated for the node etc.

`kubectl delete` - Delete resources either from a file, stdin, or specifying label selectors, names, resource selectors, or resources.

```
# Delete a pod using the type and name specified in the pod.yaml
file.
kubectl delete -f pod.yaml

# Delete all the pods and services that have the label
name=<label-name>.
kubectl delete pods,services -l name=<label-name>

# Delete all the pods and services that have the label
name=<label-name>, including uninitialized ones.
kubectl delete pods,services -l name=<label-name> --include-
uninitialized

# Delete all pods, including uninitialized ones.
kubectl delete pods --all
```

`kubectl exec` - Execute a command against a container in a pod.

```
# Get output from running 'date' from pod <pod-name>. By
default, output is from the first container.
kubectl exec <pod-name> date

# Get output from running 'date' in container <container-name>
of pod <pod-name>.
kubectl exec <pod-name> -c <container-name> date

# Get an interactive TTY and run /bin/bash from pod <pod-name>.
By default, output is from the first container.
kubectl exec -ti <pod-name> /bin/bash
```

`kubectl logs` - Print the logs for a container in a pod.

```
# Return a snapshot of the logs from pod <pod-name>.
kubectl logs <pod-name>

# Start streaming the logs from pod <pod-name>. This is similar
to the 'tail -f' Linux command.
kubectl logs -f <pod-name>
```

Examples: Creating and using plugins

Use the following set of examples to help you familiarize yourself with writing and using `kubectl` plugins:

```
# create a simple plugin in any language and name the resulting
executable file
# so that it begins with the prefix "kubectl-"
cat ./kubectl-hello
#!/bin/bash

# this plugin prints the words "hello world"
echo "hello world"

# with our plugin written, let's make it executable
sudo chmod +x ./kubectl-hello

# and move it to a location in our PATH
sudo mv ./kubectl-hello /usr/local/bin

# we have now created and "installed" a kubectl plugin.
# we can begin using our plugin by invoking it from kubectl as
if it were a regular command
kubectl hello
```

```
hello world
```

```
# we can "uninstall" a plugin, by simply removing it from our
PATH
sudo rm /usr/local/bin/kubectl-hello
```

In order to view all of the plugins that are available to `kubectl`, we can use the `kubectl plugin list` subcommand:

```
kubectl plugin list
```

```
The following kubectl-compatible plugins are available:

/usr/local/bin/kubectl-hello
/usr/local/bin/kubectl-foo
/usr/local/bin/kubectl-bar
```

```
# this command can also warn us about plugins that are
# not executable, or that are overshadowed by other
# plugins, for example
sudo chmod -x /usr/local/bin/kubectl-foo
kubectl plugin list
```

```
The following kubectl-compatible plugins are available:

/usr/local/bin/kubectl-hello
/usr/local/bin/kubectl-foo
  - warning: /usr/local/bin/kubectl-foo identified as a plugin,
but it is not executable
/usr/local/bin/kubectl-bar

error: one plugin warning was found
```

We can think of plugins as a means to build more complex functionality on top of the existing kubectl commands:

```
cat ./kubectl-whoami
#!/bin/bash

# this plugin makes use of the `kubectl config` command in order
to output
# information about the current user, based on the currently
selected context
kubectl config view --template='{{ range .contexts }}{{ if
eq .name "'$(kubectl config current-context)'" }}Current user:
{{ .context.user }}{{ end }}{{ end }}'
```

Running the above plugin gives us an output containing the user for the currently selected context in our KUBECONFIG file:

```
# make the file executable
sudo chmod +x ./kubectl-whoami

# and move it into our PATH
sudo mv ./kubectl-whoami /usr/local/bin

kubectl whoami
Current user: plugins-user
```

To find out more about plugins, take a look at the example cli plugin.

What's next

Start using the kubectl commands.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

JSONPath Support

Kubectl supports JSONPath template.

JSONPath template is composed of JSONPath expressions enclosed by curly braces {}. Kubectl uses JSONPath expressions to filter on specific fields in the JSON object and format the output. In addition to the original JSONPath template syntax, the following functions and syntax are valid:

1. Use double quotes to quote text inside JSONPath expressions.
2. Use the `range`, `end` operators to iterate lists.
3. Use negative slice indices to step backwards through a list. Negative indices do not "wrap around" a list and are valid as long as `-index + listLength >= 0`.

   Note:

   - The `$` operator is optional since the expression always starts from the root object by default.

   - The result object is printed as its String() function.

Given the JSON input:

```
{
  "kind": "List",
  "items":[
    {
      "kind":"None",
      "metadata":{"name":"127.0.0.1"},
      "status":{
        "capacity":{"cpu":"4"},
        "addresses":[{"type": "LegacyHostIP", "address":"127.0.0.1"}]
      }
    },
    {
      "kind":"None",
      "metadata":{"name":"127.0.0.2"},
      "status":{
        "capacity":{"cpu":"8"},
        "addresses":[
          {"type": "LegacyHostIP", "address":"127.0.0.2"},
          {"type": "another", "address":"127.0.0.3"}
        ]
      }
    }
  ],
  "users":[
    {
      "name": "myself",
      "user": {}
    },
```

```
    {
      "name": "e2e",
      "user": {"username": "admin", "password": "secret"}
    }
  ]
}
```

Function	Description	Example	Result
text	the plain text	kind is {.kind}	kind is List
@	the current object	{@}	the same as input
. or []	child operator	{.kind} or {[â€˜kind']}	List
..	recursive descent	{..name}	127.0.0.1 127.0.0.2 myself e2e
*	wildcard. Get all objects	{.items[*].metadata.name}	[127.0.0.1 127.0.0.2]
[start:end :step]	subscript operator	{.users[0].name}	myself
[,]	union operator	{.items[*][â€˜metadata.name', â€˜status.capacity']}	127.0.0.1 127.0.0.2 map[cpu:4] map[cpu:8]
?()	filter	{.users[?(@.name=="e2e")].user.password}	secret
range, end	iterate list	{range .items[*]}[{.metadata.name}, {.status.capacity}] {end}	[127.0.0.1, map[cpu:4]] [127.0.0.2, map[cpu:8]]
''	quote interpreted string	{range .items[*]}{.metadata.name}{'\t'}{end}	127.0.0.1 127.0.0.2

Examples using `kubectl` and JSONPath expressions:

```
kubectl get pods -o json
kubectl get pods -o=jsonpath='{@}'
kubectl get pods -o=jsonpath='{.items[0]}'
kubectl get pods -o=jsonpath='{.items[0].metadata.name}'
kubectl get pods -o=jsonpath='{range .items[*]}
{.metadata.name}{"\t"}{.status.startTime}{"\n"}{end}'
```

On Windows, you must *double* quote any JSONPath template that contains spaces (not single quote as shown above for bash). This in turn means that you must use a single quote or escaped double quote around any literals in the template. For example:

```
C:\> kubectl get pods -o=jsonpath="{range .items[*]}
{.metadata.name}{'\t'}{.status.startTime}{'\n'}{end}"
C:\> kubectl get pods -o=jsonpath="{range .items[*]}
{.metadata.name}{\"\t\"}{.status.startTime}{\"\n\"}{end}"
```

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on March 07, 2019 at 3:01 PM PST by Code snippents shouldn't include the command prompt (#12779) (Page History)

Edit This Page

- - kubectl
    - Synopsis
    - Options
    - SEE ALSO

kubectl

kubectl controls the Kubernetes cluster manager

Synopsis

kubectl controls the Kubernetes cluster manager.

Find more information at: https://kubernetes.io/docs/reference/kubectl/overview/

```
kubectl [flags]
```

Options

--alsologtostderr	
	log to standard error as well as files
--application-metrics-count-limit intÂ Â Â Â Â Default: 100	
	Max number of application metrics to store (per container)
--as string	
	Username to impersonate for the operation
--as-group stringArray	
	Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
--azure-container-registry-config string	
	Path to the file containing Azure container registry configuration information.
--boot-id-file stringÂ Â Â Â Â Default: "/proc/sys/kernel/random/boot_id"	
	Comma-separated list of files to check for boot-id. Use the first one that exists.
--cache-dir stringÂ Â Â Â Â Default: "/home/tengqm/.kube/http-cache"	
	Default HTTP cache directory
--certificate-authority string	
	Path to a cert file for the certificate authority

--client-certificate string
Path to a client certificate file for TLS
--client-key string
Path to a client key file for TLS
--cloud-provider-gce-lb-src-cidrs cidrsÂ Â Â Â Â Default: 130.211.0.0/22,209.85.152.0/22,209.85.204.0/22,35.191.0.0/16
CIDRs opened in GCE firewall for LB traffic proxy & health checks
--cluster string
The name of the kubeconfig cluster to use
--container-hints stringÂ Â Â Â Â Default: "/etc/cadvisor/container_hints.json"
location of the container hints file
--containerd stringÂ Â Â Â Â Default: "unix:///var/run/containerd.sock"
containerd endpoint
--context string
The name of the kubeconfig context to use
--default-not-ready-toleration-seconds intÂ Â Â Â Â Default: 300
Indicates the tolerationSeconds of the toleration for notReady:NoExecute that is added by default to every pod that does not already have such a toleration.
--default-unreachable-toleration-seconds intÂ Â Â Â Â Default: 300
Indicates the tolerationSeconds of the toleration for unreachable:NoExecute that is added by default to every pod that does not already have such a toleration.
--docker stringÂ Â Â Â Â Default: "unix:///var/run/docker.sock"
docker endpoint
--docker-env-metadata-whitelist string
a comma-separated list of environment variable keys that needs to be collected for docker containers
--docker-only
Only report docker containers in addition to root stats
--docker-root stringÂ Â Â Â Â Default: "/var/lib/docker"
DEPRECATED: docker root is read from docker info (this is a fallback, default: /var/lib/docker)
--docker-tls
use TLS to connect to docker
--docker-tls-ca stringÂ Â Â Â Â Default: "ca.pem"
path to trusted CA
--docker-tls-cert stringÂ Â Â Â Â Default: "cert.pem"
path to client certificate
--docker-tls-key stringÂ Â Â Â Â Default: "key.pem"
path to private key
--enable-load-reader
Whether to enable cpu load reader
--event-storage-age-limit stringÂ Â Â Â Â Default: "default=0"

	Max length of time for which to store events (per type). Value is a comma separated list of key values, where the keys are event types (e.g.: creation, oom) or "default" and the value is a duration. Default is applied to all non-specified event types
--event-storage-event-limit stringÂ Â Â Â Â Default: "default=0"	
	Max number of events to store (per type). Value is a comma separated list of key values, where the keys are event types (e.g.: creation, oom) or "default" and the value is an integer. Default is applied to all non-specified event types
--global-housekeeping-interval durationÂ Â Â Â Â Default: 1m0s	
	Interval between global housekeepings
-h, --help	
	help for kubectl
--housekeeping-interval durationÂ Â Â Â Â Default: 10s	
	Interval between container housekeepings
--insecure-skip-tls-verify	
	If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
--kubeconfig string	
	Path to the kubeconfig file to use for CLI requests.
--log-backtrace-at traceLocationÂ Â Â Â Â Default: :0	
	when logging hits line file:N, emit a stack trace
--log-cadvisor-usage	
	Whether to log the usage of the cAdvisor container
--log-dir string	
	If non-empty, write log files in this directory
--log-file string	
	If non-empty, use this log file
--log-flush-frequency durationÂ Â Â Â Â Default: 5s	
	Maximum number of seconds between log flushes
--logtostderrÂ Â Â Â Â Default: true	
	log to standard error instead of files
--machine-id-file stringÂ Â Â Â Â Default: "/etc/machine-id,/var/lib/dbus/machine-id"	
	Comma-separated list of files to check for machine-id. Use the first one that exists.
--match-server-version	
	Require server version to match client version
--mesos-agent stringÂ Â Â Â Â Default: "127.0.0.1:5051"	
	Mesos agent address
--mesos-agent-timeout durationÂ Â Â Â Â Default: 10s	
	Mesos agent timeout
-n, --namespace string	
	If present, the namespace scope for this CLI request
--password string	
	Password for basic authentication to the API server
--profile stringÂ Â Â Â Â Default: "none"	

	Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
--profile-output stringÂ Â Â Â Â Default: "profile.pprof"	
	Name of the file to write the profile to
--request-timeout stringÂ Â Â Â Â Default: "0"	
	The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
-s, --server string	
	The address and port of the Kubernetes API server
--skip-headers	
	If true, avoid header prefixes in the log messages
--stderrthreshold severityÂ Â Â Â Â Default: 2	
	logs at or above this threshold go to stderr
--storage-driver-buffer-duration durationÂ Â Â Â Â Default: 1m0s	
	Writes in the storage driver will be buffered for this duration, and committed to the non memory backends as a single transaction
--storage-driver-db stringÂ Â Â Â Â Default: "cadvisor"	
	database name
--storage-driver-host stringÂ Â Â Â Â Default: "localhost:8086"	
	database host:port
--storage-driver-password stringÂ Â Â Â Â Default: "root"	
	database password
--storage-driver-secure	
	use secure connection with database
--storage-driver-table stringÂ Â Â Â Â Default: "stats"	
	table name
--storage-driver-user stringÂ Â Â Â Â Default: "root"	
	database username
--token string	
	Bearer token for authentication to the API server
--update-machine-info-interval durationÂ Â Â Â Â Default: 5m0s	
	Interval between machine info updates.
--user string	
	The name of the kubeconfig user to use
--username string	
	Username for basic authentication to the API server
-v, --v Level	
	number for the log level verbosity
--version version[=true]	
	Print version information and quit
--vmodule moduleSpec	
	comma-separated list of pattern=N settings for file-filtered logging

SEE ALSO

- kubectl annotate - Update the annotations on a resource
- kubectl api-resources - Print the supported API resources on the server
- kubectl api-versions - Print the supported API versions on the server, in the form of "group/version"
- kubectl apply - Apply a configuration to a resource by filename or stdin
- kubectl attach - Attach to a running container
- kubectl auth - Inspect authorization
- kubectl autoscale - Auto-scale a Deployment, ReplicaSet, or ReplicationController
- kubectl certificate - Modify certificate resources.
- kubectl cluster-info - Display cluster info
- kubectl completion - Output shell completion code for the specified shell (bash or zsh)
- kubectl config - Modify kubeconfig files
- kubectl convert - Convert config files between different API versions
- kubectl cordon - Mark node as unschedulable
- kubectl cp - Copy files and directories to and from containers.
- kubectl create - Create a resource from a file or from stdin.
- kubectl delete - Delete resources by filenames, stdin, resources and names, or by resources and label selector
- kubectl describe - Show details of a specific resource or group of resources
- kubectl diff - Diff live version against would-be applied version
- kubectl drain - Drain node in preparation for maintenance
- kubectl edit - Edit a resource on the server
- kubectl exec - Execute a command in a container
- kubectl explain - Documentation of resources
- kubectl expose - Take a replication controller, service, deployment or pod and expose it as a new Kubernetes Service
- kubectl get - Display one or many resources
- kubectl kustomize - Build a kustomization target from a directory or a remote url.
- kubectl label - Update the labels on a resource
- kubectl logs - Print the logs for a container in a pod
- kubectl options - Print the list of flags inherited by all commands
- kubectl patch - Update field(s) of a resource using strategic merge patch
- kubectl plugin - Provides utilities for interacting with plugins.
- kubectl port-forward - Forward one or more local ports to a pod
- kubectl proxy - Run a proxy to the Kubernetes API server
- kubectl replace - Replace a resource by filename or stdin
- kubectl rollout - Manage the rollout of a resource
- kubectl run - Run a particular image on the cluster
- kubectl scale - Set a new size for a Deployment, ReplicaSet, Replication Controller, or Job
- kubectl set - Set specific features on objects
- kubectl taint - Update the taints on one or more nodes
- kubectl top - Display Resource (CPU/Memory/Storage) usage.
- kubectl uncordon - Mark node as schedulable
- kubectl version - Print the client and server version information
- kubectl wait - Experimental: Wait for a specific condition on one or many resources.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on April 02, 2019 at 8:25 PM PST by Updated links on kubectl documentation page (#13579) (Page History)

Edit This Page

kubectl Cheat Sheet

See also: Kubectl Overview and JsonPath Guide.

This page is an overview of the `kubectl` command.

- Kubectl Autocomplete
- Kubectl Context and Configuration
- Apply
- Creating Objects
- Viewing, Finding Resources
- Updating Resources
- Patching Resources
- Editing Resources
- Scaling Resources
- Deleting Resources
- Interacting with running Pods
- Interacting with Nodes and Cluster
- What's next

kubectl - Cheat Sheet

Kubectl Autocomplete

BASH

```
source <(kubectl completion bash) # setup autocomplete in bash into the current shell, bash-completion package should be installed first.
echo "source <(kubectl completion bash)" >> ~/.bashrc # add autocomplete permanently to your bash shell.
```

You can also use a shorthand alias for `kubectl` that also works with completion:

```
alias k=kubectl
complete -F __start_kubectl k
```

ZSH

```
source <(kubectl completion zsh)  # setup autocomplete in zsh
into the current shell
echo "if [ $commands[kubectl] ]; then source <(kubectl
completion zsh); fi" >> ~/.zshrc # add autocomplete permanently
to your zsh shell
```

Kubectl Context and Configuration

Set which Kubernetes cluster `kubectl` communicates with and modifies configuration information. See [Authenticating Across Clusters with kubeconfig]() documentation for detailed config file information.

```
kubectl config view # Show Merged kubeconfig settings.

# use multiple kubeconfig files at the same time and view merged
config
KUBECONFIG=~/.kube/config:~/.kube/kubconfig2

kubectl config view

# get the password for the e2e user
kubectl config view -o jsonpath='{.users[?(@.name ==
"e2e")].user.password}'

kubectl config view -o jsonpath='{.users[].name}'    # get a
list of users
kubectl config get-contexts                          # display
list of contexts
kubectl config current-context                         # display
the current-context
kubectl config use-context my-cluster-name           # set the
default context to my-cluster-name

# add a new cluster to your kubeconf that supports basic auth
kubectl config set-credentials kubeuser/foo.kubernetes.com --
username=kubeuser --password=kubepassword

# permanently save the namespace for all subsequent kubectl
commands in that context.
kubectl config set-context --current --namespace=ggckad-s2

# set a context utilizing a specific username and namespace.
kubectl config set-context gce --user=cluster-admin --namespace=f
oo \
  && kubectl config use-context gce

kubectl config unset users.foo                       # delete
user foo
```

Apply

`apply` manages applications through files defining Kubernetes resources. It creates and updates resources in a cluster through running `kubectl apply`. This is the recommended way of managing Kubernetes applications on production. See Kubectl Book.

Creating Objects

Kubernetes manifests can be defined in json or yaml. The file extension `.yaml`, `.yml`, and `.json` can be used.

```
kubectl apply -f ./my-manifest.yaml           # create
resource(s)
kubectl apply -f ./my1.yaml -f ./my2.yaml     # create from
multiple files
kubectl apply -f ./dir                        # create
resource(s) in all manifest files in dir
kubectl apply -f https://git.io/vPieo         # create
resource(s) from url
kubectl create deployment nginx --image=nginx  # start a single
instance of nginx
kubectl explain pods,svc                       # get the
documentation for pod and svc manifests

# Create multiple YAML objects from stdin
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "1000000"
---
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep-less
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "1000"
EOF
```

```
# Create a secret with several keys
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: $(echo -n "s33msi4" | base64 -w0)
  username: $(echo -n "jane" | base64 -w0)
EOF
```

Viewing, Finding Resources

```
# Get commands with basic output
kubectl get services                          # List all
services in the namespace
kubectl get pods --all-namespaces             # List all pods in
all namespaces
kubectl get pods -o wide                      # List all pods in
the namespace, with more details
kubectl get deployment my-dep                 # List a
particular deployment
kubectl get pods --include-uninitialized      # List all pods in
the namespace, including uninitialized ones
kubectl get pod my-pod -o yaml                # Get a pod's YAML
kubectl get pod my-pod -o yaml --export       # Get a pod's YAML
without cluster specific information

# Describe commands with verbose output
kubectl describe nodes my-node
kubectl describe pods my-pod

kubectl get services --sort-by=.metadata.name # List Services
Sorted by Name

# List pods Sorted by Restart Count
kubectl get pods --sort-by='.status.containerStatuses[0].restartC
ount'

# Get the version label of all pods with label app=cassandra
kubectl get pods --selector=app=cassandra -o \
  jsonpath='{.items[*].metadata.labels.version}'

# Get all worker nodes (use a selector to exclude results that
have a label
# named 'node-role.kubernetes.io/master')
kubectl get node --selector='!node-role.kubernetes.io/master'

# Get all running pods in the namespace
kubectl get pods --field-selector=status.phase=Running
```

```
# Get ExternalIPs of all nodes
kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP")].address}'

# List Names of Pods that belong to Particular RC
# "jq" command useful for transformations that are too complex for jsonpath, it can be found at https://stedolan.github.io/jq/
sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector | to_entries | .[] | "\(.key)=\(.value),"')%?}
echo $(kubectl get pods --selector=$sel --output=jsonpath={.items..metadata.name})

# Show labels for all pods (or any other Kubernetes object that supports labelling)
# Also uses "jq"
for item in $( kubectl get pod --output=name); do printf "Labels for %s\n" "$item" | grep --color -E '[^/]+$' && kubectl get "$item" --output=json | jq -r -S '.metadata.labels | to_entries | .[] | " \(.key)=\(.value)"' 2>/dev/null; printf "\n"; done

# Or this command can be used as well to get all the labels associated with pods
kubectl get pods --show-labels

# Check which nodes are ready
JSONPATH='{range .items[*]}{@.metadata.name}:{range @.status.conditions[*]}{@.type}={@.status};{end}{end}' \
 && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"

# List all Secrets currently in use by a pod
kubectl get pods -o json | jq '.items[].spec.containers[].env[]?.valueFrom.secretKeyRef.name' | grep -v null | sort | uniq

# List Events sorted by timestamp
kubectl get events --sort-by=.metadata.creationTimestamp
```

Updating Resources

As of version 1.11 `rolling-update` have been deprecated (see CHANGELOG-1.11.md), use `rollout` instead.

```
kubectl set image deployment/frontend www=image:v2
# Rolling update "www" containers of "frontend" deployment, updating the image
kubectl rollout undo deployment/frontend
# Rollback to the previous deployment
kubectl rollout status -w deployment/frontend
# Watch rolling update status of "frontend" deployment until completion

# deprecated starting version 1.11
```

```
kubectl rolling-update frontend-v1 -f frontend-v2.json
# (deprecated) Rolling update pods of frontend-v1
kubectl rolling-update frontend-v1 frontend-v2 --image=image:v2
# (deprecated) Change the name of the resource and update the
image
kubectl rolling-update frontend --image=image:v2
# (deprecated) Update the pods image of frontend
kubectl rolling-update frontend-v1 frontend-v2 --rollback
# (deprecated) Abort existing rollout in progress

cat pod.json | kubectl replace -f -
# Replace a pod based on the JSON passed into std

# Force replace, delete and then re-create the resource. Will
cause a service outage.
kubectl replace --force -f ./pod.json

# Create a service for a replicated nginx, which serves on port
80 and connects to the containers on port 8000
kubectl expose rc nginx --port=80 --target-port=8000

# Update a single-container pod's image version (tag) to v4
kubectl get pod mypod -o yaml | sed 's/\(image: myimage\):.*$/
\1:v4/' | kubectl replace -f -

kubectl label pods my-pod new-label=awesome
# Add a Label
kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq
# Add an annotation
kubectl autoscale deployment foo --min=2 --max=10
# Auto scale a deployment "foo"
```

Patching Resources

```
kubectl patch node k8s-node-1 -p '{"spec":
{"unschedulable":true}}' # Partially update a node

# Update a container's image; spec.containers[*].name is
required because it's a merge key
kubectl patch pod valid-pod -p '{"spec":{"containers":
[{"name":"kubernetes-serve-hostname","image":"new image"}]}}'

# Update a container's image using a json patch with positional
arrays
kubectl patch pod valid-pod --type='json' -p='[{"op": "replace",
"path": "/spec/containers/0/image", "value":"new image"}]'

# Disable a deployment livenessProbe using a json patch with
positional arrays
kubectl patch deployment valid-deployment  --type json   -p='[{"o
p": "remove", "path": "/spec/template/spec/containers/0/
livenessProbe"}]'
```

```
# Add a new element to a positional array
kubectl patch sa default --type='json' -p='[{"op": "add",
"path": "/secrets/1", "value": {"name": "whatever" } }]'
```

Editing Resources

The edit any API resource in an editor.

```
kubectl edit svc/docker-registry                      # Edit the
service named docker-registry
KUBE_EDITOR="nano" kubectl edit svc/docker-registry   # Use an
alternative editor
```

Scaling Resources

```
kubectl scale --replicas=3 rs/
foo                                  # Scale a replicaset named
'foo' to 3
kubectl scale --replicas=3 -f
foo.yaml                             # Scale a resource specified
in "foo.yaml" to 3
kubectl scale --current-replicas=2 --replicas=3 deployment/
mysql  # If the deployment named mysql's current size is 2,
scale mysql to 3
kubectl scale --replicas=5 rc/foo rc/bar rc/
baz                   # Scale multiple replication controllers
```

Deleting Resources

```
kubectl delete -f ./
pod.json                                              # Delete a
pod using the type and name specified in pod.json
kubectl delete pod,service baz
foo                                        # Delete pods and
services with same names "baz" and "foo"
kubectl delete pods,services -l name=myLabel
         # Delete pods and services with label name=myLabel
kubectl delete pods,services -l name=myLabel --include-
uninitialized      # Delete pods and services, including
uninitialized ones, with label name=myLabel
kubectl -n my-ns delete po,svc --
all                                      # Delete all pods and
services, including uninitialized ones, in namespace my-ns,
# Delete all pods matching the awk pattern1 or pattern2
kubectl get pods  -n mynamespace --no-headers=true | awk '/
pattern1|pattern2/{print $1}' | xargs  kubectl delete -n
mynamespace pod
```

Interacting with running Pods

```
kubectl logs my-pod                                 # dump pod
logs (stdout)
kubectl logs -l name=myLabel                        # dump pod
logs, with label name=myLabel (stdout)
kubectl logs my-pod --previous                      # dump pod
logs (stdout) for a previous instantiation of a container
kubectl logs my-pod -c my-container                 # dump pod
container logs (stdout, multi-container case)
kubectl logs -l name=myLabel -c my-container        # dump pod
logs, with label name=myLabel (stdout)
kubectl logs my-pod -c my-container --previous      # dump pod
container logs (stdout, multi-container case) for a previous
instantiation of a container
kubectl logs -f my-pod                              # stream pod
logs (stdout)
kubectl logs -f my-pod -c my-container              # stream pod
container logs (stdout, multi-container case)
kubectl logs -f -l name=myLabel --all-containers    # stream all
pods logs with label name=myLabel (stdout)
kubectl run -i --tty busybox --image=busybox -- sh  # Run pod as
interactive shell
kubectl attach my-pod -i                            # Attach to
Running Container
kubectl port-forward my-pod 5000:6000               # Listen on
port 5000 on the local machine and forward to port 6000 on my-pod
kubectl exec my-pod -- ls /                         # Run
command in existing pod (1 container case)
kubectl exec my-pod -c my-container -- ls /         # Run
command in existing pod (multi-container case)
kubectl top pod POD_NAME --containers               # Show
metrics for a given pod and its containers
```

Interacting with Nodes and Cluster

```
kubectl cordon my-
node                                                # Mark my-
node as unschedulable
kubectl drain my-
node                                                 # Drain my-
node in preparation for maintenance
kubectl uncordon my-
node                                              # Mark my-node
as schedulable
kubectl top node my-
node                                              # Show metrics
for a given node
kubectl cluster-
```

```
info                                                    # Display
addresses of the master and services
kubectl cluster-info
dump                                               # Dump current
cluster state to stdout
kubectl cluster-info dump --output-directory=/path/to/cluster-
state   # Dump current cluster state to /path/to/cluster-state

# If a taint with that key and effect already exists, its value
is replaced as specified.
kubectl taint nodes foo dedicated=special-user:NoSchedule
```

Resource types

List all supported resource types along with their shortnames, API group, whether they are namespaced, and Kind:

```
kubectl api-resources
```

Other operations for exploring API resources:

```
kubectl api-resources --namespaced=true      # All namespaced
resources
kubectl api-resources --namespaced=false     # All non-
namespaced resources
kubectl api-resources -o name                # All resources
with simple output (just the resource name)
kubectl api-resources -o wide                # All resources
with expanded (aka "wide") output
kubectl api-resources --verbs=list,get       # All resources
that support the "list" and "get" request verbs
kubectl api-resources --api-group=extensions # All resources in
the "extensions" API group
```

Formatting output

To output details to your terminal window in a specific format, you can add either the `-o` or `--output` flags to a supported `kubectl` command.

Output format	Description
`-o=custom-columns=<spec>`	Print a table using a comma separated list of custom columns
`-o=custom-columns-file=<filename>`	Print a table using the custom columns template in the `<filename>` file
`-o=json`	Output a JSON formatted API object
`-o=jsonpath=<template>`	Print the fields defined in a jsonpath expression
`-o=jsonpath-file=<filename>`	Print the fields defined by the jsonpath expression in the `<filename>` file
`-o=name`	Print only the resource name and nothing else
`-o=wide`	Output in the plain-text format with any additional information, and for pods, the node name is included

Output format	Description
`-o=yaml`	Output a YAML formatted API object

Kubectl output verbosity and debugging

Kubectl verbosity is controlled with the `-v` or `--v` flags followed by an integer representing the log level. General Kubernetes logging conventions and the associated log levels are described here.

Verbosity	Description
`--v=0`	Generally useful for this to *always* be visible to a cluster operator.
`--v=1`	A reasonable default log level if you don't want verbosity.
`--v=2`	Useful steady state information about the service and important log messages that may correlate to significant changes in the system. This is the recommended default log level for most systems.
`--v=3`	Extended information about changes.
`--v=4`	Debug level verbosity.
`--v=6`	Display requested resources.
`--v=7`	Display HTTP request headers.
`--v=8`	Display HTTP request contents.
`--v=9`	Display HTTP request contents without truncation of contents.

What's next

- Learn more about Overview of kubectl.
- See kubectl options.
- Also kubectl Usage Conventions to understand how to use it in reusable scripts.
- See more community kubectl cheatsheets.

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on July 12, 2019 at 1:57 AM PST by Add concept page for Operators (#14458) (Page History)

Edit This Page

kubectl Commands

kubectl Command Reference

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on May 11, 2018 at 10:14 AM PST by Put kubectl commands in left nav. (#8502) (Page History)

Edit This Page

kubectl for Docker Users

You can use the Kubernetes command line tool kubectl to interact with the API Server. Using kubectl is straightforward if you are familiar with the Docker command line tool. However, there are a few differences between the docker commands and the kubectl commands. The following sections show a docker sub-command and describe the equivalent kubectl command.

- docker run
- docker ps
- docker attach
- docker exec
- docker logs
- docker stop and docker rm
- docker login
- docker version
- docker info

docker run

To run an nginx Deployment and expose the Deployment, see kubectl run.

docker:

```
docker run -d --restart=always -e DOMAIN=cluster --name nginx-app -p 80:80 nginx
```

```
55c103fa129692154a7652490236fee9be47d70a8dd562281ae7d2f9a339a6db
```

```
docker ps
```

```
CONTAINER ID        IMAGE                   COMMAND
CREATED             STATUS                  PORTS
NAMES
55c103fa1296        nginx                   "nginx -g 'daemon
ofâ€¦"   9 seconds ago       Up 9 seconds        0.0.0.0:80->80/
tcp   nginx-app
```

kubectl:

```
# start the pod running nginx
kubectl run --image=nginx nginx-app --port=80 --env="DOMAIN=clust
er"
```

```
deployment "nginx-app" created
```

> **Note:** `kubectl` commands print the type and name of the resource created or mutated, which can then be used in subsequent commands. You can expose a new Service after a Deployment is created.

```
# expose a port through with a service
kubectl expose deployment nginx-app --port=80 --name=nginx-http
```

```
service "nginx-http" exposed
```

By using kubectl, you can create a [Deployment] to ensure that N pods are running nginx, where N is the number of replicas stated in the spec and defaults to 1. You can also create a [service] with a selector that matches the pod labels. For more information, see [Use a Service to Access an Application in a Cluster].

By default images run in the background, similar to `docker run -d ...`. To run things in the foreground, use:

```
kubectl run [-i] [--tty] --attach <name> --image=<image>
```

Unlike `docker run ...`, if you specify `--attach`, then you attach `stdin`, `stdout` and `stderr`. You cannot control which streams are attached (`docker -a ...`). To detach from the container, you can type the escape sequence Ctrl+P followed by Ctrl+Q.

Because the kubectl run command starts a Deployment for the container, the Deployment restarts if you terminate the attached process by using Ctrl+C, unlike `docker run -it`. To destroy the Deployment and its pods you need to run `kubectl delete deployment <name>`.

docker ps

To list what is currently running, see kubectl get.

docker:

```
docker ps -a
```

```
CONTAINER ID        IMAGE                   COMMAND
CREATED              STATUS
```

```
PORTS                   NAMES
14636241935f        ubuntu:16.04        "echo test"
5 seconds ago        Exited (0) 5 seconds
ago                        cocky_fermi
55c103fa1296        nginx               "nginx -g 'daemon
ofâ€¦"   About a minute ago   Up About a minute
0.0.0.0:80->80/tcp   nginx-app
```

kubectl:

```
kubectl get po
```

```
NAME                        READY     STATUS      RESTARTS   AGE
nginx-app-8df569cb7-4gd89   1/1       Running     0          3m
ubuntu                      0/1       Completed   0          20s
```

docker attach

To attach a process that is already running in a container, see kubectl attach.

docker:

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS
NAMES
55c103fa1296        nginx               "nginx -g 'daemon
ofâ€¦"   5 minutes ago       Up 5 minutes        0.0.0.0:80->80/
tcp   nginx-app
```

```
docker attach 55c103fa1296
...
```

kubectl:

```
kubectl get pods
```

```
NAME              READY     STATUS    RESTARTS   AGE
nginx-app-5jyvm   1/1       Running   0          10m
```

```
kubectl attach -it nginx-app-5jyvm
...
```

To detach from the container, you can type the escape sequence Ctrl+P followed by Ctrl+Q.

docker exec

To execute a command in a container, see kubectl exec.

docker:

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS
NAMES
55c103fa1296        nginx               "nginx -g 'daemon
ofâ€¦"   6 minutes ago       Up 6 minutes        0.0.0.0:80->80/
tcp   nginx-app
```

```
docker exec 55c103fa1296 cat /etc/hostname
```

```
55c103fa1296
```

kubectl:

```
kubectl get po
```

```
NAME              READY     STATUS    RESTARTS   AGE
nginx-app-5jyvm   1/1       Running   0          10m
```

```
kubectl exec nginx-app-5jyvm -- cat /etc/hostname
```

```
nginx-app-5jyvm
```

To use interactive commands.

docker:

```
docker exec -ti 55c103fa1296 /bin/sh
# exit
```

kubectl:

```
kubectl exec -ti nginx-app-5jyvm -- /bin/sh
# exit
```

For more information, see Get a Shell to a Running Container.

docker logs

To follow stdout/stderr of a process that is running, see kubectl logs.

docker:

```
docker logs -f a9e
```

```
192.168.9.1 - - [14/Jul/2015:01:04:02 +0000] "GET / HTTP/1.1"
200 612 "-" "curl/7.35.0" "-"
192.168.9.1 - - [14/Jul/2015:01:04:03 +0000] "GET / HTTP/1.1"
200 612 "-" "curl/7.35.0" "-"
```

kubectl:

```
kubectl logs -f nginx-app-zibvs
```

```
10.240.63.110 - - [14/Jul/2015:01:09:01 +0000] "GET / HTTP/1.1"
200 612 "-" "curl/7.26.0" "-"
10.240.63.110 - - [14/Jul/2015:01:09:02 +0000] "GET / HTTP/1.1"
200 612 "-" "curl/7.26.0" "-"
```

There is a slight difference between pods and containers; by default pods do not terminate if their processes exit. Instead the pods restart the process. This is similar to the docker run option `--restart=always` with one major difference. In docker, the output for each invocation of the process is concatenated, but for Kubernetes, each invocation is separate. To see the output from a previous run in Kubernetes, do this:

```
kubectl logs --previous nginx-app-zibvs
```

```
10.240.63.110 - - [14/Jul/2015:01:09:01 +0000] "GET / HTTP/1.1"
200 612 "-" "curl/7.26.0" "-"
10.240.63.110 - - [14/Jul/2015:01:09:02 +0000] "GET / HTTP/1.1"
200 612 "-" "curl/7.26.0" "-"
```

For more information, see Logging Architecture.

docker stop and docker rm

To stop and delete a running process, see kubectl delete.

docker:

```
docker ps
```

```
CONTAINER ID        IMAGE                   COMMAND
CREATED             STATUS
PORTS                         NAMES
a9ec34d98787        nginx                   "nginx -g 'daemon of"
22 hours ago        Up 22 hours             0.0.0.0:80->80/tcp, 443/
tcp   nginx-app
```

```
docker stop a9ec34d98787
```

```
a9ec34d98787
```

```
docker rm a9ec34d98787
```

```
a9ec34d98787
```

kubectl:

```
kubectl get deployment nginx-app
```

```
NAME         DESIRED    CURRENT    UP-TO-DATE    AVAILABLE    AGE
nginx-app    1          1          1             1            2m
```

```
kubectl get po -l run=nginx-app
```

```
NAME                         READY     STATUS     RESTARTS    AGE
nginx-app-2883164633-aklf7   1/1       Running    0           2m
```

```
kubectl delete deployment nginx-app
```

```
deployment "nginx-app" deleted
```

```
kubectl get po -l run=nginx-app
# Return nothing
```

> **Note:** When you use kubectl, you don't delete the pod directly.You have to first delete the Deployment that owns the pod. If you delete the pod directly, the Deployment recreates the pod.

docker login

There is no direct analog of `docker login` in kubectl. If you are interested in using Kubernetes with a private registry, see Using a Private Registry.

docker version

To get the version of client and server, see kubectl version.

docker:

```
docker version
```

```
Client version: 1.7.0
Client API version: 1.19
Go version (client): go1.4.2
Git commit (client): 0baf609
OS/Arch (client): linux/amd64
Server version: 1.7.0
Server API version: 1.19
Go version (server): go1.4.2
Git commit (server): 0baf609
OS/Arch (server): linux/amd64
```

kubectl:

```
kubectl version
```

```
Client Version: version.Info{Major:"1", Minor:"6",
GitVersion:"v1.6.9+a3d1dfa6f4335",
GitCommit:"9b77fed11a9843ce3780f70dd251e92901c43072",
GitTreeState:"dirty", BuildDate:"2017-08-29T20:32:58Z",
OpenPaasKubernetesVersion:"v1.03.02", GoVersion:"go1.7.5",
Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"6",
GitVersion:"v1.6.9+a3d1dfa6f4335",
GitCommit:"9b77fed11a9843ce3780f70dd251e92901c43072",
GitTreeState:"dirty", BuildDate:"2017-08-29T20:32:58Z",
OpenPaasKubernetesVersion:"v1.03.02", GoVersion:"go1.7.5",
Compiler:"gc", Platform:"linux/amd64"}
```

docker info

To get miscellaneous information about the environment and configuration, see kubectl cluster-info.

docker:

```
docker info
```

```
Containers: 40
Images: 168
Storage Driver: aufs
 Root Dir: /usr/local/google/docker/aufs
 Backing Filesystem: extfs
 Dirs: 248
 Dirperm1 Supported: false
Execution Driver: native-0.2
Logging Driver: json-file
Kernel Version: 3.13.0-53-generic
Operating System: Ubuntu 14.04.2 LTS
CPUs: 12
Total Memory: 31.32 GiB
Name: k8s-is-fun.mtv.corp.google.com
ID: ADUV:GCYR:B3VJ:HMPO:LNPQ:KD5S:YKFQ:76VN:IANZ:7TFV:ZBF4:BYJO
WARNING: No swap limit support
```

kubectl:

```
kubectl cluster-info
```

```
Kubernetes master is running at https://108.59.85.141
KubeDNS is running at https://108.59.85.141/api/v1/namespaces/
kube-system/services/kube-dns/proxy
kubernetes-dashboard is running at https://108.59.85.141/api/v1/
namespaces/kube-system/services/kubernetes-dashboard/proxy
Grafana is running at https://108.59.85.141/api/v1/namespaces/
kube-system/services/monitoring-grafana/proxy
Heapster is running at https://108.59.85.141/api/v1/namespaces/
kube-system/services/monitoring-heapster/proxy
InfluxDB is running at https://108.59.85.141/api/v1/namespaces/
kube-system/services/monitoring-influxdb/proxy
```

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on July 06, 2019 at 4:20 AM PST by Remove myself from review of all files except what-is-kubernetes and the deprecation policy. All of the lists of suggested reviewers need to be overhauled, but that's a separate task. (#15227) (Page History)

Edit This Page

Tools

Kubernetes contains several built-in tools to help you work with the Kubernetes system.

- Kubectl
- Kubeadm
- Kubefed
- Minikube
- Dashboard
- Helm
- Kompose

Kubectl

`kubectl` is the command line tool for Kubernetes. It controls the Kubernetes cluster manager.

Kubeadm

`kubeadm` is the command line tool for easily provisioning a secure Kubernetes cluster on top of physical or cloud servers or virtual machines (currently in alpha).

Kubefed

`kubefed` is the command line tool to help you administrate your federated clusters.

Minikube

`minikube` is a tool that makes it easy to run a single-node Kubernetes cluster locally on your workstation for development and testing purposes.

Dashboard

`Dashboard`, the web-based user interface of Kubernetes, allows you to deploy containerized applications to a Kubernetes cluster, troubleshoot them, and manage the cluster and its resources itself.

Helm

`Kubernetes Helm` is a tool for managing packages of pre-configured Kubernetes resources, aka Kubernetes charts.

Use Helm to:

- Find and use popular software packaged as Kubernetes charts
- Share your own applications as Kubernetes charts
- Create reproducible builds of your Kubernetes applications
- Intelligently manage your Kubernetes manifest files
- Manage releases of Helm packages

Kompose

`Kompose` is a tool to help Docker Compose users move to Kubernetes.

Use Kompose to:

- Translate a Docker Compose file into Kubernetes objects
- Go from local Docker development to managing your application via Kubernetes
- Convert v1 or v2 Docker Compose `yaml` files or Distributed Application Bundles

Feedback

Was this page helpful?

Yes No

Thanks for the feedback. If you have a specific, answerable question about how to use Kubernetes, ask it on Stack Overflow. Open an issue in the GitHub repo if you want to report a problem or suggest an improvement.

Analytics

Create an Issue Edit This Page
Page last modified on June 12, 2019 at 5:27 PM PST by Restructure the left navigation pane of setup (#14826) (Page History)